

MODAL EFFECT TYPES

Wenhao Tang



Doctor of Philosophy
Laboratory for Foundations of Computer Science
School of Informatics
The University of Edinburgh
2026

Abstract

Effect handlers are a powerful abstraction for defining, customising, and composing computational effects. Effect systems can statically guarantee that all effects are handled, ruling out uncaught effects at runtime. However, there are two main challenges in incorporating effect systems into mainstream programming languages. The first challenge is *complexity*. Traditional effect systems typically require extensive use of effect annotations, in particular effect variables. Retrofitting such systems into mainstream languages can therefore require pervasive changes to existing type signatures. The second challenge is *fragmentation*. There are many effect systems with different foundations in the literature and there remains a gap in understanding the precise relationship between them, which makes it hard for language designers to make design choices.

This thesis addresses these two challenges by introducing *modal effect types*, a novel approach to effect systems based on multimodal types. The key observation is that effect tracking in effect systems is typically *entangled* with other constructs such as function types. As a result, we have limited control over when and how effects are tracked in types. The key idea is to *decouple* effect tracking from function types via modalities, which offer fine-grained control over when and how effects are tracked. Modal effect types eliminate effect variables for a large class of programs, reducing the complexity of effect systems, and provide a foundation for unifying different effect systems, helping to address fragmentation of effect systems in the literature.

This thesis comprises three main parts, together with a prologue and an epilogue. The first part is **MET**, a core calculus with modal effect types. We define its syntax, type system, and operational semantics, develop several extensions, and prove type soundness and effect safety. **MET** enables ergonomic programming with higher-order functions and effects without requiring effect variables.

The second part is **MET($\mathcal{X}$)**, a uniform framework based on **MET** for encoding, analysing, and comparing effect systems. We give encodings as macro translations from existing row-based and capability-based effect systems into **MET($\mathcal{X}$)** and prove that these encodings preserve types. These encodings enable a direct analysis of their differences, and provide practical insights on language design.

The third part is **Maetel**, a surface language for **MET** with a novel bidirectional type inference algorithm **Fresco** that can infer both first-class polymorphism and first-class

modal types. The key insight of type inference is to use skeletons, which carry partial type information, to enable type information to flow back and forth between functions and arguments. *Maetel* infers all introduction and elimination of modalities, requiring no manual annotations. *Maetel* also supports implicit instantiation of polymorphic types with modal types and other polymorphic types via local type information.

Lay Summary

Computer programs do not merely calculate and return answers. They also interact with their environments, such as by reading and changing stored data, reporting errors, and communicating with other computers. Such actions that occur while a program computes its result are called *computational effects*. An *effect handler* allows one part of a program to request an effect and another part to decide how that request is dealt with. This separation can make software easier to develop and maintain. However, a program may crash if it makes a request while no handler is prepared to deal with it.

Programming languages can detect many mistakes before a program runs by assigning *types* to its parts. A type describes how a piece of code may be used. For instance, a type may say that this program wants an integer and generates a paragraph of text with words of that length. Types are extended to also describe what effects a program may request, e.g., that it may write the generated paragraph to a file or print it on the screen. Such extensions are called *effect systems*, which can check before a program runs that all effects will be properly handled. However, existing programming languages are typically designed without considering effect systems. There are many different effect systems in the literature, which typically extend existing type constructs with extra information to describe effects. As a result, adding such effect systems to a programming language can require significant rewriting of its existing code to include extra effect annotations in types, even if the code itself does not request any effects. The resulting types also become harder to read and understand for programmers.

This thesis introduces *modal effect types* to address these problems. The main idea is to use a separate construct called *modalities* to describe effects in types, instead of polluting existing types with extra effect information. Modal effect types keep track of the effects available in the surrounding context, and only require a type to be decorated with modalities if the available effects change. Code that does not request new effects can therefore keep its usual types familiar to programmers. Beyond simplifying effect systems, the contributions of this thesis also include a formal comparison of different effect systems based on modal effect types, which provides insights for language designers to choose the appropriate effect system for their programming languages, and an algorithm for automatically inferring where modalities should be inserted, further reducing the annotation burden for programmers.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

This work uses material from previously published work. The concrete relationships between individual parts of this thesis and my previous work are outlined in the introduction.

This work uses ChatGPT solely for grammar checking and for reviewing and revising part of the proofs in the appendix. Generative AI is not used for developing research ideas or writing the main thesis.

Wenhao Tang

Acknowledgements

This page is left blank intentionally to indicate that the current version of the thesis (compiled on 30th September 2026) is not the final version.

Contents

Abstract	i
Lay Summary	iii
Declaration	iv
Acknowledgements	v
Prologue	1
1 Introduction	3
1.1 The eee problem	4
1.2 Contributions	5
1.3 Modal effect types	5
1.4 Encoding effect systems	6
1.5 Inferring modal types	7
1.6 Structure of the thesis	8
2 Preliminaries	11
2.1 Algebraic effects and handlers	11
2.2 Modal types	13
2.3 Bidirectional typing	15
3 A tour of modal effect types	17
3.1 Pure programs	17
3.2 Effectful programs	19
3.3 Specifying effects	21
3.4 Handling effects	22
3.5 Composing handlers	23
3.6 Masking effects	24
3.7 Algebraic data types	26
3.8 Kinds	27

3.9	Effect polymorphism	29
3.10	Type inference	30
3.11	Maetel cheat sheet	30
I	MET the Calculus	33
4	A core calculus	35
4.1	Overview of MET	35
4.2	Syntax	37
4.3	Mode theory	38
4.3.1	Effect contexts as modes	39
4.3.2	Modalities manipulating effect contexts	39
4.4	Kinds, contexts, and equivalence	41
4.5	Typing	43
4.6	Operational semantics	45
4.7	Metatheory	46
4.8	Connections between MET and MTT	47
4.8.1	Indexed families of morphisms	48
4.8.2	The double category of effects	48
5	Extensions	51
5.1	Masks	51
5.2	Presence types	54
5.3	Crisp data types	56
5.4	Polymorphism	57
5.5	Commuting modalities and type abstraction	59
5.6	Local labels	59
5.7	Shallow handlers	61
5.8	Modality-parameterised handlers	61
5.9	Parameterised effects	63
II	MET($\mathcal{X}$) the Framework	65
6	A uniform framework	67
6.1	Syntax	69
6.2	Effect structures	70
6.3	Mode theory	73

6.4	Kinds, contexts, and equivalence	74
6.5	Typing	74
6.6	Operational semantics	76
6.7	Metatheory	77
7	Encodings	79
7.1	Overview of encodings	79
7.1.1	Encoding rows	80
7.1.2	Encoding capabilities	81
7.1.3	Comparing rows and capabilities	85
7.1.4	Encoding effect handlers	86
7.1.5	More encodings	88
7.2	Rows as modal effects	89
7.2.1	System F^ϵ	89
7.2.2	Encoding System F^ϵ into $\text{MET}(\mathcal{R}_{\text{scp}})$	91
7.2.3	System $F^{\epsilon+\text{sn}}$	92
7.2.4	Encoding System $F^{\epsilon+\text{sn}}$ into $\text{MET}(\mathcal{S})$	95
7.3	Capabilities as modal effects	97
7.3.1	System C	97
7.3.2	Encoding System C in $\text{MET}(\mathcal{S})$	100
7.3.3	System Ξ	103
7.3.4	Encoding System Ξ in $\text{MET}(\mathcal{S})$	103
7.4	Implicit effect polymorphism as simply-typed MET	105
7.4.1	System $F^{1\epsilon}$	105
7.4.2	Encoding System $F^{1\epsilon}$ in $\text{MET}(\mathcal{R}_{\text{scp}})$	107
7.5	Discussion	110
7.5.1	Insights for language design	110
7.5.2	Extensibility	111
7.5.3	Potential extensions to $\text{MET}(\mathcal{X})$	112
III	Maetel the Language	115
8	A haunted bidirectional type system	117
8.1	Overview	120
8.1.1	Conventional bidirectional type systems	120
8.1.2	Skeletons and ghosts	121
8.1.3	Colours	122

8.1.4	Modes	123
8.1.5	Flowing back and forth	123
8.1.6	Polymorphic subtyping with tags	125
8.1.7	Freezing	126
8.1.8	Features and design choices	127
8.1.9	Limitations	129
8.1.10	Comparison of type systems for FCP	130
8.2	Syntax, colours, and modes	134
8.3	Typing rules	135
8.4	Syntax-directed typing rules	136
8.5	Subtyping and consistency	137
8.6	Skeletons and ghosts	140
8.7	Skeleton inference	141
8.8	Skeleton subtyping and consistency	143
9	Type inference algorithm	147
9.1	Algorithmic syntax	147
9.1.1	Metasubstitutions	148
9.1.2	Declarative judgements in algorithmic contexts	148
9.1.3	From colours to input skeletons	149
9.2	Algorithmic type inference	151
9.3	Algorithmic skeleton inference	153
9.4	Algorithmic consistency	155
9.5	Algorithmic subtyping	158
9.6	Constraint collection and solving for subtyping	160
10	Inferring modal effect types	165
10.1	First-class modal types	165
10.2	A haunted bidirectional type system with modal effects	166
10.2.1	Syntax	167
10.2.2	Typing and skeleton inference	168
10.2.3	Subtyping and consistency	172
10.3	Type inference algorithm	175
10.3.1	Algorithmic syntax and kinding	175
10.3.2	Algorithmic type and skeleton inference	177
10.3.3	Algorithmic subtyping, consistency, and look	178
10.4	Extensions and design choices	181
10.4.1	Algebraic data types	181

10.4.2	Global functions are absolute	181
10.4.3	Modality elimination within telescopes	182
10.4.4	Masks	182
10.4.5	Parameterised effects	182
10.4.6	Effect variables	183
10.5	Implementation and elaboration	183
10.5.1	Implementation	183
10.5.2	Elaboration	183
Epilogue		187
11	Related work	189
11.1	Programming with Maetel, Koka, and Effekt	189
11.1.1	Invoking and handling effects	189
11.1.2	Data types and higher-order functions	190
11.2	Effect systems	191
11.3	Type inference	194
12	Conclusion	197
12.1	Future work	198
12.1.1	Modal types	198
12.1.2	Effect types	199
12.1.3	Type inference	201
12.2	Some final thoughts	202
Appendix		205
A	Metatheory and proofs for MET	207
A.1	Lemmas for the mode theory	207
A.2	Lemmas for the calculus	208
A.3	Progress	216
A.4	Subject reduction	219
A.5	Remarks on the other extensions	224
B	Metatheory and proofs for MET($\mathcal{X}$)	225
B.1	Lemmas for the mode theory of MET($\mathcal{X}$)	225
B.2	Lemmas for the calculus	227

B.3	Progress	229
B.4	Subject reduction	230
C	Proofs of encodings	231
C.1	Proofs of encoding System F^ϵ in $\text{MET}(\mathcal{R}_{\text{scp}})$	231
C.2	Proofs of encoding System C in $\text{MET}(\mathcal{S})$	235
C.3	Proofs of encoding System Ξ in $\text{MET}(\mathcal{S})$	238
C.4	Proofs of encoding System $F^{\epsilon+\text{sn}}$ in $\text{MET}(\mathcal{S})$	241
C.5	Proofs of encoding System $F^{1\epsilon}$ in $\text{MET}(\mathcal{R}_{\text{scp}})$	244
D	Proofs of algorithms	251
D.1	Lemmas	251
D.2	Type inference	252
D.3	Skeleton inference	265
D.4	Subtyping	274
D.4.1	Decorated algorithmic judgements	275
D.4.2	Soundness of subtyping	279
D.4.3	Completeness of subtyping	295
D.5	Consistency	316
D.5.1	Lemmas for subtyping	316
D.5.2	Soundness of consistency	333
D.5.3	Completeness of consistency	341
	Bibliography	353

List of Figures

4.1	Syntax of MET	37
4.2	Mode theory of MET	38
4.3	Kinding and equivalence rules of types for MET	42
4.4	Well-formedness and equivalence rules of contexts for MET	42
4.5	Typing rules for MET	44
4.6	Operational semantics for MET	46
5.1	Masks.	52
5.2	Presence types.	55
5.3	Pairs and sums.	56
5.4	Polymorphism.	58
5.5	Commuting modalities and type abstractions.	59
5.6	Local labels.	60
5.7	Shallow handlers.	61
5.8	Modality-parameterised handlers.	62
5.9	Parameterised effects.	64
6.1	Syntax of MET ($\mathcal{X}$).	69
6.2	The effect structure $\mathcal{S}$ (sets).	71
6.3	The effect structure $\mathcal{S}_{\text{mult}}$ (multisets).	72
6.4	The effect structure $\mathcal{R}_{\text{scp}}$ (scoped rows).	72
6.5	The effect structure $\mathcal{R}_{\text{simp}}$ (simple rows).	72
6.6	Kinding, equivalence, and context well-formedness rules for MET ($\mathcal{X}$).	75
6.7	Typing rules for MET ($\mathcal{X}$).	76
6.8	Operational semantics for MET ($\mathcal{X}$).	77
7.1	Syntax and typing rules for System F^ϵ	90
7.2	Operational semantics of System F^ϵ	90
7.3	An encoding of System F^ϵ in MET ($\mathcal{R}_{\text{scp}}$).	91
7.4	Syntax of System $F^{\epsilon+\text{sn}}$	93
7.5	Typing rules for System $F^{\epsilon+\text{sn}}$	94
7.6	Operational semantics for System $F^{\epsilon+\text{sn}}$	95

7.7	An encoding of System F^{e+sn} in $MET(S)$	96
7.8	Syntax and typing rules for System C.	98
7.9	Operational semantics for System C.	100
7.10	An encoding of System C in $MET(S)$	101
7.11	Syntax and typing rules for System Ξ	103
7.12	An encoding of System Ξ in $MET(S)$	104
7.13	Syntax and typing rules for System $F^{1\epsilon}$	106
7.14	Encoding of System $F^{1\epsilon}$ in $MET(\mathcal{R}_{scp})$	108
8.1	Type signatures for functions used in the comparison.	131
8.2	Comparison of type systems for FCP.	132
8.3	Syntax of Fresco.	134
8.4	Declarative typing rules for Fresco.	135
8.5	Syntax-directed declarative typing rules for Fresco.	137
8.6	Syntax and well-formedness rules for tagged types.	138
8.7	Subtyping and consistency rules for Fresco.	138
8.8	Syntax, well-formedness, and refinement rules for skeletons.	140
8.9	Skeleton inference rules for Fresco.	142
8.10	Syntax and well-formedness rules for decorated skeletons.	143
8.11	Subtyping and consistency rules of skeletons for Fresco.	144
9.1	Algorithmic syntax.	148
9.2	Metasubstitutions and equivalence of metasubstitutions.	149
9.3	Declarative judgements in algorithmic contexts.	150
9.4	Algorithmic type inference and skeleton splitting.	152
9.5	Algorithmic skeleton inference.	154
9.6	Algorithmic consistency.	157
9.7	Algorithmic subtyping.	159
9.8	Look and constraint collection.	163
9.9	Constraint solving.	164
10.1	Syntax of Maetel.	167
10.2	Kinding and refinement rules for skeletons in Maetel.	168
10.3	Declarative typing rules for Maetel.	169
10.4	Syntax-directed typing rules for Maetel.	171
10.5	Skeleton inference rules for Maetel.	171
10.6	Syntax and well-formedness rules for tagged types in Maetel.	172
10.7	Subtyping and consistency rules for Maetel.	173

List of Figures

10.8	Syntax and well-formedness rules for tagged skeletons in Maetel	174
10.9	Skeleton subtyping and consistency rules for Maetel	175
10.10	Algorithmic syntax of Maetel	176
10.11	Algorithmic kinding rules for flexible variables in Maetel	176
10.12	Changed type inference rules for Maetel	177
10.13	Changed algorithmic subtyping and consistency rules for Maetel	179
10.14	Changed constraint-collection rules for Maetel	180
10.15	Changed constraint-solving rules for Maetel	180
10.16	Elaboration from Maetel to MET	185
10.17	Elaboration of Maetel subtyping to MET	186
D.1	Decorated algorithmic subtyping.	276
D.2	Decorated algorithmic consistency.	277
D.3	Decorated tag solving.	278

Prologue

1 Introduction

Shall I be pure or impure?

—Philip Wadler ([1992](#))

Functional programmers have long enjoyed the benefits of *higher-order functions*, which may take other functions as arguments and return them as results. For instance, a standard uncurried `map` function applies a function to every element of a list:

```
map (f, [])      = []
map (f, x :: xs) = f x :: map (f, xs)
```

Its familiar type describes the values that `map` takes and returns:

```
map : forall a b . (a → b) * List a → List b
```

The function `map` takes a function from `a` to `b` and a list of `as`, and returns a list of `bs`.

This `map` function does not itself raise exceptions, access mutable state, perform input or output, or communicate with concurrent processes. These behaviours are known as *computational effects*, or just *effects* [[Moggi 1991](#)]. Although `map` performs none of them directly, the function `f` may do so whenever it is applied to an element of the list. For example, consider the effect `IO`, with operations `read` and `print` for reading and printing strings. We can apply `map` to an effectful identity function that prints `Hello world`:

```
map ((fun x → do print "Hello world"; x), xs)
```

This application prints `Hello world` the same number of times as there are elements in `xs`. As we see, the effects of applying `map` depend on the function passed to it.

Programs may crash at runtime due to unhandled or mishandled effects. An *effect system* extends a type system to track effects in types, statically guaranteeing that every effect is handled properly. A common approach is to annotate function arrows with the effects that may occur when a function is invoked. For instance, for the example above, since the argument only performs `IO`, we can give `map` the following type:

```
map : forall a b . (a →{IO} b) * List a →{IO} List b
```

The annotation following each arrow is an *effect type*. Here, it contains the single effect `IO`. This type permits the example above, but restricts the function passed to `map` to

performing only `IO`. This leads to a modularity issue: if we want to pass an argument that performs other effects, such as `Exception` or `State`, to `map`, we must give it a different type.

The traditional solution is to use *parametric effect polymorphism*. Effect polymorphism uses effect variables to quantify over effect types. For example, we can give `map` a type:

```
map : forall a b e . (a ->{e} b) * List a ->{e} List b
```

The effect variable `e` can be instantiated with an effect type. In the example above, `e` may be instantiated with `IO` or with any effect type containing `IO`, such as `IO`, `State`.

So far, the story seems complete. We encountered a modularity issue with effect types, and resolved it with parametric polymorphism. What, then, is the problem?

1.1 The eee problem

The problem arises precisely from the complexity caused by effect variables. The same effect variable `e` appears three times in the above type of `map`, even though the definition of `map` does not itself perform any effects at all. Any effects performed during the application of `map` come from its argument, rather than from `map` itself. We refer to this as the *eee problem*, a term coined by Philip Wadler and then first used by Martin Odersky at the 44th meeting of IFIP Working Group 2.8 [[IFIP Working Group 2.8 2026](#)].

The eee problem breaks backwards compatibility. Even for higher-order functions that do not themselves perform any effects, we have to change their type signatures to include effect variables. This is particularly problematic when retrofitting an effect system to an existing language such as OCaml, which already has a large codebase, including libraries in which higher-order functions are widely used. We would need significant rewriting of the type signatures of existing functions to add effect variables, even though they have nothing to do with effects. Moreover, effect variables add a mental burden when reading, understanding, and reasoning about types. A programmer must follow several occurrences of `e` merely to recover the simple fact that `map` passes on the effects of its argument and does not itself perform any effects.

Why not instead track the fact that `map` itself is pure? From a contextual view of effects, it is obvious that the definition of `map` requires no effects from the context. However, we cannot express this fact directly in a conventional effect system because effect types are *entangled* with function arrows. Each arrow describes the effects that may occur when the corresponding function is invoked. For instance, if we leave all function arrows in the type of `map` unannotated, as in its original type, they would be interpreted as pure. As a result, we cannot apply `map` to any effectful argument.

This thesis *decouples* effect types from function arrows, giving `map` its familiar type

```
map : forall a b . []((a → b) * List a → List b)
```

where the syntax `[]`, which can be omitted for global definitions such as this one, is a *modality* that states `map` itself requires no effects. We can still apply `map` to effectful arguments.

1.2 Contributions

This thesis contributes *modal effect types*, a new effect system that decouples effect types from function arrows and tracks effects with modalities. In particular, this thesis makes three main contributions:

- MET, a core calculus for modal effect types;
- MET($\mathcal{X}$), a framework based on MET for encoding and comparing effect systems;
- Maetel, a surface language with type inference for modal effect types.

The next three sections highlight the problems, insights, and results of each contribution. Effect systems and type inference are both active and established research areas in programming languages research. There are a few approaches that also target the eee problem. A detailed discussion of related work is given in Chapter 11.

1.3 Modal effect types

We design a core calculus MET for modal effect types. MET decouples effect types from function arrows, solving the eee problem of Section 1.1. MET achieves this decoupling by tracking, for each term, the ambient *effect context*, which records the effects provided by the context, and by using modalities to track changes to effect contexts in types. For instance, let us revisit the type of `map` above:

```
map : forall a b . []((a → b) * List a → List b)
```

The `[]` is the empty *absolute modality*, which states that the effect context of the inside term is empty, i.e., `map`'s definition requires no effects, regardless of what the ambient effect context is. When we apply `map`, since we know it is well-typed under the empty effect context, we may safely apply it under any ambient effect context, so that its argument may perform any effects provided by the context. For instance, let us revisit the application from the beginning of this chapter:

```
map ((fun x → do print "Hello world"; x), xs)
```

The ambient effect context for this term must contain `IO` as the argument uses the `print` operation. We can apply `map` here since its type tells us it requires no particular effects. In addition to the `[]` modality, MET supports absolute modalities `[E]` and relative modalities `<D>` that specify what effects are used or handled.

MET supports *algebraic effects and handlers* [Plotkin and Power 2003; Plotkin and Pretnar 2013], which provide a general way to define, customise, and compose computational effects. In addition to absolute modalities, MET also has *relative modalities*, which describe a change to the ambient effect context. They are particularly useful for effect handlers that handle some specific effect while forwarding other effects to the surrounding context.

The design of the MET calculus is based on the recent work on multimodal type theory (MTT) [Gratzer et al. 2020; Gratzer 2023], which is a dependent type theory that supports multiple modes and modalities. Modes refine types and terms, with modalities transforming between modes. MET extends and generalises the simply-typed fragment of MTT. We define MET, prove the correctness of its metatheory including type soundness and effect safety, and extend MET with more features.

1.4 Encoding effect systems

Effect systems [Lucassen and Gifford 1988; Wadler 1998] in the literature are fragmented: there are many systems with different foundations, and the relationships between them are not well understood. The most conventional approach, as we have seen before, associates effect types with function arrows. In these systems, effect types are often organised as *row types* [Wand 1987; Rémy 1989].¹ We refer to this approach as *row-based effect systems*. In general, a function type in a row-based effect system has the form:

$$A \rightarrow^E B$$

Here, E specifies the effects that may occur when the function is invoked.

Another popular approach is to treat effects as *capabilities* [Dennis and Horn 1966; Miller 2006] provided by the context, and track capabilities in types. We refer to such systems as *capability-based effect systems*. In the capability-based effect system of Brachthäuser et al. [2022], a function type has the form:

$$(A, f : T) \Rightarrow B.$$

¹Or sets, which are a special case of rows.

This function binds a value of type A and a capability f of type T for use in its function body which returns a value of type B .

These two styles of effect systems differ fundamentally in how they track effects. In a row-based effect system, a function type always describes the effects that it may perform. In a capability-based effect system, a function type binds capabilities and may mention these capabilities in its body. Some capability-based effect systems further restrict functions to second-class values to prevent capabilities from escaping. It is difficult to compare these two styles of effect systems directly. We encounter the same challenge as in the *eee* problem: these effect systems entangle effect tracking with function types. Any direct attempt to encode a function type from one system to another must reconcile the differences between effect tracking mechanisms built into the function types.

Modal effect types serve as a common target for encoding different effect systems since they decouple effect tracking from function types. We develop $\text{MET}(\mathcal{X})$ based on MET as a uniform framework for encoding and comparing different effect systems. $\text{MET}(\mathcal{X})$ is parameterised by an *effect structure* $\mathcal{X}$, which determines how effect collections are organised, so that we can focus on encoding the effect tracking mechanisms instead of dealing with the details of effect collections.

We present $\text{MET}(\mathcal{X})$, use it to encode row-based and capability-based effect systems, and prove that the encodings preserve typing. Our encodings enable a direct analysis of the differences between row-based and capability-based effect systems and provide practical insights into language design. We also show that a fragment of a row-based effect system with effect polymorphism can be encoded by modal effect types without any effect variables. This encoding provides formal support for the argument that modal effect types solve the *eee* problem.

1.5 Inferring modal types

Merely having the core calculus MET is not enough to use modal effect types in practice. As with other explicit core calculi, MET requires annotations on lambda abstractions and explicit terms for type abstractions and applications. Moreover, MET requires explicit terms for introducing and eliminating modalities. We want a surface language where all these annotations and terms can be omitted.

Type inference for MET is reminiscent of type inference for *first-class polymorphism* (FCP). FCP combines higher-rank polymorphism, where quantifiers may appear beneath function arrows, with polymorphic instantiation, where a type variable may be instantiated with a polymorphic type. Following this definition, we introduce the notion

of *first-class modal types*, which allows modalities to appear anywhere in a type and allows a type variable to be instantiated with a modal type.

Type inference for FCP has been studied extensively. Full type inference for System F is undecidable [Pfenning 1993; Wells 1999]. Existing approaches impose different restrictions or extensions on the type system, adopt different heuristics, or develop different constraint-solving algorithms [Garrigue and Rémy 1999; Le Botlan and Rémy 2003; Leijen 2008; Serrano et al. 2020; Emrich et al. 2020].

We propose *Fresco*, an approach to type inference for FCP that stays with System F types, has low syntactic overhead and predictable behaviour, and does not rely on sophisticated algorithms. We generalise this approach to *Maetel*, a surface language with modal effect types that supports both first-class polymorphism and first-class modal types. The central idea is to use bidirectional information flow between functions and arguments. Information from an argument may guide how a function is instantiated, while information from a function may guide the inference of polymorphic and modal types for its arguments. To allow information to flow in both directions within an application we introduce *skeletons*, which record known type information. Skeletons contain *ghosts* that represent unknown information. Skeletons first flow from arguments to a function to guide its inference and instantiation. The function then supplies the ghosts in skeletons with more information, which flows back to the arguments. We also introduce a *freezing* operator for programmers to control the direction of information flow.

We give a declarative type system and a type inference algorithm for *Fresco*, and prove soundness and completeness of the algorithm with respect to the declarative specification. We also show how *Maetel* extends *Fresco* to infer modal effect types.

1.6 Structure of the thesis

The thesis consists of a prologue, three main parts, and an epilogue. Each main part develops one of the three contributions described above.

Prologue contains this introduction, the preliminaries in Chapter 2, and a tour of programming with modal effect types in Chapter 3. The preliminaries review algebraic effects and handlers, modal types, and bidirectional typing. The tour then introduces the main ideas and programming patterns of modal effect types through examples in *Maetel*.

Part I introduces the calculus *MET*. Chapter 4 defines the syntax, mode theory, kind system, type system, and operational semantics of *MET*, and establishes its type soundness and effect safety. Chapter 5 develops extensions to *MET*, including masks, presence

types, parameterised effects, data types, polymorphism, local labels, and richer forms of handlers.

Part II introduces the framework $\text{MET}(\mathcal{X})$. Chapter 6 defines $\text{MET}(\mathcal{X})$, including effect structures, and establishes its type soundness and effect safety. Chapter 7 gives compositional type-preserving encodings of row-based and capability-based effect systems into $\text{MET}(\mathcal{X})$.

Part III introduces the language Maetel with Fresco type inference. Chapter 8 defines a declarative type system for Fresco. Chapter 9 develops a type inference algorithm for Fresco and establishes its soundness and completeness with respect to the declarative type system. Chapter 10 presents the design of Maetel, which extends Fresco to infer first-class modal effect types.

Epilogue discusses related work in Chapter 11 and concludes the thesis in Chapter 12.

Appendices contain proofs of the theorems in the metatheory of the calculi and the type inference algorithm in the main parts.

Relationship to prior work. Parts I and II are a substantial revision, simplification, and extension of the materials in the papers [Tang et al. 2025; Tang and Lindley 2026]:

- Wenhao Tang, Leo White, Stephen Dolan, Daniel Hillerström, Sam Lindley, and Anton Lorenzen. *Modal Effect Types*. In *Proceedings of the ACM on Programming Languages*, volume 9 (OOPSLA1), pages 1130–1157, 2025.
- Wenhao Tang and Sam Lindley. *Rows and Capabilities as Modal Effects*. In *Proceedings of the ACM on Programming Languages*, volume 10 (POPL), pages 923–950, 2026.

Chapters 8 and 9 of Part III are based on the materials in the paper [Tang et al. 2026]:

- Wenhao Tang, Shengyi Jiang, Aghilas Y. Boussaa, Sam Lindley, and Bruno C. d. S. Oliveira. *Bidirectional Typing with Freezing, Skeletons, and Ghosts*. Preprint, 2026.

Chapter 10 of Part III is new material.

In addition to the above papers covered in this thesis, the author also worked on other papers on types and effects during his PhD [Tang et al. 2024, 2023; Bosman et al. 2024; Tang and Schrijvers 2024]:

- Wenhao Tang, Daniel Hillerström, James McKinna, Michel Steuwer, Ornela Dardha, Rongxiao Fu, and Sam Lindley. *Structural Subtyping as Parametric Polymorphism*. In *Proceedings of the ACM on Programming Languages*, volume 7 (OOPSLA2), pages 1093–1121, 2023.

1 Introduction

- Wenhao Tang, Daniel Hillerström, Sam Lindley, and J. Garrett Morris. *Soundly Handling Linearity*. In *Proceedings of the ACM on Programming Languages*, volume 8 (POPL), article 54, 2024.
- Roger Bosman, Birthe van den Berg, Wenhao Tang, and Tom Schrijvers. *A Calculus for Scoped Effects & Handlers*. In *Logical Methods in Computer Science*, volume 20, issue 4, 2024.
- Wenhao Tang and Tom Schrijvers. *From High to Low: Simulating Nondeterminism and State with State*. In *Journal of Functional Programming*, volume 34, e16, 2025.

2 Preliminaries

The meanings of simple signs (words) must be explained to us if we are to understand them.

—Ludwig Wittgenstein (1921)

We explain the meanings of some basic concepts this thesis builds on, including algebraic effects and handlers, modal types, and bidirectional typing. Effect handlers and modal types are the two main features of our calculi in Parts I and II. Bidirectional typing is the basis of our type inference approach in Part III.

2.1 Algebraic effects and handlers

In order to express effects in our calculi, we use *algebraic effects and handlers* [Plotkin and Power 2003; Plotkin and Pretnar 2013], an expressive language feature which provides a modular way to define and compose effects. Modal effect types are not restricted to effect handlers; the idea is independent of how effects are implemented in a language.

Algebraic effects and handlers achieve modularity through separating syntax and semantics of effects. Effects can be freely composed syntactically, while an effect handler interprets some selected effect and leaves other effects to other handlers, generalising standard exception handlers. An effect is described by a collection of operations. For example, the `IO` effect from Chapter 1 provides two operations for reading and printing strings:

```
eff IO = read : 1 ⇒ String, print : String ⇒ 1
```

The signature `1 ⇒ String` of `read` states that `read` takes a unit and returns a string. Similarly, the signature `String ⇒ 1` of `print` states that `print` takes a string and returns a unit. For example, the following program reads a string, prepends "hello " to it, and prints it:

```
hello () = do print ("hello " ++ do read ())
```

Invoking an operation transfers control to a handler that handles this operation. For example, this simple handler supplies a fixed result "world" for `read` and discards strings passed to `print`:

2 Preliminaries

```
mockIO f = handle f () with
  return x    ⇒ x
  read () r   ⇒ r "world"
  print s r   ⇒ r ()
```

This handler takes a function f , invokes this function and handles its `IO` operations. The handler has three clauses. The first return clause binds the return value as x and directly returns it. The remaining two are operation clauses, which bind the argument of the operation as the first parameter and bind the continuation of the operation as the second parameter r . Calling r resumes the computation with the provided result.

As a more interesting example, we implement a generator with effect handlers. We use a `Gen` effect with one operation for yielding integers:

```
eff Gen = yield : Int ⇒ 1
```

The following handler collects all yielded numbers into a list.

```
asList f = handle f () with
  return () ⇒ nil
  yield x r ⇒ cons x (r ())
```

The return clause returns the empty list. The operation clause prepends the integer x onto the head of the list returned by the continuation r . For example, `asList` collects three yielded integers as follows:

```
> asList (fun () → do yield 6; do yield 37; do yield 42)
# [6, 37, 42] : List Int
```

We can also implement mutable state with effect handlers. A state effect is parameterised by the type of the state s and has two operations:

```
eff State s = get : 1 ⇒ s, put : s ⇒ 1
```

The following implements mutable state in a parameter-passing style:

```
state m = handle m () with
  return x ⇒ fun s → x
  get () r ⇒ fun s → r s s
  put s' r ⇒ fun s → r () s'
```

The handler transforms a stateful computation into a function that takes the current state. The return clause does not use the current state. The `get` clause resumes the continuation with the current state and makes no change to it. The `put` clause resumes the continuation with unit and replaces the current state with the new state s' . For example, we start with an initial state 37 and increment it by 5:

```
> state (fun () → do put (do get () + 5); do get ()) 37
# 42 : Int
```

For a gentle introduction to algebraic effects and handlers, I recommend Matija Pretnar’s tutorial *An Introduction to Algebraic Effects and Handlers* [Pretnar 2015]. For a broader collection of papers and other resources, see the community-maintained effects bibliography [Yallop 2026].

2.2 Modal types

Modal types arise from the propositions-as-types correspondence for modal logic. Modal type systems enrich ordinary type systems with modalities, which control how variables may be used, e.g., how many times, under which conditions, or in which scope. Classic examples are the *necessity* modality $\Box$ and the *possibility* modality $\Diamond$. In Kripke semantics with some given model, a proposition $\Box P$ is true in a world if P is true in all accessible worlds, while a proposition $\Diamond P$ is true in a world if P is true in some accessible world.

To provide some intuition for what a modal type system looks like, we present a simply-typed lambda calculus with the necessity modality $\Box$ from the modal logic IS4. The necessity modality $\Box$ in the modal logic IS4 is used to characterise staged computation in programming languages [Davies and Pfenning 2001]. There are different styles for formulating modal type systems. We follow the style of multimodal type theory (MTT) [Gratzer et al. 2020], which adopts *locks* from Fitch-style modal lambda calculi [Clouston 2018] and uses *let-style* elimination for modalities. The typing rules are as follows.

$$\begin{array}{c}
\text{T-VAR} \\
\frac{\mathbf{\boxtimes} \notin \Gamma'}{\Gamma, x : A, \Gamma' \vdash x : A} \\
\\
\text{T-LETMOD} \\
\frac{\Gamma \vdash M : \Box A \quad \Gamma, x : \Box A \vdash N : B}{\Gamma \vdash \text{let mod } x = M \text{ in } N : B} \\
\\
\text{T-VARMODAL} \\
\frac{}{\Gamma, x : \Box A, \Gamma' \vdash x : A} \\
\\
\text{T-ABS} \\
\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x^A. M : A \rightarrow B} \\
\\
\text{T-MOD} \\
\frac{\Gamma, \mathbf{\boxtimes} \vdash M : A}{\Gamma \vdash \text{mod } M : \Box A} \\
\\
\text{T-APP} \\
\frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash M N : B}
\end{array}$$

The first four rules are interesting. Rule T-MOD introduces the necessity modality $\Box$ and puts a lock in the context for M . Contexts are ordered since we have locks. These locks control how variables may be used. In rule T-VAR, we cannot use a variable x if there is a lock in the context to the right of x . We can only use those variables that are annotated with the modality $\Box$ as in rule T-VARMODAL. These variables come from modality elimination as in rule T-LETMOD, which eliminates the modality of M and

annotates the binding of x with the modality $\Box$. As a result, we have the guarantee that inside a necessity modality, we can only use variables that come from another necessity modality. This exactly characterises the intuition of staged computation: a term of type $\Box A$ is a piece of generated code that can only use other pieces of generated code, and cannot use any ordinary values from the current stage. Another classic example is the bang modality $!$ from linear logic, which is used in linear type systems where a type A is linear by default and $!A$ is unrestricted [Girard 1987; Wadler 1991].

We only have one modality in the above calculus. Graded modal type systems generalise a single modality to a family of modalities indexed by grades r [Orchard et al. 2019; Abel and Bernardy 2020], which are usually specified by a semiring-like structure. For instance, graded modalities can refine the bang modality to $!_r$ which tracks that a value should be used r times.

Modes assign types and terms to different syntactic categories. There is only one mode in graded modal type systems. Each modality is an endomorphism on the single mode. Multimodal type theory further supports multiple modes, in addition to multiple modalities between them. It is parameterised by a *mode theory*, which specifies modes m, n , modalities $\mu : m \rightarrow n$, and modality transformations $\alpha : \mu \Rightarrow \nu$. A mode theory is a 2-category, where modes are objects, modalities are morphisms, and modality transformations are 2-cells. A typing judgement has the form $\Gamma \vdash M : A @ m$, stating that M has type A under Γ at mode m .

The core calculus MET in Chapter 4 is based on MTT, where modes represent effect contexts. We assume no prior knowledge of MTT when presenting MET. The following is a brief introduction to the typing rules of simply-typed MTT for interested readers; it is not necessary for understanding the rest of the thesis.

A modality moves types and terms between modes. If $\mu : m \rightarrow n$ and A is a type at mode m , then μA is a type at mode n . The typing rules for introduction and elimination of modalities are as follows, similar to the rules for the necessity modality above.

$$\begin{array}{c}
 \text{MTT-MOD} \\
 \hline
 \mu : m \rightarrow n \quad \Gamma, \mathbf{lock}_\mu \vdash M : A @ m \\
 \hline
 \Gamma \vdash \text{mod}_\mu M : \mu A @ n
 \end{array}
 \qquad
 \begin{array}{c}
 \text{MTT-LETMOD} \\
 \hline
 \Gamma \vdash M : \mu A @ n \quad \Gamma, x :_\mu A \vdash N : B @ n \\
 \hline
 \Gamma \vdash \text{let mod}_\mu x = M \text{ in } N : B @ n
 \end{array}$$

Introducing a modality μ adds a lock annotated by μ . Eliminating a modality μ binds its contents as $x :_\mu A$ annotated by μ .¹ Ordinary type constructors are *mode-local*. For instance, if $A \rightarrow B$ is a type at mode m , then both A and B are types at m .

¹The full syntax of modality elimination has an extra modality annotation which enables sequential eliminations. We omit it here for simplicity and reintroduce it in Chapter 4.

As in the variable rules above, the variable rule in MTT controls accessibility with locks and modality annotations on bindings:

$$\text{MTT-VAR} \quad \frac{\mu : m \rightarrow n \quad \nu = \text{locks}(\Gamma') : m \rightarrow n \quad \alpha : \mu \Rightarrow \nu}{\Gamma, x :_{\mu} A, \Gamma' \vdash x : A @ m}$$

Here, $\text{locks}(\Gamma')$ is the modality obtained by composing the modalities of the locks in the context Γ' . We want to use a variable $x :_{\mu} A$ after locks whose modalities compose to ν . The use is permitted when the mode theory provides a transformation $\alpha : \mu \Rightarrow \nu$.

We can instantiate MTT to recover the typing rules for the necessity modality $\Box$ above. There is only one mode $\bullet$. There are two modalities: the necessity modality $\Box : \bullet \rightarrow \bullet$ and the identity modality $\mathbb{1} : \bullet \rightarrow \bullet$, satisfying:

$$\mathbb{1} \circ \mu = \mu = \mu \circ \mathbb{1} \quad \Box \circ \Box = \Box.$$

We have two identity modality transformations $\mathbb{1} \Rightarrow \mathbb{1}$ and $\Box \Rightarrow \Box$, ignoring their names. The only non-identity transformation is $\Box \Rightarrow \mathbb{1}$; in particular, there is no transformation $\mathbb{1} \Rightarrow \Box$. The interesting typing rules are as follows.

$$\begin{array}{ccc} \text{T-VAR} & \text{T-MOD} & \text{T-LETMOD} \\ \frac{\mu \Rightarrow \text{locks}(\Gamma')}{\Gamma, x :_{\mu} A, \Gamma' \vdash x : A @ \bullet} & \frac{\Gamma, \blacksquare_{\mu} \vdash M : A @ \bullet}{\Gamma \vdash \text{mod}_{\mu} M : \mu A @ \bullet} & \frac{\Gamma \vdash M : \mu A @ \bullet \quad \Gamma, x :_{\mu} A \vdash N : B @ \bullet}{\Gamma \vdash \text{let mod}_{\mu} x = M \text{ in } N : B @ \bullet} \end{array}$$

The single variable rule recovers the two variable rules above. An ordinary binding is by default annotated with $\mathbb{1}$ and cannot cross a $\Box$ -lock because there is no transformation $\mathbb{1} \Rightarrow \Box$. A modal binding is annotated with $\Box$ and can always be used since we have both $\Box \Rightarrow \Box$ and $\Box \Rightarrow \mathbb{1}$.

For a gentle introduction to simply-typed MTT which is enough for understanding the relationship between MTT and this thesis, I recommend Kavvos and Gratzer’s paper *Under Lock and Key: A Proof System for a Multimodal Logic* [Kavvos and Gratzer 2023]. For a thorough introduction to MTT, I recommend Daniel Gratzer’s PhD thesis *Syntax and Semantics of Modal Type Theory* [Gratzer 2023].

2.3 Bidirectional typing

Bidirectional type systems split typing into inference mode I, where we infer the type of a term, and checking mode C, where we check a term against a given type. The usage of the word “mode” here has nothing to do with the usage of the word “mode” in

modal type systems. We write the type inference judgement as $\Gamma \vdash_I M : A$ and the type checking judgement as $\Gamma \vdash_C M : A$. For the simply-typed lambda calculus, the typing rules are as follows.

$$\begin{array}{c}
 \begin{array}{c} \text{T-VAR} \\ \hline \Gamma \ni x : A \\ \hline \Gamma \vdash_I x : A \end{array}
 \quad
 \begin{array}{c} \text{T-ANNO} \\ \hline \Gamma \vdash_C M : A \\ \hline \Gamma \vdash_I (M : A) : A \end{array}
 \quad
 \begin{array}{c} \text{T-SWITCH} \\ \hline \Gamma \vdash_I M : A \\ \hline \Gamma \vdash_C M : A \end{array}
 \quad
 \begin{array}{c} \text{T-ABS} \\ \hline \Gamma, x : A \vdash_C M : B \\ \hline \Gamma \vdash_C \lambda x. M : A \rightarrow B \end{array}
 \\
 \\
 \begin{array}{c} \text{T-APP} \\ \hline \Gamma \vdash_I M : A \rightarrow B \quad \Gamma \vdash_C N : A \\ \hline \Gamma \vdash_I M N : B \end{array}
 \end{array}$$

A variable infers its type from the context. A type annotation checks its term against the annotated type and infers the annotated type. Rule T-SWITCH switches from checking mode to inference mode. A lambda abstraction is checked against a function type. An application infers the function type and uses its argument type to check the argument.

For a detailed introduction to bidirectional typing, I recommend Dunfield and Krishnaswami's survey *Bidirectional Typing* [[Dunfield and Krishnaswami 2022](#)].

3

A tour of modal effect types

Everyone embarks on a journey dreaming of the Sea of Stars.

—Opening lines, *Galaxy Express 999* (1979)

Your journey has just begun.

—Maetel, *Galaxy Express 999* (1979)

This chapter is a self-contained tutorial on programming in Maetel, our surface language with modal effect types, from a programmer’s perspective. Maetel is a functional programming language with Haskell-like syntax, algebraic data types, effect handlers, modal effect types, and type inference for both first-class polymorphism and first-class modal types. We assume the readers are functional programmers with basic knowledge of effect handlers as introduced in Section 2.1. We assume no background knowledge of programming language theory because we hope that modal effect types can be a practical type-system feature that improves programmers’ experience. We illustrate how modal effect types stay close to ordinary functional programming while extending the types in a modular and conservative way to track effects, providing effect safety and serving as useful documentation for programmers. In particular, most types in Maetel retain their familiar shape; modalities mainly appear either at outermost positions for specifying which effects are used, or where effects are handled.

For programmers who are familiar with other programming languages with effect systems such as Koka [Leijen 2026] and Effekt [Brachthäuser et al. 2026], we provide a direct comparison between programs in these languages in Section 11.1.

I do not yet have a full implementation of Maetel. Aghilas Y. Boussaa implemented a surface language with many of the features of Maetel [Boussaa 2026]. His implementation uses ML-like syntax instead of the Haskell-like syntax of Maetel.

3.1 Pure programs

We begin with ordinary, familiar pure programs. A prefix of π as a list:

3 A tour of modal effect types

```
pi : List Int
pi = [3,1,4,1,5,9]
```

A first-order function `inc` increments an integer:

```
inc : Int → Int
inc x = x + 1
```

A pair containing two copies of `inc`:

```
pinc : (Int → Int) * (Int → Int)
pinc = (inc, inc)
```

An uncurried higher-order function `map`, as in Chapter 1, maps a function over a list:

```
map : forall a b . (a → b) * List a → List b
map (f, nil)      = nil
map (f, cons x xs) = cons (f x) (map (f, xs))
```

A curried higher-order function `iter` iterates a function over a list:

```
iter : forall a . (a → 1) → List a → 1
iter f nil      = ()
iter f (cons x xs) = f x; iter f xs
```

We write `1` for the unit type, whose only value is `()`. Maetel adopts bidirectional typing and requires all global definitions to be accompanied by type signatures.

All the above programs are *pure*; they do not perform any effects themselves. Whenever we say a program is pure in this chapter, we always mean that the program itself performs no effects that the surrounding context must handle, or equivalently, the program requests no effects to run. Maetel keeps their ordinary types as above. Since we know they are pure, we can do a bit more, wrapping their types in `[]` to document that they require no effects:

```
pi   : [](List Int)
inc  : [](Int → Int)
pinc : []((Int → Int) * (Int → Int))
map  : [](forall a b . (a → b) * List a → List b)
iter : [](forall a . (a → 1) → List a → 1)
# alternatively, for map and iter:
map  : forall a b . []((a → b) * List a → List b)
iter : forall a . []((a → 1) → List a → 1)
```

The syntax `[]` is the empty *absolute modality*. It says that the program inside uses no effects, as we can easily verify from the definitions of these functions. Modalities have

higher precedence than function arrows. Annotating types of pure programs with top-level `[]` is recommended as it provides useful documentation for programmers and informs the type checker that these programs can be used wherever we want, regardless of what effects are available. Since Maetel is a call-by-value language, the modality `[]` is redundant for `pi` as a value of type `List Int` can never perform any effects.

Maetel requires global programs to require no effects from the context in order to guarantee that programs would not get stuck due to unhandled effects. Furthermore, Maetel allows omitting this top-level `[]` from a global type signature, so the type signatures of earlier definitions are implicitly wrapped in `[]` when they are global definitions. We will always keep top-level `[]` explicit in the rest of this chapter.

Programmers familiar with traditional effect types may intend to annotate every function arrow in the types of `pinc`, `map`, and `iter` with `[]`. They would still type check, but unnecessarily restrict the arguments of `map` and `iter` to also be pure functions. The modality `[]` we added before only requires the definitions of `map` and `iter` to be pure, but still allows their arguments to be effectful, as we will see in the next section.

3.2 Effectful programs

Pure programs only compute results from their inputs. Effectful programs may also interact with their context, for example, by printing messages, yielding values, updating state, or making nondeterministic choices. Maetel uses the `eff` syntax to declare effects and their operations with signatures:

```
eff IO      = read : 1 ⇒ String | print : String ⇒ 1
eff Gen [a] = yield : a ⇒ 1
eff State [a] = get : 1 ⇒ a | put : a ⇒ 1
eff ND      = choose : 1 ⇒ Bool | fail : 1 ⇒ [](forall a . a)
```

An operation signature $A \Rightarrow B$ types operations that take an argument of type `A` and return a result of type `B`. We write $\Rightarrow$ to distinguish operation signatures from function arrows. The `IO` effect provides operations for reading and printing strings. The parameterised effect `Gen [a]` provides `yield` for yielding a value of type `a`. Similarly, `State [a]` provides `get` and `put` for reading and updating a state cell of type `a`. Ignore the brackets around the type variable `a` for now; we will return to them in Section 3.8. The `ND` effect provides nondeterministic choice and failure. The operation `choose` returns a boolean representing a choice, while `fail` returns an uninhabited type representing failure.

Maetel tracks *effect contexts*, the effects available at each point in a program. Effect contexts belong to the underlying type checking mechanism and are not visible in programs by default. We can only use an effect if it is available in the ambient effect

3 A tour of modal effect types

context. Suppose, for example, the surrounding programs install handlers for `Gen Int`, `State Int`, and `ND`. The ambient effect context is then `Gen Int`, `State Int`, `ND`. Within this context, the following variants of effectful `inc` type check:

```
genInc : Int → 1
genInc x = do yield (x + 1)

stateInc : 1 → 1
stateInc () = do put (inc (do get))

randomInc : Int → Int
randomInc x = if do choose then x + 1 else x
```

The keyword `do` invokes an operation. It allows unit arguments to be omitted, as in `do get` and `do choose`. The function `genInc` uses `Gen Int` to yield the incremented argument. The function `stateInc` uses `State Int` to read the current state, increment it, and store the result. The function `randomInc` uses `ND` to choose nondeterministically whether to increment its argument. Nevertheless, all three functions retain their ordinary types as long as the ambient effect context provides the effects they use. The pure higher-order functions from Section 3.1 can be applied under the current effect context:

```
iter genInc pi      # : 1
map (randomInc, pi) # : List Int
```

Since `iter` and `map` are pure as indicated by their modality `[]`, we can use them in any effect context. Their arguments `genInc` and `randomInc` can still use the effects provided by the ambient effect context. For this reason, *Maetel* rejects the following type signature for `iter`:

```
iter : forall a . []((a → 1) → [] (List a → 1)) # ill-typed
```

This type incorrectly claims that the result function of `iter` is pure, which is not true since it calls the argument.

As another negative example, since we assumed the current effect context to be `Gen Int`, `State Int`, `ND`, a program that prints a list does not type check:

```
printlist : forall a . List a → (a → String) → 1 # ill-typed
printlist xs f = iter (fun x → do print (f x)) xs
```

The pure function `iter` is safe to use here, but its argument uses the missing `IO` effect.

3.3 Specifying effects

What if we want to define an effectful function where the ambient effect context does not provide the effects it needs? For instance, it is useful to define the variants of `inc` from Section 3.2 globally, so that we can use them later within different handlers. Recall the modality `[]` from Section 3.1 for pure programs. We call it the *empty* absolute modality and extend it to be non-empty to specify which effects a program uses. We can therefore write the following global definitions for `inc` variants:

```
genInc : [Gen Int](Int → 1)
genInc x = do yield (x + 1)

stateInc : [State Int](1 → 1)
stateInc () = do put (inc (do get))

randomInc : [ND](Int → Int)
randomInc x = if do choose then x + 1 else x
```

The absolute modality `[Gen Int]` says that `genInc` is defined in the effect context `Gen Int`. Thus, its body may use `yield`, but cannot depend on other effects from the surrounding context. Similarly, `[State Int]` allows `stateInc` to use integer state, and `[ND]` allows `randomInc` to make a nondeterministic choice. In general, an absolute modality `[E]` specifies that a program is defined in the effect context `E`, and therefore may use only effects from `E`. The modality `[E]` makes sure that when the program is run, the effects in `E` must be available in the ambient effect context.

With absolute modalities, we can now fix the ill-typed `printlist` function and define it where the ambient effect context does not provide `IO`:

```
printlist : forall a . [IO](List a → (a → String) → 1)
printlist xs f = iter (fun x → do print (f x)) xs
```

Absolute modalities allow programmers to specify the effects a program uses independently of what the ambient effect context provides. They usually appear in the type signatures of global effectful programs, serving as useful documentation for programmers to indicate which effects are used in the programs. They are also useful locally in programs to have more precise control over uses of effects. For instance, we can require concurrent processes to send only pure functions to each other to avoid crashes due to running a program from another process.

As readers may have noticed, we have been writing modalities in types, but not in the bodies of programs. Maetel infers introduction and elimination of modalities, similarly

to how ordinary functional programming languages infer polymorphic types and their instantiations.

3.4 Handling effects

We now define handlers for the `Gen`, `State`, and `ND` effects from Section 3.2. The handler `asList` handles `Gen` `a` by collecting the yielded values in a list:

```
asList : forall [a] . [](<Gen a>(1 → 1) → List a)
asList f = handle f () with
  return () ⇒ nil
  yield x r ⇒ cons x (r ())
```

The handler `all` handles `ND` by collecting all possible results in a list:

```
all : forall [a] . [](<ND>(1 → a) → List a)
all m = handle m () with
  return x    ⇒ cons x nil
  choose () r ⇒ r true ++ r false
  fail  () _  ⇒ nil
```

The handler `state` handles `State` `s` by threading a state through the computation:

```
state : forall [s] [a] . [](<State s>(1 → a) → s → a)
state m = handle m () with
  return x    ⇒ fun s → x
  get  () r    ⇒ fun s → r s
  put s' r     ⇒ fun s → r () s'
```

We can use these handlers to handle effectful programs from the previous sections:

```
> asList (fun () → iter genInc pi)
# [4,2,5,2,6,10] : List Int

> all (fun () → map (randomInc, pi))
# [[4,2,5,2,6,10],...,[3,1,4,1,5,9]] : List (List Int)

> state (fun () → iter (fun _ → stateInc ()) pi; do get) 0
# 6 : Int
```

The first program collects the incremented elements yielded by `genInc`. The second program returns every list that the nondeterministic `map` can produce, in total $2^6 = 64$

lists. The final program runs `stateInc` once for each element of `pi`, starting from 0, and returns the final state 6.

Now we explain the types of these handlers. Each handler type has a top-level `[]` as we have seen before, which says that the handler itself uses no effects in its definition. Each handler also has a *relative modality* within angle brackets, which specifies the effect that the handler interprets. For example, `<ND>` says that the handler `all` handles the `ND` effect for its argument. Similarly, `<Gen a>` and `<State s>` say that the handlers `asList` and `state` handle the `Gen a` and `State s` effects, respectively.

One may wonder why handlers need a different form of modality. For example, why not give `asList` an absolute modality as follows?

```
asList : forall [a] . []([Gen a](1 → 1) → List a)
```

It type checks, but breaks compositionality. One of the main benefits of effect handlers is that they compose. A handler interprets one effect and automatically forwards other effects to surrounding handlers. Programmers can therefore combine independently defined handlers, with each handler responsible for only one effect. We will give examples of composing handlers in Section 3.5.

The absolute modality above prevents this form of composition. The modality `[Gen a]` fixes the argument of `asList` to only use `Gen a`; no other effects from the ambient effect context where `asList` is called are allowed. Instead, the relative modality `<Gen a>` allows the argument to use `Gen a` provided by the handler `asList`, together with any other effects from the ambient effect context of the call site. The handler handles `Gen a` and forwards the ambient effects.

Relative modalities allow programmers to specify relative changes to the ambient effect context. They usually appear in the type signatures of handlers, serving as useful documentation for programmers to indicate which effects are handled. In general, a handler that handles effects `D` and itself uses effects `E` usually has type `[E](<D>(1 → A) → B)`. Maetel enables programmers to only care about what effects are used and handled, leaving other forwarded effects untouched.

3.5 Composing handlers

We compose handlers by composing functions. For instance, the function `prefixSum` uses state to maintain the current sum and a generator to yield each prefix sum:

```
prefixSum : [State Int, Gen Int](List Int → 1)
prefixSum xs = iter (fun x → do put (do get + x); do yield (do get)) xs
```

3 A tour of modal effect types

Its absolute modality documents the two effects used by its definition. To run `prefixSum`, we need to provide both effects. We use the two handlers `state` and `asList`:

```
> asList (fun () → state (fun () → prefixSum pi) 0)
# [3,4,8,9,14,23] : List Int
```

The global effect context is empty, while the effect context for the program `prefixSum pi` is `State Int`, `Gen Int`, where `Gen Int` is added by the relative modality `<Gen Int>` of the outer handler `asList`, and `State Int` is added by the relative modality `<State Int>` of the inner handler `state`.

As another example, the function `subsetSum` combines state with nondeterminism to calculate the sum of a subset of a list of integers:

```
subsetSum : [State Int, ND](List Int → Int)
subsetSum xs =
  iter (fun x → if do choose then do put (do get + x) else ()) xs; do get
```

Handling state first and nondeterminism second enumerates all $2^6 = 64$ possible sums of subsets of `pi`:

```
> all (fun () → state (fun () → subsetSum pi) 0)
# [23,14,18,...,9,0] : List Int
```

As above, the global effect context is empty, while the effect context for the program `subsetSum pi` is `State Int`, `ND` extended by the relative modalities of the two handlers.

3.6 Masking effects

Sometimes we want a program to retain most of its ambient effects but prevent it from using some particular effects. This situation typically arises when a library function is implemented by a local handler and invokes a callback within the scope of that handler. The callback may use any ambient effects at the call site, but we do not want the callback to use the effect installed by this local handler, since it would leak the implementation details of library functions. For instance, consider a function that uses `yield` internally to find the first value satisfying a predicate:

```
find' : forall [a] . []((a → Bool) → List a → Maybe a) # ill-typed
find' p xs =
  handle (iter (fun x → if p x then do yield x else ()) xs) with
    return () ⇒ nothing
    yield x r ⇒ just x
```


This function iterates over a list and invokes the `yield` operation when it finds a value satisfying the predicate `p`. The local handler handles the `yield` operation, returning the yielded value and terminating the iteration by discarding the continuation `r`. The function call `p x` is in the scope of the local handler. If `p` calls `yield`, the handler could mistake that call for a value yielded by the search itself and return the wrong result. The type for `find'` is invalid because it does not reflect the fact that the `yield` operation performed by the argument `p` is also handled by the local handler. This unintended handling is usually called the *accidental handling* problem [Zhang and Myers 2019] or the *effect pollution* problem [Convent et al. 2020]. In other words, we want *abstraction safety* for effect types, so that effects are only handled if the type tells us so. Maetel achieves this property by forcing the argument type to be annotated with the relative modality `<Gen a>` as:

```
find' : forall [a] . [](<Gen a>(a → Bool) → List a → Maybe a)
```

The relative modality `<Gen a>` explicitly states that the argument may use the `Gen a` effect provided by the local handler.

Even though the type of `find'` now tells us what will happen, the behaviour of `find'` is still not exactly what we want. We want to prevent the handler from handling the `yield` operation invoked by the argument `p`. *Masks* allow us to encapsulate internal effects, solving the effect pollution problem. Masking an effect for a program prevents the dynamically nearest handler from handling this effect performed by this program. For the above example, we can mask the `Gen` effect around the argument `p` as follows.

```
find : forall [a] . []((a → Bool) → List a → Maybe a)
find p xs =
  handle (iter (fun x → if mask<Gen>(p x) then do yield x else ()) xs) with
    return () ⇒ nothing
    yield x r ⇒ just x
```

The expression `mask<Gen>(p x)` masks `Gen` for the call `p x`. Any `yield` operation performed by `p` is therefore forwarded out of the local handler here. This handler still handles the `yield` calls in the `then` branch. Consequently, the type of `find` does not require the argument type to have the relative modality `<Gen a>`, since the `mask<Gen>` removes the `Gen` effect from the effect context for the call `p x`.

With masks, relative modalities are extended from the form `<D>` to `<L|D>`, where `D` denotes the effects added to the ambient effect context and `L` denotes the effects removed from the ambient effect context. We write `<D>` as a shorthand for `<|D>`. We call `<D>` an *extension modality* and `<L|>` a *mask modality*. Mask modalities appear in types when we want to state that some programs cannot use certain effects from the ambient effect

context. For example, the following function yields the current state after invoking a callback:

```
yieldState : [Gen Int, State Int](<Gen|>(1 → 1) → 1)
yieldState f = mask<Gen>(f ()); do yield (do get)
```

The absolute modality says that `yieldState` may use `Gen Int` and `State Int`. The mask modality `<Gen|>` states that the effect context for the callback `f` does not include the effect `Gen Int` from the ambient effect context, coinciding with the `mask<Gen>` in the body. There is no mask modality in the type of `find`, because the mask term in the definition of `find` cancels the effect introduced by the local handler, so that the effect context inside the mask is the same as the ambient effect context outside the handler.

3.7 Algebraic data types

Maetel supports algebraic data types and pattern matching. For example, the list type used throughout this chapter can be defined as follows:

```
data List a = nil | cons a (List a)
```

Data constructors may also store functions, and those functions may use ambient effects. For example, we can define a data type `Proc` for cooperative processes, whose constructor stores a function that takes a queue of other suspended processes:

```
data Proc = proc (List Proc → 1)

push : [] (Proc → List Proc → List Proc)
push p ps = ps ++ cons p nil

next : [] (List Proc → 1)
next nil = ()
next (cons (proc p) ps) = p ps
```

The function `push` appends a process to the queue, while `next` runs the process at its head. The top-level `[]` modalities say that these auxiliary functions are pure, while the stored processes may use ambient effects at the call site. For example, the following process increments the state and passes control to the next process in the queue:

```
incProc : [State Int]Proc
incProc = proc (fun ps → stateInc ()); next ps)
```

Instead of manually scheduling processes with queue operations, we can implement a cooperative scheduler with effect handlers through a Unix-style fork operation [Ritchie and Thompson 1974; Hillerström 2022]:

```
eff UCoop = ufork : 1 ⇒ Bool | suspend : 1 ⇒ 1
```

The operation `ufork` returns `false` in the parent process and `true` in the child. The operation `suspend` asks the scheduler to pause the current process and run another one.

The scheduler handles `UCoop` and maintains the process queue:

```
schedule : [](<UCoop>(1 → 1) → List Proc → 1)
schedule m = handle m () with
  return () ⇒ fun ps → next ps
  suspend () r ⇒ fun ps → next (push (proc (fun ps' → r () ps')) ps)
  ufork () r ⇒ fun ps → r true (push (proc (fun ps' → r false ps')) ps)
```

When a process returns, the handler runs the next process in the queue. When a process suspends, the handler stores its continuation at the end of the queue and runs the next process. When a process forks, the handler runs the child process immediately and queues the parent process. The type of `schedule` follows the same pattern as the earlier handlers. Its top-level `[]` says that the scheduler itself is pure. Its relative modality `<UCoop>` says that it handles the `UCoop` effect of its argument. Each process can freely use ambient effects from its call site.

3.8 Kinds

Now we explain what the brackets in the syntax `forall [a]` mean. These brackets appear in some of the polymorphic types, mostly in the types of handlers. As a running example, recall the type of the state handler:

```
state : forall [s] [a] . [](<State s>(1 → a) → s → a)
```

After this handler finishes, the `State` effect handled by it is no longer available. Any value returned by the handler must therefore not use the `State` effect provided by the handler. This matters when the returned value is a function, because the function body may read and update the state:

```
state (fun () → (fun () → do put (do get + 42))) 0 # ill-typed
```

The returned function `fun () → do put (do get + 42)` captures state operations provided by the handler `state` and carries these operations outside the handler. This function would get stuck when called due to unhandled state operations. On the other hand, returning a value of type `Int` is safe since it cannot perform the `State` effect

provided by the handler. Returning a value of type `[State Int](1 → 1)` is also safe; although it may use the `State` effect, this effect is not provided by the handler state here, but from the effect context specified by the absolute modality `[State Int]`.

Based on this observation, Maetel has a kind system which classifies types according to whether their values may depend on ambient effects. The kind `Abs` represents *absolute types*, whose values are independent of the ambient effect context. Base types such as `Int` are absolute, as are data structures containing only absolute values. An absolute modality also produces an absolute type because it records the complete effect context on which a value depends, such as `[State Int](1 → 1)`. The kind `Any` contains all types. Function types have this kind because a function body may use ambient effects. Every absolute type can also be used where a type of kind `Any` is expected. These two kinds explain the brackets on polymorphic variables. A type variable bound by `forall` has kind `Any` by default; writing `forall [a]` instead restricts the type variable `a` to kind `Abs`.

With this kind system, Maetel rejects the earlier problematic program. For that program to type check, the type variable `a` in the type of state would have to be instantiated with the function type `1 → 1`, which has kind `Any`. However, the type variable `a` has kind `Abs`, so the instantiation is rejected.

Sometimes a programmer wants to deliberately assert that a handler returns a value that depends on the handled effect. A more general type for the state handler can record that dependence in its result type:

```
state' : forall [s] a . [](<State s>(1 → a) → s → <State s>a)
state' m = handle m () with
  return x ⇒ fun s → x
  get () r ⇒ fun s → r s s
  put s' r ⇒ fun s → r () s'
```

Here `a` has kind `Any`, so it may be instantiated with the function type `1 → 1`. We do not face the earlier effect leakage problem because the variable `x` in the return clause has type `<State s>a` as reflected in the result type of `state'`. The earlier computation is therefore accepted by `state'`:

```
state' (fun () → fun () → do put (do get + 42)) 0 # : <State Int>(1 → 1)
```

The result type `<State Int>(1 → 1)` records that the returned function still relies on the `State Int` effect. The earlier type of state is usually sufficient in practice.

The state type `s` is restricted to kind `Abs` for the same reason. The `get` and `put` operations may move states between different effect contexts. We restrict states to be independent of the ambient effect context to avoid leaking effects. Otherwise, if `s` could be instantiated with `1 → 1`, we could use state operations to leak effects:

```
state (fun () → asList (fun () → do put (fun () → do yield 42));
      (do get) ()))
```

The `put` operation stores the function that uses the `Gen Int` effect provided by the handler `asList`. After `asList` finishes, `get` retrieves the same function outside that handler. Invoking this function would then call `yield` outside the scope of the handler `asList`. Therefore, Maetel forces the state type to have kind `Abs`. Maetel achieves this by requiring the argument and result types of all operations to have kind `Abs`, as we have seen in the declaration of the `State` effect:

```
eff State [s] = get : 1 ⇒ s | put : s ⇒ 1
```

Most programs encounter the `Abs` kind only in type signatures that move values across handler boundaries or pass values to operations. Ordinary polymorphic functions such as `map` and `iter` continue to use variables of the default kind `Any`.

3.9 Effect polymorphism

All examples in the preceding sections use no type variables that abstract over effects. Absolute modalities specify a fixed effect context, while relative modalities allow functions to transform the ambient context without naming it. These modalities, together with the notion of ambient effect contexts, enable programmers to write modular effectful programs without relying on any explicit polymorphism mechanism.

Nevertheless, effect variables can still be useful when we want to explicitly abstract over different effect contexts specified by absolute modalities. The most prominent case is higher-order effects, whose operations take functions or other suspended computations as arguments [Wu et al. 2014; Piróg et al. 2018; van den Berg and Schrijvers 2024; Yang and Wu 2023, 2026]. For example, consider a different interface for the cooperative concurrency effect in Section 3.7 with a higher-order `fork` operation that takes the child process as an argument. As discussed in Section 3.8, an operation argument must have an absolute type so that it cannot leak ambient effects. We could therefore define `fork` as follows:

```
eff Coop = fork : [Coop](1 → 1) ⇒ 1 | suspend : 1 ⇒ 1
```

The absolute modality here restricts the child process to using only `Coop`. We have to define different versions of `Coop` to allow the child process to use other effects.

To admit other effects without fixing them in the declaration, we abstract over effect contexts with an effect variable `e`:

```
eff Coop e = fork : [Coop e, e](1 → 1) ⇒ 1 | suspend : 1 ⇒ 1
```

The variable e ranges over the additional effects that a child may use.

As far as we know, the requirements for effect polymorphism in Maetel mainly arise when we want a modular interface for higher-order effects. In our recent extended abstract [Boussaa et al. 2026], we sketched a new approach to extend modal effect types with modular types for higher-order effects without relying on effect polymorphism. Another use case for effect polymorphism is to faithfully encode other effect systems as we will demonstrate in Part II.

3.10 Type inference

The preceding examples contain no explicit terms for type abstractions, instantiations, modality introductions, or modality eliminations. Maetel implicitly infers them using a bidirectional typing approach; we defer a detailed description to Part III. One thing worth mentioning is that Maetel can implicitly instantiate type variables with a type that contains quantifiers or modalities, which is what we mean by *first-class polymorphism* and *first-class modal types*. Such instantiations are useful since a handler is often polymorphic over its return type, which is instantiated to a concrete type when applying a handler to a program. For instance, suppose we have a list of pure polymorphic functions:

```
pureFuncs : List (forall a . [] (a → a → a))
```

We can apply the state handler to a program that returns this list:

```
state (fun () → do put 42; pureFuncs)
```

In order for this program to type check, we need to instantiate the type variable a of `state` with the type `List (forall a . [] (a → a → a))`. Maetel infers this instantiation automatically by locally making use of the type information of `pureFuncs`, while also making use of the type information of `state` to introduce the relative modality `<State Int>` for the whole argument.

3.11 Maetel cheat sheet

We conclude this chapter with a Maetel cheat sheet. It is not a complete reference, but shows representative programming patterns using examples from this chapter.

```
## Effects are declared as a collection of operations
eff IO      = read : 1 ⇒ String | print : String ⇒ 1
eff Gen [a] = yield : a ⇒ 1
eff State [s] = get : 1 ⇒ s | put : s ⇒ 1
eff ND      = choose : 1 ⇒ Bool | fail : 1 ⇒ forall a . a
```

```

## Absolute modalities track which effects are used
inc  : [](Int → Int)  # A pure function
iter : forall a . []((a → 1) → List a → 1) # A pure higher-order function
genInc : [Gen Int](Int → 1) # An effectful function that uses only Gen Int
genInc x = do yield (x + 1)

stateInc  : [State Int](1 → 1) # Uses only State Int
prefixSum : [State Int, Gen Int](List Int → 1) # Uses both effects

## Relative modalities track which effects are handled
asList : forall [a] . [](<Gen a>(1 → 1) → List a) # A handler for Gen
asList f = handle f () with
  return () ⇒ nil
  yield x r ⇒ cons x (r ())

state : forall [s] [a] . [](<State s>(1 → a) → s → a) # Handles State
all   : forall [a] . [](<ND>(1 → a) → List a) # Handles ND

```


Part I

MET the Calculus

We introduce a core calculus **MET** for modal effect types, together with different extensions ranging from standard type system features to effect specific features.

Chapter [4](#) introduces the core calculus **MET**. Chapter [5](#) presents extensions of **MET**.

4 A core calculus

Find the simplest algebra of types, and some syntax for them.

—Stephen Dolan (2016)

We introduce **MET**, a simply-typed calculus with effect handlers and modal effect types. We start with an overview of the core calculus, followed by its syntax and mode theory, which defines the algebraic structures of modes (effect contexts) and modalities. We then present its static and dynamic semantics as well as its metatheory, including type soundness and effect safety. We aim at presenting a minimal core calculus in a self-contained way in this chapter, assuming no prior knowledge of multimodal type theory (MTT) [Gratzer 2023; Gratzer et al. 2020]. We conclude this chapter with a discussion of the connections between **MET** and MTT. We defer extensions including data types, parametric polymorphism, masks, and enriched forms of handlers to Chapter 5.

4.1 Overview of **MET**

We give a high-level overview of the core calculus **MET**. This overview complements the Maetel tutorial in Chapter 3 to show how effect contexts are tracked and how modalities are explicitly introduced and eliminated in **MET**.

Every well-typed term in the simply-typed λ -calculus is also well-typed in **MET**. For example, we may define a higher-order application function:

$$\text{app}_{\text{MET}} \doteq \lambda f^{\text{Int} \rightarrow 1}. \lambda x^{\text{Int}}. f \ x : (\text{Int} \rightarrow 1) \rightarrow \text{Int} \rightarrow 1$$

We use meta-level macros defined by $\doteq$ in red to refer to code snippets.

Effect contexts. **MET** adopts a contextual reading of effects, inspired by Frank [Lindley et al. 2017] and Effekt [Brachthäuser et al. 2020]. For brevity, we directly track operation labels, instead of organising operations into effect names (an effect consists of several operations) as in Section 3.2. For our examples we assume two operations $\text{yield} : \text{Int} \rightarrow 1$ and $\text{ask} : 1 \multimap \text{Int}$. In addition to a typing context Γ , typing judgements include an

ambient effect context which tracks the operations that may be performed. Consider the typing judgement:

$$\Gamma_1 \vdash \text{gen}_{\text{MET}} \doteq \lambda x^{\text{Int}}. \text{do yield } x : \text{Int} \rightarrow 1 @ \text{yield}$$

The function has type $\text{Int} \rightarrow 1$. When applied, it performs the `yield` operation using the `do` syntax. The judgement specifies the effect context with the syntax `@yield`, which tracks the possibility of performing `yield`. We can now apply app_{MET} to gen_{MET} and 42:

$$\Gamma_1 \vdash (\lambda f^{\text{Int} \rightarrow 1}. \lambda x^{\text{Int}}. f \ x) (\lambda x^{\text{Int}}. \text{do yield } x) 42 : 1 @ \text{yield}$$

There is a natural notion of subeffecting on effect contexts. We also have:

$$\Gamma_2 \vdash \lambda x^{\text{Int}}. \text{do yield } x : \text{Int} \rightarrow 1 @ \text{yield, ask}$$

Absolute modalities. Effect contexts specified by `@E` belong to typing judgements instead of types. MET uses modalities to track effects in types. An *absolute modality* $[E]$ allows us to specify a new effect context E in types different from the ambient one. For example, consider the typing derivation:

$$\frac{\Gamma_3, \blacksquare_{[\text{yield}]} \vdash \lambda x^{\text{Int}}. \text{do yield } x : \text{Int} \rightarrow 1 @ \text{yield}}{\Gamma_3 \vdash \text{gen}'_{\text{MET}} \doteq \text{mod}_{[\text{yield}]} (\lambda x^{\text{Int}}. \text{do yield } x) : [\text{yield}](\text{Int} \rightarrow 1) @ \text{ask}}$$

This term has type $[\text{yield}](\text{Int} \rightarrow 1)$. We highlight modalities in blue when they appear in types. The syntax $\text{mod}_{[\text{yield}]}$ introduces an absolute modality $[\text{yield}]$ which specifies a singleton effect context of `yield` and uses it to override the ambient effect context `ask`. The typing judgement of the premise uses the new effect context `yield` as its ambient effect context. The lock $\blacksquare_{[\text{yield}]}$ tracks the switch of the effect context and controls the accessibility of variables on the left of it. Only variables that are known not to use effects other than `yield` may be used. This is important to ensure effect safety. For example, consider the following invalid judgement:

$$\Gamma_3, f : \text{Int} \rightarrow 1 \not\vdash \text{mod}_{[\text{yield}]} (\lambda x^{\text{Int}}. f \ x) : [\text{yield}](\text{Int} \rightarrow 1) @ \text{ask}$$

This program is unsafe as f may invoke `ask` which we must not use under the effect context `yield` specified by the modality $[\text{yield}]$. MET rejects this judgement as it relies on the following invalid judgement for the inner function:

$$\Gamma_3, f : \text{Int} \rightarrow 1, \blacksquare_{[\text{yield}]} \not\vdash \lambda x^{\text{Int}}. f \ x : \text{Int} \rightarrow 1 @ \text{yield}$$

This typing judgement is invalid as the lock $\blacksquare_{[\text{yield}]}$ prevents the use of f . To make it valid, we can annotate the binding of f with the modality $[\text{yield}]$ as $f : [\text{yield}] \text{Int} \rightarrow 1$. This annotation tracks that the function f may only use the operation `yield`. Such annotated

bindings are introduced by modality elimination. For instance, we can eliminate the modality of gen'_{MET} and then apply it via the `let mod` syntax (where we elide the typing of the bound term):

$$\frac{\dots \quad \Gamma_1, f : [\text{yield}] \text{Int} \rightarrow 1 \vdash f \text{ 42} : 1 @ \text{yield}}{\Gamma_1 \vdash \text{let mod}_{[\text{yield}]} f = \text{mod}_{[\text{yield}]} (\lambda x^{\text{Int}}. \text{do yield } x) \text{ in } f \text{ 42} : 1 @ \text{yield}}$$

The term $\lambda x^{\text{Int}}. \text{do yield } x$ inside the modality $[\text{yield}]$ is bound to f . The binding of f is annotated with this absolute modality. Consequently, the use of f in $f \text{ 42}$ requires the ambient effect context to contain the operation `yield`. In general, whether a variable binding $f :_{\mu} A$ can be used after a lock $\mathbf{\mu}_v$ is controlled by a modality transformation relation which we will introduce in Section 4.3.

Relative modalities. As well as being able to specify a fresh effect context from scratch with an absolute modality, MET also has *relative modalities* $\langle D \rangle$ which allow us to extend the ambient effect context with an *extension* D . For instance, consider the derivation:

$$\frac{\mathbf{\mu}_{\langle \text{yield} \rangle} \vdash \lambda x^{\text{Int}}. \text{do yield } (\text{do ask } ()) : \text{Int} \rightarrow 1 @ \text{yield, ask}}{\vdash \text{mod}_{\langle \text{yield} \rangle} (\lambda x^{\text{Int}}. \text{do yield } (\text{do ask } ())) : \langle \text{yield} \rangle (\text{Int} \rightarrow 1) @ \text{ask}}$$

The relative modality $\langle \text{yield} \rangle$ extends the ambient effect context `ask` with the operation `yield`. Consequently, the function inside can use both operations. Relative modalities are especially useful for giving composable types to effect handlers as we have seen in Section 3.4.

4.2 Syntax

Figure 4.1 gives the syntax of the core part of MET . MET extends a simply-typed λ -calculus with standard constructs for effects and handlers (highlighted in orange) as well as modal effect types (highlighted in grey).

Types	$A, B ::= 1 \mid A \rightarrow B \mid \mu A$	Terms	$M, N ::= x \mid \lambda x^A. M \mid M N \mid \text{mod}_{\mu} V$
Effects	$D, E, F ::= \cdot \mid \ell, E$		$\mid \text{let}_v \text{ mod}_{\mu} x = V \text{ in } M$
Modalities	$\mu, \nu ::= [E] \mid \langle D \rangle$		$\mid \text{do } \ell \text{ } M \mid \text{handle } M \text{ with } H$
Kinds	$K ::= \text{Abs} \mid \text{Any}$	Values	$V, W ::= x \mid \lambda x^A. M \mid \text{mod}_{\mu} V$
Contexts	$\Gamma ::= \cdot \mid \Gamma, x :_{\mu} A \mid \Gamma, \mathbf{\mu}_{\mu}$	Handlers	$H ::= \{\text{return } x \mapsto M\}$
Label contexts	$\Sigma ::= \cdot \mid \Sigma, \ell : A \twoheadrightarrow B$		$\mid \{\ell \text{ } p \text{ } r \mapsto M\} \uplus H$

Figure 4.1: Syntax of MET .

We adopt the Barendregt convention that all variables are distinct [Barendregt 1985]. We implicitly identify each syntactic category up to its equivalence relation, which is

the standard α -equivalence by default unless defined explicitly in later sections. For brevity, we fix a global context Σ which associates each global operation label with its signature. An entry $\ell : A \rightarrow B$ indicates that the operation ℓ takes an argument of type A and returns a value of type B .

In the following sections, we first present the mode theory of **Met**, which defines the structures of modes and modalities, before moving to the kinding, typing, operational semantics, and metatheory of **Met**.

4.3 Mode theory

Figure 4.2 summarises the mode theory of **Met**. We explain it in the following sections.

Well-formedness $\boxed{\Gamma \vdash E} \quad \boxed{\Gamma \vdash \mu}$	
$\frac{}{\Gamma \vdash \cdot}$	$\frac{\ell \in \text{dom}(\Sigma) \quad \Gamma \vdash E}{\Gamma \vdash \ell, E} \quad \frac{\Gamma \vdash E}{\Gamma \vdash [E]} \quad \frac{\Gamma \vdash D}{\Gamma \vdash \langle D \rangle}$
Equivalence and subeffecting $\boxed{\Gamma \vdash E \equiv F} \quad \boxed{\Gamma \vdash E \leq F} \quad \boxed{\Gamma \vdash \mu \equiv \nu}$	
$\frac{}{\Gamma \vdash E \equiv E}$	$\frac{\Gamma \vdash E_1 \equiv E_2 \quad \Gamma \vdash E_2 \equiv E_3}{\Gamma \vdash E_1 \equiv E_3} \quad \frac{\Gamma \vdash E \equiv E'}{\Gamma \vdash \ell, E \equiv \ell, E'} \quad \frac{\ell \neq \ell'}{\Gamma \vdash \ell, \ell', E \equiv \ell', \ell, E}$
$\frac{\Gamma \vdash E, E' \equiv F}{\Gamma \vdash E \leq F}$	$\frac{\Gamma \vdash E \equiv F}{\Gamma \vdash [E] \equiv [F]} \quad \frac{\Gamma \vdash D \equiv D'}{\Gamma \vdash \langle D \rangle \equiv \langle D' \rangle}$
Modality action $\boxed{\mu(E)}$	
$[E](F) = E$	$\langle D \rangle(E) = D, E$
Modality composition $\boxed{\mu \circ \nu}$	
$\mu \circ [E] = [E]$	$[E] \circ \langle D \rangle = [D, E] \quad \langle D_1 \rangle \circ \langle D_2 \rangle = \langle D_2, D_1 \rangle$
Modality transformation $\boxed{\Gamma \vdash \mu \Rightarrow \nu @ F}$	
$\frac{\text{MT-ABS} \quad \Gamma \vdash E \leq \mu(F)}{\Gamma \vdash [E] \Rightarrow \mu @ F}$	$\frac{\text{MT-REL}}{\Gamma \vdash \langle D \rangle \Rightarrow \langle D \rangle @ F}$

Figure 4.2: Mode theory of **Met**.

As usual, equivalence, subeffecting, and modality transformation judgements implicitly require well-formedness of effects and modalities as well-formedness conditions. We sometimes omit Γ when it is clear from the context to simplify the presentation.

4.3.1 Effect contexts as modes

The syntax of effects is defined in Figure 4.1. We distinguish *effect contexts*, which specify the available effects and over which E and F range, from *extensions*, which extend effect contexts and over which D ranges. For the minimal core calculus in this chapter, extensions and effect contexts share the same syntax and rules. The extensions to MET in Chapter 5 distinguish them explicitly.

The *modes* of MET are effect contexts E . Each type and term is at some effect context E , specifying the available effects from the context. Figure 4.2 gives the well-formedness, equivalence, and subeffecting rules for effect contexts. Well-formedness $\Gamma \vdash E$ checks that each label in E is defined in the global label context Σ . The context Γ is only needed to maintain well-scopedness for the extension with polymorphism in Section 5.4.

Effect contexts E are defined as scoped rows of operation labels [Leijen 2005]. Each label denotes an effectful operation present in the global label context Σ . An effect context may contain the same label multiple times. In Chapter 6, we will see different ways to organise effect contexts in addition to scoped rows.

Following Rémy [1994] and Leijen [2005], we identify effects up to reordering of distinct labels. Since we only have labels here, it is technically impossible to distinguish between two identical labels. It would become possible to distinguish between them in Sections 5.2 and 5.9 with the extensions of parameterised effects, which parameterise operation signatures with type parameters, and presence types, which allow us to assign different operation signatures to the same labels in effect contexts.

We define a subeffecting relation on effect contexts $E \leq F$ if we can append another effect context to E to obtain F .

4.3.2 Modalities manipulating effect contexts

Components of types and terms may have different effect contexts from the ambient one. For instance, under a pure global context, we may want to define a function that performs some effects when it is invoked. We use *modalities* to manipulate effect contexts. For the modal type μA , the effect context for A is derived from the ambient effect context manipulated by the modality μ as follows.

$$[E](F) = E \qquad \langle D \rangle(E) = D, E$$

The *absolute modality* $[E]$ completely replaces the effect context F with E , similar to effect annotations on function types in traditional effect systems. The *relative modality* $\langle D \rangle$ is the key novelty of MET. It specifies a transformation on the input effect context, extending it with D . We write D before E because the effects in D are more recent than

those in E . We call $\langle \rangle$ the identity modality since $\langle \rangle(E) = E$ for any E . Modalities are monotone total functions on effect contexts. If $E \leq F$, we have $\mu(E) \leq \mu(F)$.

Modality composition. We compose the actions of modalities as follows.

$$\mu \circ [E] = [E] \quad [E] \circ \langle D \rangle = [D, E] \quad \langle D_1 \rangle \circ \langle D_2 \rangle = \langle D_2, D_1 \rangle$$

Our composition reads from left to right. First, an absolute modality completely specifies the new effect context, thus shadowing any other modality μ . Second, replacing the effect context with E and then extending it with D is equivalent to just replacing it with D, E . Third, successive extensions can be combined into one.

Composition is well defined because applying a composite modality is equivalent to applying its components sequentially: $(\mu \circ \nu)(E) = \nu(\mu(E))$. We also have associativity $(\mu \circ \nu) \circ \xi = \mu \circ (\nu \circ \xi)$ and identity $\langle \rangle \circ \mu = \mu \circ \langle \rangle = \mu$.

We choose a left-to-right composition because the modality μ in a modal type μA represents a manipulation of the ambient effect context. As a result, for the type $\mu \nu A$, the modality ν indicates a more recent manipulation of the effect context than μ . For instance, for the type $[E][F](1 \rightarrow 1)$, the available effects for the function are F instead of E , which is consistent with composing the two modalities: $([E] \circ [F])(1 \rightarrow 1) = [F](1 \rightarrow 1)$.

Modality transformations. Just as modalities allow us to manipulate effect contexts, we need a transformation relation that tells us when we can change modalities.

We write $\Gamma \vdash \mu \Rightarrow \nu @ F$ for a transformation from the modality μ to ν under the ambient effect context F . Intuitively, such a transformation indicates that under ambient effect context F , the action of μ can be replaced by the action of ν . This relation is used to control variable access as we have demonstrated in Section 2.2. For instance, supposing we have a variable of type $\mu(1 \rightarrow 1)$ under ambient effect context F , we can rewrap it as a function of type $\nu(1 \rightarrow 1)$ if $\mu \Rightarrow \nu @ F$.

Intuitively, $\mu \Rightarrow \nu @ F$ is safe when $\nu(F)$ is larger than $\mu(F)$ so that we have not lost any operations. Moreover, subeffecting should not break the safety guarantee of transformations. That is, $\mu(F') \leq \nu(F')$ should hold for any effect context F' with $F \leq F'$. For instance, consider the following program which coerces the modality $\langle D_1 \rangle$ of a value V to $\langle D_2 \rangle$.

$$\Gamma \vdash \text{let mod}_{\langle D_1 \rangle} x = V \text{ in mod}_{\langle D_2 \rangle} x : \langle D_2 \rangle(\text{Int} \rightarrow \text{Int}) @ E$$

We need the transformation relation $\langle D_1 \rangle \Rightarrow \langle D_2 \rangle @ E$ in its derivation tree:

$$\begin{array}{c}
 \frac{\Gamma @ E \quad \Gamma \vdash \langle D_1 \rangle \Rightarrow \langle D_2 \rangle @ E}{\Gamma \vdash (\langle D_1 \rangle, \text{Int} \rightarrow \text{Int}) \Rightarrow \langle D_2 \rangle} \text{T-VAR} \\
 \frac{\Gamma, x : \langle D_1 \rangle \text{Int} \rightarrow \text{Int}, \blacksquare_{\langle D_2 \rangle} \vdash x : \text{Int} \rightarrow \text{Int} @ D_2, E}{\Gamma, x : \langle D_1 \rangle \text{Int} \rightarrow \text{Int} \vdash \text{mod}_{\langle D_2 \rangle} x : \langle D_2 \rangle (\text{Int} \rightarrow \text{Int}) @ E} \text{T-MOD} \\
 \frac{\Gamma \vdash V : \langle D_1 \rangle (\text{Int} \rightarrow \text{Int}) @ E \quad \Gamma, x : \langle D_1 \rangle \text{Int} \rightarrow \text{Int} \vdash \text{mod}_{\langle D_2 \rangle} x : \langle D_2 \rangle (\text{Int} \rightarrow \text{Int}) @ E}{\Gamma \vdash \text{let mod}_{\langle D_1 \rangle} x = V \text{ in mod}_{\langle D_2 \rangle} x : \langle D_2 \rangle (\text{Int} \rightarrow \text{Int}) @ E} \text{T-LETMOD}
 \end{array}$$

To preserve the typing derivation after upcasting E to a larger F , the transformation requires $D_1, F \leq D_2, F$ for any $E \leq F$.

We formally define $\Gamma \vdash \mu \Rightarrow \nu @ F$ as the following two rules.

$$\begin{array}{c}
 \text{MT-ABS} \\
 \frac{\Gamma \vdash E \leq \mu(F)}{\Gamma \vdash [E] \Rightarrow \mu @ F} \\
 \text{MT-REL} \\
 \frac{}{\Gamma \vdash \langle D \rangle \Rightarrow \langle D \rangle @ F}
 \end{array}$$

Rule MT-ABS allows us to transform an absolute modality to any other modality as long as no effect leaks. Rule MT-REL only allows us to transform a relative modality to another equivalent relative modality. The following lemma shows the soundness of modality transformation.

Lemma 1 (Soundness of modality transformation). *If $\Gamma \vdash \mu \Rightarrow \nu @ F$, then $\Gamma \vdash \mu(F') \leq \nu(F')$ for all F' with $\Gamma \vdash F \leq F'$.*

Let us give some examples. First, $[] \Rightarrow \mu @ E$ always holds, consistent with the intuition that pure values can be used anywhere safely. Second, $\langle \ell \rangle \Rightarrow \langle \ell \rangle @ E$ always holds. Third, $\langle \rangle \Rightarrow \langle \ell \rangle @ E$ does not hold for any E .

An interesting observation is that $\langle \rangle \Rightarrow \langle \ell \rangle @ \cdot$ is not allowed, even though it is sound in the core calculus as we have $F \leq \ell, F$ for any effect context F . We do not allow it here for two reasons. First, with the extension of parameterised effects or presence types in Sections 5.2 and 5.9, it would not be sound when F contains another ℓ with a different parameter or signature. Second, allowing it would break abstraction safety as we discussed in Section 3.6.

4.4 Kinds, contexts, and equivalence

Figure 4.3 gives the kinding and type equivalence rules for MET. We have two kinds Abs and Any. The Abs kind is a sub-kind of the kind of all types Any, and denotes types of values that are guaranteed not to use operations from the ambient effect context. Function arrows have kind Any due to the possibility of using operations from the

ambient effect context. A modal type $[E]A$ is absolute as it cannot depend on the ambient effect context.

$$\boxed{\Gamma \vdash A : K}$$

$$\frac{}{\Gamma \vdash 1 : \text{Abs}} \quad \frac{\Gamma \vdash A : \text{Abs}}{\Gamma \vdash A : \text{Any}} \quad \frac{\Gamma \vdash [E] \quad \Gamma \vdash A : \text{Any}}{\Gamma \vdash [E]A : \text{Abs}} \quad \frac{\Gamma \vdash \langle D \rangle \quad \Gamma \vdash A : K}{\Gamma \vdash \langle D \rangle A : K}$$

$$\frac{\Gamma \vdash A : \text{Any} \quad \Gamma \vdash B : \text{Any}}{\Gamma \vdash A \rightarrow B : \text{Any}}$$

$$\boxed{\Gamma \vdash A \equiv B}$$

$$\frac{}{\Gamma \vdash 1 \equiv 1} \quad \frac{\Gamma \vdash \mu \equiv \nu \quad \Gamma \vdash A \equiv B}{\Gamma \vdash \mu A \equiv \nu B} \quad \frac{\Gamma \vdash A_1 \equiv B_1 \quad \Gamma \vdash A_2 \equiv B_2}{\Gamma \vdash A_1 \rightarrow A_2 \equiv B_1 \rightarrow B_2}$$

Figure 4.3: Kinding and equivalence rules of types for MET.

$$\boxed{\Gamma @ E}$$

$$\frac{}{\cdot @ \cdot} \quad \frac{\Gamma @ E \quad \Gamma \vdash \mu \quad \Gamma \vdash A : K}{\Gamma, x :_{\mu} A @ E} \quad \frac{\Gamma @ F \quad \Gamma \vdash \mu \quad \mu(F) = E}{\Gamma, \mathbf{\mu}_{\mu} @ E}$$

$$\boxed{\Gamma \equiv \Gamma' @ E}$$

$$\frac{\Gamma @ E}{\Gamma \equiv \Gamma @ E} \quad \frac{\Gamma \equiv \Gamma_1 @ E \quad \Gamma_1 \equiv \Gamma_2 @ E}{\Gamma \equiv \Gamma_2 @ E} \quad \frac{\Gamma \equiv \Gamma' @ E \quad \Gamma \vdash \mu \equiv \mu' \quad \Gamma \vdash A \equiv A'}{\Gamma, x :_{\mu} A \equiv \Gamma', x :_{\mu'} A' @ E}$$

$$\frac{\Gamma \equiv \Gamma' @ F \quad \mu(F) = E \quad \Gamma \vdash \mu \equiv \mu'}{\Gamma, \mathbf{\mu}_{\mu} \equiv \Gamma', \mathbf{\mu}_{\mu'} @ E} \quad \frac{}{\Gamma, \mathbf{\mu}_{\langle \rangle} \equiv \Gamma @ E} \quad \frac{}{\Gamma, \mathbf{\mu}_{\mu}, \mathbf{\mu}_{\nu} \equiv \Gamma, \mathbf{\mu}_{\mu \circ \nu} @ E}$$

$$\boxed{\Gamma \vdash A \twoheadrightarrow B} \quad \boxed{\vdash \Sigma}$$

$$\frac{\Gamma \vdash A : \text{Abs} \quad \Gamma \vdash B : \text{Abs}}{\Gamma \vdash A \twoheadrightarrow B} \quad \frac{}{\vdash \cdot} \quad \frac{\vdash A \twoheadrightarrow B \quad \vdash \Sigma}{\vdash \ell : A \twoheadrightarrow B, \Sigma}$$

Figure 4.4: Well-formedness and equivalence rules of contexts for MET.

Contexts are ordered. Each term variable binding $x :_{\mu} A$ in contexts is tagged with a modality μ . We omit this annotation when μ is the identity modality. Contexts contain locks $\mathbf{\mu}_{\mu}$ carrying modalities μ . As shown in Section 2.2, they track the introduction and elimination of modalities and play an important role in controlling variable access.

Figure 4.4 gives the well-formedness and equivalence rules for contexts. We define the relation $\Gamma @ E$ to state that the context Γ is well-formed at effect context E . For instance, given some modalities μ, ν , and ξ , the following judgement is valid.

$$\underbrace{\mathfrak{L}_{[F]}, x :_{\mu} A_1, y :_{\nu} A_2}_{@F}, \underbrace{\mathfrak{L}_{[E]}, z :_{\xi} A_3}_{@E} @ E$$

We annotate below the effect context for each variable binding and annotate above the effect context for each type. Reading from left to right, the lock $\mathfrak{L}_{[E]}$ switches the effect context from F to E since $[E](F) = E$.

The context equivalence relation $\Gamma \equiv \Gamma' @ E$ states that the contexts Γ and Γ' are equivalent at effect context E . The interesting rules are the last two rules, which allow us to remove locks with identity modalities and to compose adjacent locks.

Following MTT, we define the meta operation $\text{locks}(-)$ to compose all the modalities on the locks in a context:

$$\text{locks}(\cdot) = \langle \rangle \quad \text{locks}(\Gamma, \mathfrak{L}_{\mu}) = \text{locks}(\Gamma) \circ \mu \quad \text{locks}(\Gamma, x :_{\mu} A) = \text{locks}(\Gamma)$$

Figure 4.4 also gives the well-formedness rules for the label context Σ . For an operation signature $A \rightarrow B$, we restrict the kind of A and B to be Abs in order to prevent effect leakage as discussed in Section 3.8.

4.5 Typing

Figure 4.5 gives the typing rules of MET. The typing judgement $\Gamma \vdash M : A @ E$ states that the term M has type A under context Γ and effect context E . As usual, we require $\Gamma @ E, \Gamma \vdash E, \Gamma \vdash A : K$ for some K , and well-formedness for type annotations as well-formedness conditions. We highlight interesting rules in grey.

Modality introduction and elimination. Modalities are introduced by T-MOD and eliminated by T-LETMOD. The term $\text{mod}_{\mu} V$ introduces modality μ to the type of the conclusion and a lock $\mathfrak{L}_{\mu}$ into the context of the premise, and requires the value V to be well-typed under the new effect context $\mu(E)$ obtained by applying μ to E . The lock $\mathfrak{L}_{\mu}$ tracks the change to the effect context. Specialising the modality μ to either absolute or relative modalities, we get the following two rules.

$$\frac{\Gamma, \mathfrak{L}_{[E]} \vdash V : A @ E}{\Gamma \vdash \text{mod}_{[E]} V : [E]A @ F} \quad \frac{\Gamma, \mathfrak{L}_{\langle D \rangle} \vdash V : A @ D, F}{\Gamma \vdash \text{mod}_{\langle D \rangle} V : \langle D \rangle A @ F}$$

$\boxed{\Gamma \vdash (\mu, A) \Rightarrow v}$		
$\frac{\Gamma \vdash A : \text{Abs}}{\Gamma \vdash (\mu, A) \Rightarrow v}$		
$\frac{\Gamma @ E \quad \Gamma \vdash \mu \Rightarrow v @ E}{\Gamma \vdash (\mu, A) \Rightarrow v}$		
$\boxed{\Gamma \vdash M : A @ E}$		
$\frac{\text{T-VAR} \quad \Gamma \vdash (\mu, A) \Rightarrow \text{locks}(\Gamma')}{\Gamma, x :_{\mu} A, \Gamma' \vdash x : A @ E}$	$\frac{\text{T-MOD} \quad \Gamma, \blacksquare_{\mu} \vdash V : A @ \mu(E)}{\Gamma \vdash \text{mod}_{\mu} V : \mu A @ E}$	$\frac{\text{T-LETMOD} \quad \begin{array}{l} \Gamma, \blacksquare_v \vdash V : \mu A @ v(E) \\ \Gamma, x :_{v \circ \mu} A \vdash M : B @ E \end{array}}{\Gamma \vdash \text{let}_v \text{mod}_{\mu} x = V \text{ in } M : B @ E}$
$\frac{\text{T-ABS} \quad \Gamma, x : A \vdash M : B @ E}{\Gamma \vdash \lambda x^A. M : A \rightarrow B @ E}$	$\frac{\text{T-APP} \quad \begin{array}{l} \Gamma \vdash M : A \rightarrow B @ E \\ \Gamma \vdash N : A @ E \end{array}}{\Gamma \vdash M N : B @ E}$	$\frac{\text{T-DO} \quad \begin{array}{l} \Sigma \ni \ell : A \rightarrow B \\ \Gamma \vdash N : A @ \ell, E \end{array}}{\Gamma \vdash \text{do } \ell N : B @ \ell, E}$
$\frac{\text{T-HANDLE} \quad \begin{array}{l} D = \bar{\ell}_i \quad \Gamma, \blacksquare_{\langle D \rangle} \vdash M : A @ D, E \quad \Gamma, x :_{\langle D \rangle} A \vdash N : B @ E \\ [\Sigma \ni \ell_i : A_i \rightarrow B_i \quad \Gamma, p_i : A_i, r_i : B_i \rightarrow B \vdash N_i : B @ E]_i \end{array}}{\Gamma \vdash \text{handle } M \text{ with } \{\text{return } x \mapsto N\} \uplus \{\ell_i p_i r_i \mapsto N_i\}_i : B @ E}$		

Figure 4.5: Typing rules for MET.

We restrict mod to values to avoid effect leakage [Lorenzen et al. 2024; Ahman 2023]. Otherwise, a term such as $\text{mod}_{\langle \ell \rangle} (\text{do } \ell V)$ would type check under the empty effect context but get stuck due to the unhandled operation ℓ .

The term $\text{let}_v \text{mod}_{\mu} x = V \text{ in } M$ moves the modality μ from the type of V to the binding of x . As with modality introduction, modality elimination is also restricted to values. Following MTT, we use let-style modality elimination which takes another modality v in addition to the modality μ that is eliminated from V . This is crucial for sequential unboxing. For instance, the following term sequentially unboxes $x : v\mu A$. The variables y and z are bound as $y :_v \mu A$ and $z :_{v \circ \mu} A$, respectively.

$$\text{let mod}_v y = x \text{ in let}_v \text{mod}_{\mu} z = y \text{ in } M$$

Without the modality v in the second let, we would get stuck on further eliminating the modality μ from y .

Abstraction and application. Rules T-ABS and T-APP are standard. The effect context E remains the same in the premises and conclusions of both rules, because abstraction and application do not change the ambient effect context.

Invoking and handling operations. Rule T-Do is standard. It invokes an operation ℓ in the ambient effect context ℓ, E . Rule T-HANDLE is more interesting; it introduces a relative modality. The handler $\text{handle } M \text{ with } H$ extends the ambient effect context E with D , the collection of operation labels handled by H , for the term M . We put a lock $\mathbf{lock}_{\langle D \rangle}$ into the context of M to track this change to the effect context. For the return value of M which is bound as x in the return clause, we need to wrap it with $\langle D \rangle$ to reconcile the mismatch between D, E and E , because the return value may capture effects in D . We directly annotate this modality $\langle D \rangle$ on the binding of x in the context. The typing premises for the operation clauses are standard. For an operation with signature $A_i \rightarrow B_i$, the parameter p_i has type A_i and the continuation r_i has type $B_i \rightarrow B$.

Accessing variables. Locks control the accessibility of variables. Rule T-VAR uses the auxiliary judgement $\Gamma \vdash (\mu, A) \Rightarrow \text{locks}(\Gamma')$ (also defined in Figure 4.5) to check whether we can access a variable $x :_{\mu} A$ given all locks in Γ' . When A has kind **Abs**, we can always use x as it does not depend on the effect context. Otherwise we need to make sure the coercion from μ to $\text{locks}(\Gamma')$ is safe by checking the modality transformation relation $\mu \Rightarrow \text{locks}(\Gamma') @ E$, where E is the ambient effect context before manipulation given by $\Gamma @ E$ and $\text{locks}(\Gamma')$ composes the modalities on locks in Γ' . We have seen an example in Section 4.1 showing that the variable $f : \text{Int} \rightarrow 1$ cannot be used after the lock $[\text{yield}]$ while $f :_{[\text{yield}]} \text{Int} \rightarrow 1$ can. As another example, $x :_{\langle \ell \rangle} 1 \rightarrow 1, \mathbf{lock}_{[\ell']} \vdash x : 1 \rightarrow 1 @ \ell'$ is ill-typed since we cannot transform the modality $\langle \ell \rangle$ to $[\ell']$. It would be well-typed if x had type Int which has kind **Abs**, or if the lock had modality $\langle \ell \rangle$ which can be transformed from $\langle \ell \rangle$, or if x were annotated with a modality $[\]$ which can be transformed to $[\ell']$.

Subeffecting. We do not have an explicit typing rule for subeffecting. However, the type system does allow us to use a variable under a larger effect context. We have seen how this works in Section 4.1. We give another example here which upcasts the empty effect context of a variable x to E . It is well-typed because $[\] \Rightarrow [E] @ \cdot$ holds.

$$\lambda x^{[\](\text{Int} \rightarrow \text{Int})}. \text{let mod}_{[\]} y = x \text{ in mod}_{[E]} y : [\](\text{Int} \rightarrow \text{Int}) \rightarrow [E](\text{Int} \rightarrow \text{Int})$$

4.6 Operational semantics

Figure 4.6 gives the operational semantics for MET including the definition of evaluation contexts.

Rule E-LETMOD eliminates a modality. Other reduction rules are standard for effect handlers [Hillerström et al. 2020]. In rule E-OP, we use the meta function $\text{bl}(\mathcal{E})$ to obtain the set of bound operation labels which have handlers installed in the evaluation context

Evaluation contexts $\mathcal{E} ::= [] \mid \mathcal{E} N \mid V \mathcal{E} \mid \text{do } \ell \mathcal{E} \mid \text{handle } \mathcal{E} \text{ with } H$	
E-APP	$(\lambda x^A.M) V \rightsquigarrow M[V/x]$
E-LETMOD	$\text{let}_\nu \text{ mod}_\mu x = \text{mod}_\mu V \text{ in } M \rightsquigarrow M[V/x]$
E-RET	$\text{handle } V \text{ with } H \rightsquigarrow N[V/x],$ where $H = \{\text{return } x \mapsto N\} \uplus \{\ell_i p_i r_i \mapsto N_i\}_i$
E-OP	$\text{handle } \mathcal{E}[\text{do } \ell V] \text{ with } H \rightsquigarrow N[V/p, (\lambda y. \text{handle } \mathcal{E}[y] \text{ with } H)/r],$ where $\ell \notin \text{bl}(\mathcal{E})$ and $(\ell p r \mapsto N) \in H$
E-LIFT	$\mathcal{E}[M] \rightsquigarrow \mathcal{E}[N],$ if $M \rightsquigarrow N$

Figure 4.6: Operational semantics for MET.

$\mathcal{E}$. It is inductively defined naturally as follows, where we write $\text{dom}(H)$ for the set of operation labels handled by H .

$$\begin{aligned}
\text{bl}([]) &= \emptyset & \text{bl}(\text{do } \ell \mathcal{E}) &= \text{bl}(\mathcal{E}) \\
\text{bl}(\mathcal{E} N) &= \text{bl}(\mathcal{E}) & \text{bl}(\text{handle } \mathcal{E} \text{ with } H) &= \text{bl}(\mathcal{E}) \cup \{\ell \mid \ell \in \text{dom}(H)\} \\
\text{bl}(V \mathcal{E}) &= \text{bl}(\mathcal{E})
\end{aligned}$$

The condition $\ell \notin \text{bl}(\mathcal{E})$ makes sure each operation ℓ is handled by the dynamically innermost handler of ℓ .

4.7 Metatheory

MET enjoys standard structural properties such as weakening by a variable binding and exchanging consecutive variable bindings. There are also some metatheoretical properties specific to modal type systems that are different from what we usually see in conventional type systems [Kavvos and Gratzer 2023]. For instance, we have the following admissible rule for weakening a lock of a context.

$$\frac{\Gamma, \mathbf{\mu}_\mu, \Gamma' \vdash M : A @ E \quad \Gamma @ F \quad \Gamma \vdash \mu \Rightarrow \nu @ F}{\Gamma, \mathbf{\mu}_\nu, \Gamma' \vdash M : A @ E'}$$

We use the modality transformation relation $\mu \Rightarrow \nu @ F$ to check whether we can weaken the lock $\mathbf{\mu}_\mu$ to $\mathbf{\mu}_\nu$ under the ambient effect context F . As a result of lock weakening, the ambient effect context E , which satisfies $\Gamma, \mathbf{\mu}_\mu, \Gamma' @ E$, may be changed to a larger effect context E' , which satisfies $\Gamma, \mathbf{\mu}_\nu, \Gamma' @ E'$. As another example, we have the following admissible rule for substituting values with modalities into terms.

$$\frac{\Gamma, \mathbf{\mu}_\mu \vdash V : A @ F \quad \Gamma, x :_\mu A, \Gamma' \vdash M : B @ E}{\Gamma, \Gamma' \vdash M[V/x] : B @ E}$$

The variable binding $x :_{\mu} A$ is indexed by a modality μ , which coincides with the modality of the lock $\blacksquare_{\mu}$ in the context of the value V being substituted. We prove these properties as well as further lemmas in Appendices A.1 and A.2.

To state syntactic type soundness, we first define normal forms, which are either values or terms stuck on an unhandled operation.

Definition 1 (Normal Forms). *We say a term M is in a normal form with respect to effect context E if it is either a value V or of the form $M = \mathcal{E}[\text{do } \ell \ V]$ for $\ell \in E$ and $\ell \notin \text{bl}(\mathcal{E})$.*

We have the following two theorems for syntactic type soundness.

Theorem 2 (Progress). *If $\Gamma \vdash M : A @ E$ and $\text{dom}(\Gamma) = \emptyset$, then either there exists N such that $M \rightsquigarrow N$ or M is in a normal form with respect to E .*

Theorem 3 (Subject Reduction). *If $\Gamma \vdash M : A @ E$ and $M \rightsquigarrow N$, then $\Gamma \vdash N : A @ E$.*

Progress states that a well-typed term is either reducible or in a normal form with respect to the ambient effect context. The condition $\text{dom}(\Gamma) = \emptyset$ rules out free term variables while allowing Γ to contain locks. Subject reduction states that the type of a well-typed term is preserved under reduction. Their proofs are in Appendices A.3 and A.4.

These two theorems together give type soundness and effect safety.

Theorem 4 (Type Soundness and Effect Safety). *If $\cdot \vdash M : A @ \cdot$, then either M diverges or there exists a value V such that $M \rightsquigarrow^* V$, where $\rightsquigarrow^*$ denotes the reflexive-transitive closure of the reduction relation $\rightsquigarrow$.*

Proof. By Theorems 2 and 3. □

4.8 Connections between MET and MTT

We have presented MET as a core calculus in a self-contained way. In this section we discuss its connections to multimodal type theory (MTT) [Gratzer 2023; Gratzer et al. 2020], the theoretical foundation of MET. Roughly speaking, MET extends simply-typed MTT [Kavvos and Gratzer 2023] with effect handlers and instantiates the mode theory of MTT to track effects. However, MET also generalises simply-typed MTT in several ways to improve its usability and to support subeffecting. In this section, we explain the connections between MET and MTT in more detail.

4.8.1 Indexed families of morphisms

As shown in Section 2.2, a modality in MTT is a morphism between two modes, transforming one mode to another. However, in MET, a modality is not a morphism between two specific modes; it can take any mode as input and return another mode as output. Thus, a modality μ in MET is actually an indexed family of morphisms between modes. We write μ_F for the pair of a modality μ and a mode F that μ acts on. This notation gives us a morphism $\mu_F : E \rightarrow F$ where $\mu(F) = E$. The morphism goes from E to F instead of the other direction to remain consistent with MTT. This also explains why in Section 4.3, modality composition $\mu \circ \nu$ reads from left to right.

In contexts, types, and terms, we use μ without indices. We do not lose any information by omitting the indices. Given a typing judgement $\Gamma \vdash M : A @ E$, the indices for all modalities in Γ , M , and A can be computed. This relies on the fact that for well-formedness of contexts, we require the empty context to be at the empty effect context, i.e., $\cdot @ \cdot$ instead of $\cdot @ E$ for any E , which is different from MTT.

Using indexed families of morphisms in MET instead of concrete morphisms is not an arbitrary choice. Using indexed families of modalities in types and terms is very useful to allow term variables to be used flexibly in effect contexts larger than those in which they are defined. This greatly simplifies the support of subeffecting; we do not need to update all indices of modalities in a term or type when upcasting this term or type to a larger effect context. As a result, every type is always well-defined at any mode, which means that we do not need to define the well-formedness judgement $A @ E$ as in MTT. Moreover, one important benefit of having types well-defined at any mode is that when adding polymorphism for values, type quantifiers do not need to carry the additional information about the modes at which the type variables can be used, greatly simplifying the type system. Otherwise, polymorphic types would need to have the form $\forall \alpha^{K @ E}. A$, where E indicates the mode of the type variable α .

4.8.2 The double category of effects

As shown in Section 2.2, the mode theory of MTT is a 2-category, where objects are modes, morphisms are modalities, and 2-cells are modality transformations. However, beyond these structures, MET also has a natural partial order between modes — subeffecting. A 2-category is not expressive enough to capture this partial order. We extend the mode theory to a double category in MET to support subeffecting. In this section, we use the notation μ_F since it corresponds to a modality in MTT. We fix a well-formed context Γ throughout and sometimes omit it from judgements to simplify the presentation.

A double category extends a 2-category with an additional kind of morphism. Alongside the regular morphisms, now called *horizontal* morphisms, there are also *vertical* morphisms that connect the objects of the 2-category. This makes it possible to generalise the 2-cells to transform arbitrary morphisms, whose source and target are connected by vertical morphisms. The following figure shows the differences between 2-cells in a 2-category and those in a double category using syntax of MET.

$$\begin{array}{ccc}
 E & \xrightarrow{\mu_F} & F \\
 \downarrow \Downarrow & & \downarrow \Downarrow \\
 E & \xrightarrow{\nu_F} & F
 \end{array}
 \qquad
 \begin{array}{ccc}
 E & \xrightarrow{\mu_F} & F \\
 \downarrow \leq & \Downarrow & \downarrow \leq \\
 E' & \xrightarrow{\nu_{F'}} & F'
 \end{array}$$

In MET, objects are given by effect contexts (i.e., modes), the horizontal morphisms by modalities, the vertical morphisms by the subeffecting relation, and 2-cells by the modality transformations. We prove that this structure indeed forms a double category.

First, we need to show that effect contexts (objects) and subeffecting (vertical morphisms) form a category. This is obvious since the subeffecting relation is a partial order. We have the following lemma.

Lemma 5. *Effect contexts and subeffecting form a category with the identity morphisms $E \leq E$ and the morphism composition given by transitivity of subeffecting.*

Proof. It is easy to show that effect contexts and subeffecting form a poset. Every poset is a category. $\square$

Second, we need to show that effect contexts (objects) and modalities (horizontal morphisms) also form a category. We repeat the definition of modalities and modality composition from Section 4.3 here using the syntax μ_F for ease of reference:

$$\begin{aligned}
 [E]_F : E &\rightarrow F & \langle D \rangle_F : D, F &\rightarrow F \\
 \mu_F \circ [E]_{\mu(F)} &= [E]_F & [E]_F \circ \langle D \rangle_E &= [D, E]_F & \langle D_1 \rangle_F \circ \langle D_2 \rangle_{D_1, F} &= \langle D_2, D_1 \rangle_F
 \end{aligned}$$

The effect contexts and modalities naturally form a category since modality composition possesses identity and associativity. We have the following lemma.

Lemma 6. *Effect contexts and modalities form a category with the identity morphisms $\langle \rangle_E : E \rightarrow E$ and the morphism composition $\mu_F \circ \nu_{F'}$ such that*

1. *Identity:* $\langle \rangle_F \circ \mu_F = \mu_F = \mu_F \circ \langle \rangle_E$ for $\mu_F : E \rightarrow F$.
2. *Associativity:* $(\mu_{E_1} \circ \nu_{E_2}) \circ \xi_{E_3} = \mu_{E_1} \circ (\nu_{E_2} \circ \xi_{E_3})$ for $\mu_{E_1} : E_2 \rightarrow E_1$, $\nu_{E_2} : E_3 \rightarrow E_2$, and $\xi_{E_3} : E \rightarrow E_3$.

Proof. By inlining the definitions of modalities and checking each case. $\square$

Finally, we need to show that modality transformations are 2-cells in the double category of effects. We first repeat the definition of modality transformations from Section 4.3 here using the syntax μ_F for ease of reference:

$$\begin{array}{c} \text{MT-ABS} \\ \hline \mu_F : E' \rightarrow F \quad \Gamma \vdash E \leq E' \\ \hline \Gamma \vdash [E]_F \Rightarrow \mu_F \end{array} \qquad \begin{array}{c} \text{MT-REL} \\ \hline \Gamma \vdash \langle D \rangle_F \Rightarrow \langle D \rangle_F \end{array}$$

We have required the effect context that the source and target modalities act on to be the same F in these rules. This is enough for presenting the calculus, but we can further extend it to allow $\mu_F \Rightarrow \nu_{F'}$ where $F \leq F'$. This is used in the metatheory for MET such as the lock weakening lemma we discussed in Section 4.7. The extended modality transformation relation is defined by the transitive closure of the following rules. Compared to the definition in Section 4.3, the only new rule is MT-MONO.

$$\begin{array}{c} \text{MT-ABS} \\ \hline \mu_F : E' \rightarrow F \quad \Gamma \vdash E \leq E' \\ \hline \Gamma \vdash [E]_F \Rightarrow \mu_F \end{array} \qquad \begin{array}{c} \text{MT-REL} \\ \hline \Gamma \vdash \langle D \rangle_F \Rightarrow \langle D \rangle_F \end{array} \qquad \begin{array}{c} \text{MT-MONO} \\ \hline \Gamma \vdash F \leq F' \\ \hline \Gamma \vdash \mu_F \Rightarrow \mu_{F'} \end{array}$$

The following lemma shows that the transformation $\Gamma \vdash \mu_F \Rightarrow \nu_{F'}$ satisfies the requirement of being a 2-cell in the double category of effects with well-defined vertical and horizontal composition.

Lemma 7 (Modality transformations are 2-cells). *If $\Gamma \vdash \mu_F \Rightarrow \nu_{F'}$, $\mu_F : E \rightarrow F$, and $\nu_{F'} : E' \rightarrow F'$, then $\Gamma \vdash E \leq E'$ and $\Gamma \vdash F \leq F'$. Moreover, the transformation relation is closed under vertical and horizontal composition as shown by the following admissible rules.*

$$\begin{array}{c} \Gamma \vdash \mu_{F_1} \Rightarrow \nu_{F_2} \quad \Gamma \vdash \nu_{F_2} \Rightarrow \xi_{F_3} \\ \hline \Gamma \vdash \mu_{F_1} \Rightarrow \xi_{F_3} \end{array} \qquad \begin{array}{c} \Gamma \vdash \mu_F \Rightarrow \mu'_{F'} \quad \Gamma \vdash \nu_E \Rightarrow \nu'_{E'} \\ \mu_F : E \rightarrow F \quad \mu'_{F'} : E' \rightarrow F' \\ \hline \Gamma \vdash \mu_F \circ \nu_E \Rightarrow \mu'_{F'} \circ \nu'_{E'} \end{array}$$

Proof. Straightforward case analysis. □

We have proved that the minimal mode theory presented in Section 4.3 forms a double category. It is not hard to verify that with extensions such as masks and presence types in Sections 5.1 and 5.2, the mode theory still forms a double category.

5 Extensions

How much can we add to the language without losing the semantics?

—Paul Blain Levy (2001)

We add several extensions to **MET**, including both standard type system features and specialised features for algebraic effects and handlers. We introduce masks, presence types, and parameterised effects, which make the mode theory and effect systems more expressive. We demonstrate that **MET** scales to support data types and polymorphism, including polymorphism for both value types and effects. Effect polymorphism helps deal with situations in which it is useful to refer to one or more effect contexts that differ from the ambient one (such as the higher-order fork operation in Section 3.9), recovering the full expressive power of row-based effect systems. We also introduce a minimal extension of **MET** with local labels, which allows us to generate operation labels dynamically [de Vilhena and Pottier 2023], and extensions with richer forms of handlers.

5.1 Masks

Masks allow us to remove labels from the ambient effect context. They are useful for solving the effect pollution and accidental handling problems as discussed in Section 3.6.

Figure 5.1 defines the extensions of the mode theory with masks. Masks L are simply multisets of labels. Well-formedness $\Gamma \vdash L$ ensures that all labels in masks are defined in the global label context Σ . Equivalence on masks, written $\Gamma \vdash L \equiv L'$, is just equivalence of well-formed multisets. We define two meta-operations $E - L$ and $L - D$ in order to define the mode theory for the extension with masks. The operation $E - L$ removes the labels in L from E . The operation $L - D$ removes the labels in D from L .

Modality action $\mu(E)$ and composition $\mu \circ \nu$ are extended naturally. For $\langle L|D \rangle(E)$, we remove the labels in L from E and then extend it with D . For $[E] \circ \langle L|D \rangle$, replacing the effect context with E and then masking L and extending it with D is equivalent to just replacing it with $D, (E - L)$. For $\langle L_1|D_1 \rangle \circ \langle L_2|D_2 \rangle$, sequential masking and extension can be combined into one modality by cancelling the overlapping part of L_2 and D_1 with

Syntax

Masks $L ::= \cdot \mid \ell, L$ Modalities $\mu, \nu ::= [E] \mid \langle L \mid D \rangle$

Well-formedness and equivalence

$$\boxed{\Gamma \vdash L} \quad \boxed{\Gamma \vdash \mu} \quad \boxed{\Gamma \vdash \mu \equiv \nu}$$

$$\frac{}{\Gamma \vdash \cdot} \quad \frac{\ell \in \text{dom}(\Sigma) \quad \Gamma \vdash L}{\Gamma \vdash \ell, L} \quad \frac{\Gamma \vdash L \quad \Gamma \vdash D}{\Gamma \vdash \langle L \mid D \rangle} \quad \frac{\Gamma \vdash L \equiv L' \quad \Gamma \vdash D \equiv D'}{\Gamma \vdash \langle L \mid D \rangle \equiv \langle L' \mid D' \rangle}$$

Meta-operations on effects $\boxed{E - L} \quad \boxed{L - D}$

$$\begin{aligned} \cdot - L &= \cdot \\ (\ell, E) - L &= \begin{cases} E - L' & \text{if } L \equiv \ell, L' \\ \ell, (E - L) & \text{otherwise} \end{cases} \end{aligned} \quad \begin{aligned} \cdot - D &= \cdot \\ (\ell, L) - D &= \begin{cases} L - D' & \text{if } D \equiv \ell, D' \\ \ell, (L - D) & \text{otherwise} \end{cases} \end{aligned}$$

Modality action $\boxed{\mu(E)}$

$$[E](F) = E \quad \langle L \mid D \rangle(F) = D, (F - L)$$

Modality composition $\boxed{\mu \circ \nu}$

$$\begin{aligned} \mu \circ [E] &= [E] \\ [E] \circ \langle L \mid D \rangle &= [D, (E - L)] \\ \langle L_1 \mid D_1 \rangle \circ \langle L_2 \mid D_2 \rangle &= \langle L_1, (L_2 - D_1) \mid D_2, (D_1 - L_2) \rangle \end{aligned}$$

Modality transformation $\boxed{\Gamma \vdash \mu \Rightarrow \nu @ F}$

$$\begin{array}{ccc} \text{MT-ABS} & \text{MT-REL} & \text{MT-EXPAND} \\ \frac{\Gamma \vdash E \leq \mu(F)}{\Gamma \vdash [E] \Rightarrow \mu @ F} & \frac{}{\Gamma \vdash \langle L \mid D \rangle \Rightarrow \langle L \mid D \rangle @ F} & \frac{\Gamma \vdash (F - L) \equiv \ell, E}{\Gamma \vdash \langle L \mid D \rangle \Rightarrow \langle \ell, L \mid D, \ell \rangle @ F} \\ \text{MT-SHRINK} & \text{MT-TRANS} & \\ \frac{\Gamma \vdash (F - L) \equiv \ell, E}{\Gamma \vdash \langle \ell, L \mid D, \ell \rangle \Rightarrow \langle L \mid D \rangle @ F} & \frac{\Gamma \vdash \mu \Rightarrow \nu @ F \quad \Gamma \vdash \nu \Rightarrow \xi @ F}{\Gamma \vdash \mu \Rightarrow \xi @ F} & \end{array}$$

Figure 5.1: Masks.

the meta-operations $L_2 - D_1$ and $D_1 - L_2$ as $\langle L_1, (L_2 - D_1) \mid D_2, (D_1 - L_2) \rangle$. For instance, we have $\langle \ell \rangle \circ \langle \ell \rangle = \langle \rangle$ and $\langle \ell_1 \mid \ell_2, \ell_3 \rangle \circ \langle \ell_2 \mid \ell_4 \rangle = \langle \ell_1 \mid \ell_4, \ell_3 \rangle$.

Modality transformation is extended with three new rules. MT-EXPAND allows us to simultaneously mask and extend some operations given that these operations exist in the ambient effect context F . MT-SHRINK allows us to do the reverse for any operations. MT-TRANS is a standard transitivity rule for modality transformation. It enables the sequential application of MT-EXPAND and MT-SHRINK. For instance, we have $\langle \ell_1 \mid \ell_1 \rangle \Rightarrow \langle \ell_2 \mid \ell_2 \rangle @ \ell_1, \ell_2$ by first applying MT-SHRINK to simultaneously remove ℓ_1 from both the mask and extension, then applying MT-EXPAND to simultaneously add ℓ_2 .

Syntax

Terms $M, N ::= \dots \mid \text{mask}_L M$ Kinding $\boxed{\Gamma \vdash A : K}$

$$\frac{\Gamma \vdash \langle L \mid D \rangle \quad \Gamma \vdash A : K}{\Gamma \vdash \langle L \mid D \rangle A : K}$$

Typing $\boxed{\Gamma \vdash M : A @ E}$

$$\frac{\text{T-MASK} \quad \Gamma, \blacksquare_{\langle L \rangle} \vdash M : A @ E - L}{\Gamma \vdash \text{mask}_L M : \langle L \rangle A @ E}$$

Operational semantics $\boxed{M \rightsquigarrow N}$ Evaluation contexts $\mathcal{E} ::= \dots \mid \text{mask}_L \mathcal{E}$

$$\begin{array}{l} \text{E-MASK} \quad \text{mask}_L V \rightsquigarrow \text{mod}_{\langle L \rangle} V \\ \text{E-OP} \quad \text{handle } \mathcal{E}[\text{do } \ell V] \text{ with } H \rightsquigarrow N[V/p, (\lambda y. \text{handle } \mathcal{E}[y] \text{ with } H)/r], \\ \quad \text{where } 0\text{-free}(\ell, \mathcal{E}) \text{ and } (\ell \ p \ r \mapsto N) \in H \end{array}$$

Figure 5.1: Masks (continued).

Figure 5.1 defines the extensions to kinding, typing, and operational semantics for masks. The kinding rule is standard. In T-MASK, the mask $\text{mask}_L M$ removes effects L from the ambient effect context for M and introduces a lock $\blacksquare_{\langle L \rangle}$ to the context of the premise. For the return value of M , we need to box it with $\langle L \rangle$ to reconcile the mismatch between $E - L$ and E . In E-MASK, we also reduce a mask on a value to the introduction of the corresponding mask modality. To define the semantics of masks, we follow [Biernacki et al. \[2018\]](#) and define the predicate $n\text{-free}(\ell, \mathcal{E})$ inductively on evaluation contexts as follows.

$$\begin{array}{c} \frac{}{0\text{-free}(\ell, [\])} \quad \frac{n\text{-free}(\ell, \mathcal{E})}{n\text{-free}(\ell, \text{do } \ell' \mathcal{E})} \quad \frac{n\text{-free}(\ell, \mathcal{E}) \quad \text{count}(\ell; L) = m}{(n + m)\text{-free}(\ell, \text{mask}_L \mathcal{E})} \\[10pt] \frac{(n + 1)\text{-free}(\ell, \mathcal{E}) \quad \ell \in \text{dom}(H)}{n\text{-free}(\ell, \text{handle } \mathcal{E} \text{ with } H)} \quad \frac{n\text{-free}(\ell, \mathcal{E}) \quad \ell \notin \text{dom}(H)}{n\text{-free}(\ell, \text{handle } \mathcal{E} \text{ with } H)} \end{array}$$

We have $n\text{-free}(\ell, \mathcal{E})$ if there are n masks for ℓ in $\mathcal{E}$ without corresponding handlers. The E-OP rule requires $0\text{-free}(\ell, \mathcal{E})$ to guarantee that the current handler is not masked by masks in $\mathcal{E}$. The meta function $\text{count}(\ell; L)$ yields the number of ℓ labels in L . We omit the inductive cases that do not change n .

Since masks are a basic and important extension when treating effects as scoped rows, we base all subsequent extensions in this chapter on the version of MET extended with masks.

5.2 Presence types

Instead of having a global label context which associates operation labels with their signatures, we can also assign different signatures to labels in types. We can further extend this idea to allow some labels to be absent. This is usually referred to as presence types [Rémy 1994] and adopted in the Links language [Hillerström and Lindley 2016].

Figure 5.2 gives the extension with presence types. We mostly need to extend the mode theory. We now distinguish between the syntactic categories of extensions and effect contexts as they have different equivalence and subeffecting relations. Each label has a presence type P [Rémy 1994]. A presence type P can be an operation arrow of the form $A \rightarrow B$, which indicates that the operation takes an argument of type A and returns a value of type B , or the absent type $-$, which indicates that the operation with this label cannot be invoked. Extensions and effect contexts both assign presence types to labels.

For the equivalence of effect contexts, we allow absent labels to be freely added to or removed from the right of effect contexts. For instance, $\ell : P, \ell' : -$ is equivalent to $\ell : P$. We can think of an effect context as denoting a map from labels to infinite sequences of presence types, each with a cofinite tail containing only $-$. For extensions, we do not ignore labels with absent types when defining equivalence, so $\ell : P, \ell' : -$ and $\ell : P$ are distinct.

We define the subeffecting relation differently from before to account for presence types. We have $E \leq F$ if we can replace the absent types in E with proper operation arrows to obtain F . We also have a subtyping relation on extensions $D \leq D'$. Like subeffecting on effect contexts, it requires D and D' to contain the same row of labels, but it allows absent types in D to be replaced by concrete signatures in D' .

We extend the meta-operations on effects from Figure 5.1 to account for presence types. Since we distinguish between the syntactic categories of extensions and effect contexts, we define $E - L$ and $D - L$ separately, though their definitions are identical. The definitions of modality action and composition remain unchanged. For modality transformation, we modify rules MT-EXPAND and MT-SHRINK to account for presence types and rule MT-REL to allow upcasting presence types in extensions for relative modalities. For instance, $\langle \ell : - \rangle \Rightarrow \langle \ell : A \rightarrow B \rangle @ E$ is allowed. It is easy to verify that Lemma 1 still holds.

Syntax

Presence $P ::= A \rightarrow B \mid -$ Extensions $D ::= \cdot \mid \ell : P, D$ Effect contexts $E, F ::= \cdot \mid \ell : P, E$ Well-formedness $\boxed{\Gamma \vdash D} \boxed{\Gamma \vdash E} \boxed{\Gamma \vdash P}$

$$\frac{}{\Gamma \vdash \cdot} \quad \frac{\boxed{\Gamma \vdash P} \quad \Gamma \vdash D}{\Gamma \vdash \ell : P, D} \quad \frac{\boxed{\Gamma \vdash P} \quad \Gamma \vdash E}{\Gamma \vdash \ell : P, E} \quad \boxed{\Gamma \vdash -} \quad \frac{\Gamma \vdash A : \text{Abs} \quad \Gamma \vdash B : \text{Abs}}{\Gamma \vdash A \rightarrow B}$$

Equivalence $\boxed{\Gamma \vdash D \equiv D'} \boxed{\Gamma \vdash E \equiv E'} \boxed{\Gamma \vdash P \equiv P'}$

$$\frac{}{\Gamma \vdash D \equiv D} \quad \frac{\Gamma \vdash D_1 \equiv D_2 \quad \Gamma \vdash D_2 \equiv D_3}{\Gamma \vdash D_1 \equiv D_3} \quad \frac{\boxed{\Gamma \vdash P \equiv P'} \quad \Gamma \vdash D \equiv D'}{\Gamma \vdash \ell : P, D \equiv \ell : P', D'} \quad \frac{\ell \neq \ell'}{\Gamma \vdash \ell : P, \ell' : P', D \equiv \ell' : P', \ell : P, D}$$

$$\frac{}{\Gamma \vdash E \equiv E} \quad \frac{\Gamma \vdash E_1 \equiv E_2 \quad \Gamma \vdash E_2 \equiv E_3}{\Gamma \vdash E_1 \equiv E_3} \quad \frac{\boxed{\Gamma \vdash P \equiv P'} \quad \Gamma \vdash E \equiv E'}{\Gamma \vdash \ell : P, E \equiv \ell : P', E'} \quad \frac{\ell \neq \ell'}{\Gamma \vdash \ell : P, \ell' : P', E \equiv \ell' : P', \ell : P, E}$$

$$\boxed{\Gamma \vdash E, \ell : - \equiv E} \quad \boxed{\Gamma \vdash - \equiv -} \quad \frac{\Gamma \vdash A \equiv A' \quad \Gamma \vdash B \equiv B'}{\Gamma \vdash A \rightarrow B \equiv A' \rightarrow B'}$$

Subeffecting $\boxed{\Gamma \vdash D \leq D'} \boxed{\Gamma \vdash E \leq E'} \boxed{\Gamma \vdash P \leq P'}$

$$\frac{}{\Gamma \vdash \cdot \leq E} \quad \frac{\Gamma \vdash E_1 \equiv \ell : P_1, E'_1 \quad \Gamma \vdash E_2 \equiv \ell : P_2, E'_2 \quad \Gamma \vdash P_1 \leq P_2 \quad \Gamma \vdash E'_1 \leq E'_2}{\Gamma \vdash E_1 \leq E_2}$$

$$\frac{}{\Gamma \vdash \cdot \leq \cdot} \quad \frac{\Gamma \vdash D_1 \equiv \ell : P_1, D'_1 \quad \Gamma \vdash D_2 \equiv \ell : P_2, D'_2 \quad \Gamma \vdash P_1 \leq P_2 \quad \Gamma \vdash D'_1 \leq D'_2}{\Gamma \vdash D_1 \leq D_2}$$

$$\boxed{\Gamma \vdash P \leq P} \quad \boxed{\Gamma \vdash - \leq P}$$

Meta-operations on effect contexts $\boxed{E - L} \boxed{D - L} \boxed{L - D}$

$$(\ell : P, E) - L = \begin{cases} E - L' & \text{if } L \equiv \ell, L' \\ \ell : P, (E - L) & \text{otherwise} \end{cases} \quad \begin{matrix} \cdot - L = \cdot \\ \cdot - D = \cdot \end{matrix}$$

$$(\ell : P, D) - L = \begin{cases} D - L' & \text{if } L \equiv \ell, L' \\ \ell : P, (D - L) & \text{otherwise} \end{cases} \quad (\ell, L) - D = \begin{cases} L - D' & \text{if } D \equiv \ell : P, D' \\ \ell, (L - D) & \text{otherwise} \end{cases}$$

Modality transformation $\boxed{\Gamma \vdash \mu \Rightarrow \nu @ F}$

$$\frac{\text{MT-ABS} \quad \Gamma \vdash E \leq \mu(F)}{\Gamma \vdash [E] \Rightarrow \mu @ F} \quad \frac{\text{MT-REL} \quad \Gamma \vdash D \leq D'}{\Gamma \vdash \langle L | D \rangle \Rightarrow \langle L | D' \rangle @ F} \quad \frac{\text{MT-EXPAND} \quad \Gamma \vdash (F - L) \equiv \ell : A \rightarrow B, E}{\Gamma \vdash \langle L | D \rangle \Rightarrow \langle \ell, L | D, \ell : A \rightarrow B \rangle @ F}$$

$$\frac{\text{MT-SHRINK} \quad \Gamma \vdash (F - L) \equiv \ell : P, E}{\Gamma \vdash \langle \ell, L | D, \ell : P \rangle \Rightarrow \langle L | D \rangle @ F} \quad \frac{\text{MT-TRANS} \quad \Gamma \vdash \mu \Rightarrow \nu @ F \quad \Gamma \vdash \nu \Rightarrow \xi @ F}{\Gamma \vdash \mu \Rightarrow \xi @ F}$$

Figure 5.2: Presence types. The subeffecting judgements are new; changes to the other judgements are highlighted in grey.

5.3 Crisp data types

We demonstrate the extensibility of MET with data types by extending it with pair and sum types. It is straightforward to extend MET further with algebraic data types. Figure 5.3 gives the syntax, kinding, equivalence, typing, and operational semantics for the extension with pair and sum types.

Syntax

Types $A, B ::= \dots \mid A \times B \mid A + B$
 Terms $M, N ::= \dots \mid (M, N) \mid \text{case}_\nu V \text{ of } (x, y) \mapsto M$
 $\mid \text{inl } M \mid \text{inr } M \mid \text{case}_\nu V \text{ of } \{\text{inl } x \mapsto M, \text{inr } y \mapsto N\}$
 Values $V, W ::= \dots \mid (V, W) \mid \text{inl } V \mid \text{inr } V$

Kinding and equivalence $\boxed{\Gamma \vdash A : K} \quad \boxed{\Gamma \vdash A \equiv B}$

$\frac{\Gamma \vdash A : K \quad \Gamma \vdash B : K}{\Gamma \vdash A \times B : K}$	$\frac{\Gamma \vdash A : K \quad \Gamma \vdash B : K}{\Gamma \vdash A + B : K}$	$\frac{\Gamma \vdash A_1 \equiv A_2 \quad \Gamma \vdash B_1 \equiv B_2}{\Gamma \vdash A_1 \times B_1 \equiv A_2 \times B_2}$	$\frac{\Gamma \vdash A_1 \equiv A_2 \quad \Gamma \vdash B_1 \equiv B_2}{\Gamma \vdash A_1 + B_1 \equiv A_2 + B_2}$
--	---	--	--

Typing $\boxed{\Gamma \vdash M : A @ E}$

$\frac{\text{T-PAIR} \quad \Gamma \vdash M : A @ E \quad \Gamma \vdash N : B @ E}{\Gamma \vdash (M, N) : A \times B @ E}$	$\frac{\text{T-INL} \quad \Gamma \vdash M : A @ E}{\Gamma \vdash \text{inl } M : A + B @ E}$	$\frac{\text{T-INR} \quad \Gamma \vdash M : B @ E}{\Gamma \vdash \text{inr } M : A + B @ E}$
$\frac{\text{T-CRISPPAIR} \quad \nu(F) = E \quad \Gamma, \mathbf{\blacklozenge}_\nu \vdash V : A \times B @ E \quad \Gamma, x :_\nu A, y :_\nu B \vdash M : A' @ F}{\Gamma \vdash \text{case}_\nu V \text{ of } (x, y) \mapsto M : A' @ F}$	$\frac{\text{T-CRISPSUM} \quad \nu(F) = E \quad \Gamma, \mathbf{\blacklozenge}_\nu \vdash V : A + B @ E \quad \Gamma, x :_\nu A \vdash M_1 : A' @ F \quad \Gamma, y :_\nu B \vdash M_2 : A' @ F}{\Gamma \vdash \text{case}_\nu V \text{ of } \{\text{inl } x \mapsto M_1, \text{inr } y \mapsto M_2\} : A' @ F}$	

Operational semantics $\boxed{M \rightsquigarrow N}$

Evaluation contexts $\mathcal{E} ::= \dots \mid (\mathcal{E}, N) \mid (V, \mathcal{E}) \mid \text{case}_\nu \mathcal{E} \text{ of } (x, y) \mapsto M$
 $\mid \text{inl } \mathcal{E} \mid \text{inr } \mathcal{E} \mid \text{case}_\nu \mathcal{E} \text{ of } \{\text{inl } x \mapsto M, \text{inr } y \mapsto N\}$

E-PAIR $\text{case}_\mu (V_1, V_2) \text{ of } (x, y) \mapsto N \rightsquigarrow N[V_1/x, V_2/y]$
 E-INL $\text{case}_\mu \text{inl } V \text{ of } \{\text{inl } x \mapsto N_1, \dots\} \rightsquigarrow N_1[V/x]$
 E-INR $\text{case}_\mu \text{inr } V \text{ of } \{\text{inr } y \mapsto N_2, \dots\} \rightsquigarrow N_2[V/y]$

Figure 5.3: Pairs and sums.

Rules T-PAIR, T-INL, and T-INR are standard introduction rules. The elimination rules T-CRISPPAIR and T-CRISPSUM are more interesting. In addition to ordinary pattern matching, they interpret the value V under the effect context transformed by the modality ν , which can then be attached to the variable bindings in the case clauses. They follow the crisp induction principles of multimodal type theory [Gratzer et al. 2020; Shulman 2018; Gratzer et al. 2021]. These crisp elimination rules provide extra expressiveness. For

example, we can write the following function, which transforms a sum of type $\mu(A + B)$ into a sum of type $(\mu A + \mu B)$. This function is not expressible without crisp elimination rules.

$$\lambda x^{\mu(A+B)}. \text{let } \text{mod}_\mu y = x \text{ in case}_\mu y \text{ of } \{\text{inl } x_1 \mapsto \text{inl } (\text{mod}_\mu x_1), \text{inr } x_2 \mapsto \text{inr } (\text{mod}_\mu x_2)\}$$

5.4 Polymorphism

Figure 5.4 gives the extension of MET with polymorphism for values and effects.

To support effect polymorphism, we extend the syntax of effect contexts E with effect variables ε and introduce a new kind **Effect** for them. As is typical for row polymorphism, we restrict each effect type to contain at most one effect variable. We also extend the syntax with effect masking $E \setminus L$, which denotes the effect type obtained by masking L from E . The latter is needed to keep the syntax of effect contexts closed under the masking operation $E - L$; otherwise we cannot define $\varepsilon - L$. In other words, the syntax of effects is the free algebra generated by effect extension D, E and masking $E \setminus L$, with the empty effect $\cdot$ and effect variables ε as base elements. With the new syntax $E \setminus L$, we directly redefine the meta-operation $E - L$ as $E \setminus L$.

The effect equivalence is extended in a relatively standard way. We do not allow non-trivial equivalence or subtyping between different effect variables. We always identify effects up to the equivalence relation. That is, we can directly treat the syntax of effects as the free algebra quotiented by the equivalence relation $E \equiv F$. Observe that using the equivalence relation, all open effect types with the effect variable ε can be simplified to an equivalent normal form $D, \varepsilon \setminus L$. Subeffecting is unchanged from core MET in Section 4.3 and thus omitted.

The kinding rule checks the body of $\forall \alpha^K. A$ under $\alpha : \text{Abs}$ when $K \neq \text{Effect}$. Equivalently, a polymorphic type has kind **Abs** if any of its instantiations has kind **Abs**. This is reminiscent of how the linearity of a polymorphic type is determined in the qualified linear type system of Tang et al. [2024]. For example, $\forall \alpha^{\text{Any}}. \alpha$ has kind **Abs**, even though its variable can be instantiated with any type such as a function type. It does not break soundness because this type has no closed values.

The typing and operational semantics rules for polymorphism are standard.

It may appear surprising that we treat a type application $V A$ as a value. This permits type instantiation within modality introduction, which restricts the term to be a value. For instance, $\text{mod}_{[E]} ((\lambda \alpha. V) A)$ is safe. The evaluation context $\text{let}_v \text{mod}_\mu x = \mathcal{E} \text{ in } M$ allows a type application to reduce. We could also allow values to reduce inside

Syntax

Types	$A, B ::= \dots \mid \alpha \mid \forall \alpha^K. A$	Contexts	$\Gamma ::= \dots \mid \Gamma, \alpha : K$
Effects	$E ::= \dots \mid \varepsilon \mid E \setminus L$	Terms	$M, N ::= \dots \mid \Lambda \alpha^K. V \mid M A$
Kinds	$K ::= \dots \mid \text{Effect}$	Values	$V, W ::= \dots \mid \Lambda \alpha^K. V \mid V A$

Kinding $\boxed{\Gamma \vdash A : K} \quad \boxed{\Gamma \vdash E : \text{Effect}}$

$\frac{\Gamma \ni \alpha : K}{\Gamma \vdash \alpha : K}$	$\frac{K \neq \text{Effect} \quad \Gamma, \alpha : \text{Abs} \vdash A : K'}{\Gamma \vdash \forall \alpha^K. A : K'}$	$\frac{\Gamma, \varepsilon : \text{Effect} \vdash A : K'}{\Gamma \vdash \forall \varepsilon^{\text{Effect}}. A : K'}$
$\frac{\Gamma \vdash E : \text{Effect}}{\Gamma \vdash [E]}$	$\frac{}{\Gamma \vdash \cdot : \text{Effect}}$	$\frac{\ell \in \text{dom}(\Sigma) \quad \Gamma \vdash E : \text{Effect}}{\Gamma \vdash \ell, E : \text{Effect}}$
		$\frac{\Gamma \vdash E : \text{Effect} \quad \Gamma \vdash L}{\Gamma \vdash E \setminus L : \text{Effect}}$

Contexts $\boxed{\Gamma @ E} \quad \boxed{\Gamma \equiv \Gamma' @ E}$

$\frac{\Gamma @ E}{\Gamma, \alpha : K @ E}$	$\frac{\Gamma \equiv \Gamma' @ E}{\Gamma, \alpha : K \equiv \Gamma', \alpha : K @ E}$	$\text{locks}(\Gamma, \alpha : K) = \text{locks}(\Gamma)$
---	---	---

Equivalence $\boxed{\Gamma \vdash A \equiv B} \quad \boxed{\Gamma \vdash E \equiv F}$

$\frac{}{\Gamma \vdash \alpha \equiv \alpha}$	$\frac{\Gamma, \alpha : K \vdash A \equiv B}{\Gamma \vdash \forall \alpha^K. A \equiv \forall \alpha^K. B}$	$\frac{}{\Gamma \vdash \varepsilon \setminus L \equiv \varepsilon \setminus L}$	$\frac{\Gamma \vdash E \equiv E' \quad \Gamma \vdash L \equiv L'}{\Gamma \vdash E \setminus L \equiv E' \setminus L'}$
$\frac{}{\Gamma \vdash E \setminus \cdot \equiv E}$	$\frac{}{\Gamma \vdash \cdot \setminus L \equiv \cdot}$	$\frac{}{\Gamma \vdash (\ell, E) \setminus (\ell, L) \equiv E \setminus L}$	$\frac{\ell \notin L}{\Gamma \vdash (\ell, E) \setminus L \equiv \ell, E \setminus L}$
$\frac{}{\Gamma \vdash (\varepsilon \setminus L) \setminus L' \equiv \varepsilon \setminus (L, L')}$			

Meta-operations on effects $\boxed{E - L}$

$$E - L = E \setminus L$$

Typing $\boxed{\Gamma \vdash M : A @ E}$

$\frac{\text{T-TABS} \quad \Gamma, \alpha : K \vdash V : A @ E}{\Gamma \vdash \Lambda \alpha^K. V : \forall \alpha^K. A @ E}$	$\frac{\text{T-TAPP} \quad \Gamma \vdash M : \forall \alpha^K. B @ E \quad \Gamma \vdash A : K}{\Gamma \vdash M A : B[A/\alpha] @ E}$
---	---

Operational semantics $\boxed{M \rightsquigarrow N}$

Evaluation contexts $\mathcal{E} ::= \dots \mid \mathcal{E} A \mid \text{let}_\nu \text{mod}_\mu x = \mathcal{E} \text{ in } M$

E-TAPP $(\Lambda \alpha^K. V) A \rightsquigarrow V[A/\alpha]$

Figure 5.4: Polymorphism.

modalities, so that the type application inside $\text{mod}_{[E]}$ in the previous example can reduce. This extension is not necessary, so we omit it here for simplicity.

5.5 Commuting modalities and type abstraction

Crisp elimination rules in Section 5.3 allow us to commute modalities and data type constructors, e.g., from $\mu(A + B)$ to $\mu A + \mu B$. Similarly, it is also sound and useful to commute type abstractions and modalities. However, the current modality elimination rule cannot do so, for a similar reason to why it is not possible to transform $\forall \alpha. A + B$ to $(\forall \alpha. A) + (\forall \alpha. B)$ in System F. We extend modality elimination to the form $\text{let}_\nu \text{mod}_\mu \Lambda \alpha^K. x = V \text{ in } M$, which allows V to use additional type variables in α^K ; these variables are abstracted when V is bound to x . For instance, we can now write a function of type $(\forall \alpha^K. \mu A) \rightarrow \mu(\forall \beta^K. A[\beta/\alpha])$ as follows.

$$\lambda x^{\forall \alpha^K. \mu A}. \text{let}_\nu \text{mod}_\mu \Lambda \beta^K. y = x \beta \text{ in } \text{mod}_\mu y$$

Figure 5.5 shows the extension to syntax, typing, and semantics.

Syntax

$$\text{Terms } M, N ::= \dots \mid \text{let}_\nu \text{mod}_\mu \Lambda \alpha^K. x = V \text{ in } M$$

Typing $\boxed{\Gamma \vdash M : A @ E}$

$$\frac{\text{T-LETMOD}' \quad \nu(F) = E \quad \Gamma, \alpha : \overline{K} \vdash V : \mu A @ E \quad \Gamma, x : \nu \circ \mu \forall \alpha^K. A \vdash M : B @ F}{\Gamma \vdash \text{let}_\nu \text{mod}_\mu \Lambda \alpha^K. x = V \text{ in } M : B @ F}$$

Operational semantics $\boxed{M \rightsquigarrow N}$

$$\text{E-LETMOD}' \quad \text{let}_\nu \text{mod}_\mu \Lambda \alpha^K. x = \text{mod}_\mu V \text{ in } M \rightsquigarrow M[(\Lambda \alpha^K. V)/x]$$

Figure 5.5: Commuting modalities and type abstractions.

5.6 Local labels

Local labels enable us to generate fresh operation labels with a local scope. This extension is inspired by the local effects of [Biernacki et al. \[2019\]](#), dynamic labels of [de Vilhena and Pottier \[2023\]](#), and fresh labels of the Links language [[Jourde et al. 2022](#)]. It enables us to encode named handlers [[Xie et al. 2022](#); [Brachthäuser et al. 2020](#)] as we will see in Chapter 7.

Figure 5.6 gives the syntax, typing, and operational semantics for the extension with local labels. Well-formedness rules of masks, extensions, and effect contexts are extended to permit local labels from Γ . Typing rule T-LOCALEFFECT binds a fresh local label ℓ with type signature $A \rightarrow B$. This label ℓ appears in the context of the typing judgement for

the term M . (As is standard for term and type variables, we also implicitly adopt the Barendregt convention for local labels, applying α -conversion as necessary to ensure that ℓ is fresh.) Well-formedness of type A' and effect context E under Γ ensures that ℓ cannot appear in A' or E . Rules T-Do and T-HANDLE are modified to permit any label from either Σ or Γ .

We adopt the generative semantics of [Biernacki et al. \[2020\]](#) for local labels. Each local label ℓ introduced by `local  $\ell : A \rightarrow B$  in  $M$` is replaced by a fresh label generated at runtime. We manage these labels in a context defined as $\Omega ::= \cdot \mid \Omega, \ell : A \rightarrow B$. We do not syntactically distinguish runtime-generated labels from static labels; runtime labels are tracked in Ω . The reduction relation has the form $M \mid \Omega \rightsquigarrow N \mid \Omega'$. Only E-GEN extends Ω , adding a fresh runtime label for a local label binding. Other reduction rules leave Ω unchanged. To prove type soundness, we also extend the other judgements with Ω . For instance, typing judgements have the form $\Omega \mid \Gamma \vdash M : A @ E$ for runtime terms. We omit these modified versions of the rules since the changes are trivial.

Syntax

Terms $M, N ::= \dots \mid \text{local } \ell : A \rightarrow B \text{ in } M$

Well-formedness $\boxed{\Gamma \vdash L} \quad \boxed{\Gamma \vdash D} \quad \boxed{\Gamma \vdash E}$

$$\frac{\ell \in \text{dom}(\Sigma, \Gamma) \quad \Gamma \vdash L}{\Gamma \vdash \ell, L} \quad \frac{\ell \in \text{dom}(\Sigma, \Gamma) \quad \Gamma \vdash D}{\Gamma \vdash \ell, D} \quad \frac{\ell \in \text{dom}(\Sigma, \Gamma) \quad \Gamma \vdash E}{\Gamma \vdash \ell, E}$$

Typing $\boxed{\Gamma \vdash M : A @ E}$

$$\frac{\text{T-LOCAL EFFECT} \quad \Gamma, \ell : A \rightarrow B \vdash M : A' @ E}{\Gamma \vdash \text{local } \ell : A \rightarrow B \text{ in } M : A' @ E}$$

$$\frac{\text{T-Do} \quad \Sigma, \Gamma \ni \ell : A \rightarrow B \quad \Gamma \vdash N : A @ \ell, E}{\Gamma \vdash \text{do } \ell N : B @ \ell, E}$$

$$\frac{\text{T-HANDLE} \quad D = \bar{\ell}_i \quad \Gamma, \ell_i : A_i \rightarrow B_i \vdash M : A @ D, E \quad \Gamma, x : \langle D \rangle A \vdash N : B @ E \quad [\Sigma, \Gamma \ni \ell_i : A_i \rightarrow B_i \quad \Gamma, p_i : A_i, r_i : B_i \rightarrow B \vdash N_i : B @ E]_i}{\Gamma \vdash \text{handle } M \text{ with } \{\text{return } x \mapsto N\} \uplus \{\ell_i p_i r_i \mapsto N_i\}_i : B @ E}$$

Operational semantics $\boxed{M \mid \Omega \rightsquigarrow N \mid \Omega'}$

E-GEN `local  $\ell : A \rightarrow B$  in  $M \mid \Omega \rightsquigarrow M[\ell'/\ell] \mid \Omega, \ell' : A \rightarrow B$` where ℓ' is globally fresh

Figure 5.6: Local labels. New parts in well-formedness and typing rules are highlighted.

5.7 Shallow handlers

Up to now we have considered only *deep* handlers of the form `handle M with H`. Deep handlers automatically wrap the handler around the body of the continuation r captured in a handler clause. We extend the handler syntax with *shallow* handlers `shandle M with H`, in which the handler is not automatically wrapped around the bodies of continuations [Kammar et al. 2013; Hillerström and Lindley 2018]. Figure 5.7 presents the extension with shallow handlers. The most interesting part is that in the typing rule T-SHALLOWHANDLE, the continuation r is wrapped in the modality $\langle D \rangle$, because the continuation is not under the ambient effect context E but may additionally perform effects in D . We do not have this extra modality in the typing rule T-HANDLE in Figure 4.5 for deep handlers, because the continuation of a deep handler is recursively wrapped by the handler itself, which handles effects in D .

Syntax

Terms $M, N ::= \dots \mid \text{shandle } M \text{ with } H$

Typing $\boxed{\Gamma \vdash M : A @ E}$

$$\begin{array}{c} \text{T-SHALLOWHANDLE} \\ \frac{D = \bar{\ell}_i \quad \Gamma, \mu_{\langle D \rangle} \vdash M : A @ D, E \quad \Gamma, x : \langle D \rangle A \vdash N : B @ E \quad [\Sigma \ni \ell_i : A_i \rightarrow B_i \quad \Gamma, p_i : A_i, r_i : \langle D \rangle B_i \rightarrow A \vdash N_i : B @ E]_i}{\Gamma \vdash \text{shandle } M \text{ with } \{\text{return } x \mapsto N\} \uplus \{\ell_i \ p_i \ r_i \mapsto N_i\}_i : B @ E} \end{array}$$

Operational semantics $\boxed{M \rightsquigarrow N}$

Evaluation contexts $\mathcal{E} ::= \dots \mid \text{shandle } \mathcal{E} \text{ with } H$

$$\begin{array}{ll} \text{E-SHALLOWRET} & \text{shandle } V \text{ with } H \rightsquigarrow N[V/x] \\ & \text{where } (\text{return } x \mapsto N) \in H \\ \text{E-SHALLOWOP} & \text{shandle } \mathcal{E}[\text{do } \ell \ V] \text{ with } H \rightsquigarrow N[V/p, (\lambda y. \mathcal{E}[y])/r] \\ & \text{where } 0\text{-free}(\ell, \mathcal{E}) \text{ and } (\ell \ p \ r \mapsto N) \in H \end{array}$$

Figure 5.7: Shallow handlers.

5.8 Modality-parameterised handlers

In the typing rule T-HANDLE in Figure 4.5, the continuation depends on the ambient effect context E of the handled computation M . This is not always desirable. We introduce *modality-parameterised handlers*, a natural generalisation in which effect handlers are parameterised by a modality that is used to wrap continuations.

Figure 5.8 gives the syntax and typing for the extension with modality-parameterised handlers. Rule T-HANDLE now defines a handler parameterised by the modality μ . The

Syntax

Terms $M, N ::= \dots \mid \text{handle}^\mu M \text{ with } H$ Typing $\boxed{\Gamma \vdash M : A @ E}$

$$\begin{array}{c}
\text{T-HANDLE} \\
\frac{
\begin{array}{c}
\mu(F) = E \quad \Gamma \vdash \mu \Rightarrow \langle \rangle @ F \quad \Gamma \vdash \mu \Rightarrow \mu \circ \mu @ F \\
D = \bar{\ell}_i \quad \Gamma, \blacksquare_\mu, \blacksquare_{\langle D \rangle} \vdash M : A @ D, E \quad \Gamma, \blacksquare_\mu, x : \mu \circ \langle D \rangle A \vdash N : B @ E \\
[\Sigma \ni \ell_i : A_i \twoheadrightarrow B_i \quad \Gamma, \blacksquare_\mu, p_i : A_i, r_i : \mu B_i \rightarrow B \vdash N_i : B @ E]_i
\end{array}
}{
\Gamma \vdash \text{handle}^\mu M \text{ with } \{\text{return } x \mapsto N\} \uplus \{\ell_i \ p_i \ r_i \mapsto N_i\}_i : B @ E
}
\end{array}$$

Figure 5.8: Modality-parameterised handlers. New parts in the typing rule are highlighted.

modality μ transforms the effect context F to $\mu(F) = E$ for the whole term as witnessed by the addition of the lock $\blacksquare_\mu$ to the context of each premise. Since both the handled computation and the handler clauses are well-typed under the lock $\blacksquare_\mu$, we can wrap the continuation r , which captures the handled computation and the handler, in the modality μ . The return value x is also wrapped in the modality μ as it is returned from M whose context contains the lock $\blacksquare_{\mu F}$. In contrast, the handler rule in Figure 4.5, shown below, gives r only the function type $B' \rightarrow B$.

$$\frac{
\begin{array}{c}
D = \bar{\ell}_i \quad \Gamma, \blacksquare_{\langle D \rangle} \vdash M : A @ D, E \quad \Gamma, x : \langle D \rangle A \vdash N : B @ E \\
[\Sigma \ni \ell_i : A_i \twoheadrightarrow B_i \quad \Gamma, p_i : A_i, r_i : B_i \rightarrow B \vdash N_i : B @ E]_i
\end{array}
}{
\Gamma \vdash \text{handle } M \text{ with } \{\text{return } x \mapsto N\} \uplus \{\ell_i \ p_i \ r_i \mapsto N_i\}_i : B @ E
}$$

The operational semantics requires no change.

To recover the original handler construct of Tang et al. [2025], we just need to instantiate the modality μ to the identity modality $\langle \rangle$. When the modality annotation is omitted, we take it to be $\langle \rangle$.

Having a modality μ for the continuation r gives us finer-grained control over effect tracking for the continuation. As discussed in Section 7.1.4, the extra expressiveness provided by this rule is especially useful for encoding other effect systems in our unified framework in Chapter 6, as different encodings typically translate function types, including continuation types, using different modalities. We give an example of an effect handler annotated with the empty absolute modality $[]$.

$\text{handle}^[] (\text{do yield } 42; \text{do yield } 37; 0) \text{ with } \{\text{return } x \mapsto x, \text{yield } p \ r \mapsto p + r ()\}$

As a result of the annotation $[]$, the binding for the continuation r has the form $r : [] \ 1 \rightarrow \text{Int}$ instead of $r : 1 \rightarrow \text{Int}$. The annotation $[]$ allows us to use r more flexibly. For example, we can use r inside another absolute modality.

The new handler rule requires the modality μ to have a comonadic structure as specified by the conditions $\mu \Rightarrow \langle \rangle @F$ and $\mu \Rightarrow \mu \circ \mu @F$. These conditions are important because semantically a handler for operation ℓ may not be used (when ℓ is not invoked) or be used multiple times (when ℓ is invoked multiple times). Intuitively, each use of the handler consumes one modality μ . The condition $\mu \Rightarrow \langle \rangle @F$ ensures that when the handler is not used, we can transform away the modality μ at effect context F . The condition $\mu \Rightarrow \mu \circ \mu @F$ ensures that when the handler is used multiple times, we can duplicate the modality μ each time the handler is used. For example, the identity modality $\langle \rangle$ trivially satisfies the comonadic structure, and the absolute modality $[E]$ satisfies the comonadic structure at F with $E \leq F$.

5.9 Parameterised effects

Parameterised effects allow us to define effects that are parameterised by types, such as state effects parameterised by the type of the state. It is standard to extend MET with parameterised effects. Figure 5.9 gives the extension with parameterised effects, only showing the parts that require changes and highlighting the changes in grey. Since parameterised effects require type variables, we assume we have extended contexts with type variables as in Section 5.4.

5 Extensions

Syntax

Effects $D, E, F ::= \dots \mid \ell \ \bar{A}, E$ Label contexts $\Sigma ::= \cdot \mid \Sigma, \ell : \forall \alpha^{\bar{K}}. A \rightarrow B$
 Terms $M, N ::= \dots \mid \text{do } \ell \ \bar{A} \ M$

Well-formedness $\boxed{\Gamma \vdash E} \quad \boxed{\Gamma \vdash \forall \alpha^{\bar{K}}. A \rightarrow B} \quad \boxed{\vdash \Sigma}$

$$\frac{\Sigma \ni \ell : \forall \alpha^{\bar{K}_i}. A \rightarrow B \quad \boxed{\Gamma \vdash A_i : K_i}_i \quad \Gamma \vdash E}{\overline{\Gamma \vdash \cdot} \quad \Gamma \vdash \ell \ \bar{A}_i, E} \quad \frac{\Gamma, \alpha : \bar{K} \vdash A : \text{Abs} \quad \Gamma, \alpha : \bar{K} \vdash B : \text{Abs}}{\Gamma \vdash \forall \alpha^{\bar{K}}. A \rightarrow B} \quad \frac{\vdash \forall \alpha^{\bar{K}}. A \rightarrow B \quad \vdash \Sigma}{\overline{\vdash \cdot} \quad \vdash \ell : \forall \alpha^{\bar{K}}. A \rightarrow B, \Sigma}$$

Equivalence $\boxed{\Gamma \vdash E \equiv F}$

$$\frac{}{\overline{\Gamma \vdash E \equiv E}} \quad \frac{\Gamma \vdash E_1 \equiv E_2 \quad \Gamma \vdash E_2 \equiv E_3}{\Gamma \vdash E_1 \equiv E_3} \quad \frac{[\Gamma \vdash A_i \equiv B_i]_i}{\Gamma \vdash E \equiv E'} \quad \frac{\ell \neq \ell'}{\Gamma \vdash \ell \ \bar{A}_i, \ell' \ \bar{B}_i, E \equiv \ell' \ \bar{B}_i, \ell \ \bar{A}_i, E}$$

Meta-operations on effects $\boxed{E - L} \quad \boxed{L - D}$

$$\begin{aligned} \cdot - L &= \cdot \\ (\ell \ \bar{A}, E) - L &= \begin{cases} E - L' & \text{if } L \equiv \ell, L' \\ \ell \ \bar{A}, (E - L) & \text{otherwise} \end{cases} \end{aligned} \quad \begin{aligned} \cdot - D &= \cdot \\ (\ell, L) - D &= \begin{cases} L - D' & \text{if } D \equiv \ell \ \bar{A}, D' \\ \ell, (L - D) & \text{otherwise} \end{cases} \end{aligned}$$

Modality transformation $\boxed{\Gamma \vdash \mu \Rightarrow \nu @ F}$

$$\begin{aligned} \text{MT-ABS} \quad & \frac{\Gamma \vdash E \leq \mu(F)}{\Gamma \vdash [E] \Rightarrow \mu @ F} & \text{MT-REL} \quad & \frac{}{\Gamma \vdash \langle L | D \rangle \Rightarrow \langle L | D \rangle @ F} & \text{MT-EXPAND} \quad & \frac{\Gamma \vdash (F - L) \equiv \ell \ \bar{A}, E}{\Gamma \vdash \langle L | D \rangle \Rightarrow \langle \ell, L | D, \ell \ \bar{A} \rangle @ F} \\ \text{MT-SHRINK} \quad & \frac{\Gamma \vdash (F - L) \equiv \ell \ \bar{A}, E}{\Gamma \vdash \langle \ell, L | D, \ell \ \bar{A} \rangle \Rightarrow \langle L | D \rangle @ F} & \text{MT-TRANS} \quad & \frac{\Gamma \vdash \mu \Rightarrow \nu @ F \quad \Gamma \vdash \nu \Rightarrow \xi @ F}{\Gamma \vdash \mu \Rightarrow \xi @ F} \end{aligned}$$

Typing $\boxed{\Gamma \vdash M : A @ E}$

T-Do

$$\frac{\Sigma \ni \ell : \forall \alpha^{\bar{K}_i}. A \rightarrow B \quad [\Gamma \vdash A_i : K_i]_i \quad \Gamma \vdash N : A [\bar{A}_i / \alpha_i] @ \ell \ \bar{A}_i, E}{\Gamma \vdash \text{do } \ell \ \bar{A}_i \ N : B [\bar{A}_i / \alpha_i] @ \ell \ \bar{A}_i, E} \quad \text{T-HANDLE} \quad \frac{D = \ell_j \ \bar{A} \quad \Gamma, \mathbf{0}_{\langle D \rangle} \vdash M : A @ D, E \quad \Gamma, x : \langle D \rangle \ A \vdash N : B @ E \quad [\Sigma \ni \ell_j : \forall \alpha^{\bar{K}}. A' \rightarrow B' \quad \Gamma, \alpha : \bar{K}, p : A', r : B' \rightarrow B \vdash N' : B @ E]_j}{\Gamma \vdash \text{handle } M \text{ with } \{\text{return } x \mapsto N\} \uplus \{\ell_j \ p \ r \mapsto N'\}_j : B @ E}$$

Operational semantics $\boxed{M \rightsquigarrow N}$

Evaluation contexts $\mathcal{E} ::= \dots \mid \text{do } \ell \ \bar{A} \ \mathcal{E}$

E-OP $\text{handle } \mathcal{E}[\text{do } \ell \ \bar{A} \ V] \text{ with } H \rightsquigarrow N[\bar{A}/\alpha, V/p, (\lambda y. \text{handle } \mathcal{E}[y] \text{ with } H)/r],$
 where 0-free($\ell, \mathcal{E}$), $(\ell \ p \ r \mapsto N) \in H$, and $\Sigma \ni \ell : \forall \alpha^{\bar{K}}. A' \rightarrow B'$

Figure 5.9: Parameterised effects.

Part II

MET($\mathcal{X}$) the Framework

We propose a parameterised uniform framework $\text{MET}(\mathcal{X})$ based on modal effect types, which enables us to encode different effect systems in the literature.

Chapter 6 introduces the uniform framework $\text{MET}(\mathcal{X})$. Chapter 7 presents encodings of different effect systems into $\text{MET}(\mathcal{X})$.

6 A uniform framework

A family of unimplemented computing languages is described that is intended to span differences of application area by a unified framework.
—P. J. Landin (1966)

The literature includes much work on effect systems for effect handlers based on a range of different theoretical foundations. Two of the most popular and well-studied approaches are row-based effect systems [Hillerström and Lindley 2016; Leijen 2017; Lindley et al. 2017] and capability-based effect systems [Brachthäuser et al. 2020, 2022; Boruch-Gruszecki et al. 2023].

Row-based effect systems, as in the languages Koka [Leijen 2017; Xie et al. 2022], Links [Hillerström and Lindley 2016], and Frank [Lindley et al. 2017], follow the traditional monadic reading of effects: effects are what computations do when they run. They treat effect types as rows of effects and annotate each function arrow with an effect row. For modularity, they implement *parametric effect polymorphism* via row polymorphism [Rémy 1994; Leijen 2005]. For example, a standard application function in System F^ε [Xie et al. 2020], a core calculus of Koka, has type:

$$\forall \varepsilon. (\text{Int} \rightarrow^\varepsilon 1) \rightarrow \text{Int} \rightarrow^\varepsilon 1$$

It is polymorphic in its effects ε , which must agree with the effects performed by its first argument because the argument is invoked in the function body.

Capability-based effect systems, as in the language Effekt [Brachthäuser et al. 2020, 2022] and an experimental extension of Scala 3 [Boruch-Gruszecki et al. 2023], adopt a contextual reading of effects: effects are capabilities provided by the context. Treating effects as capabilities enables a notion of *contextual effect polymorphism* [Brachthäuser et al. 2020] which allows effect-polymorphic reuse of functions without effect variables. For example, an uncurried application function in System C [Brachthäuser et al. 2022], a core calculus of Effekt, has type:

$$(f : \text{Int} \Rightarrow 1, \text{Int}) \Rightarrow 1$$

The argument f is a capability. It is a second-class function that cannot be returned as a value. It can use any capabilities the context provides. We write $\Rightarrow$ for second-class functions. For a curried application function, which requires returning a function, we must capture capabilities in types:

$$(f : \text{Int} \Rightarrow 1) \Rightarrow (\text{Int} \Rightarrow 1 \text{ at } \{f\})$$

Its result has type $(\text{Int} \Rightarrow 1 \text{ at } \{f\})$. As well as specifying argument and result types as usual, this type also includes a *capture set* $\{f\}$ which records that the returned function may use the capability f bound by the argument type $(f : \text{Int} \Rightarrow 1)$.

Though row-based and capability-based effect systems are both well-studied, their relationship is not. We aim to bridge this gap in the literature by encoding both styles of effect systems into a uniform framework. Row-based and capability-based effect systems differ significantly in both their theoretical foundations and their interpretations of effects. Moreover, their mechanisms for tracking effects are *entangled* with other features such as functions. For instance, as we have seen above, a function arrow in **System F^e** describes a standard function type and also provides effect annotations. Similarly, a function arrow in **System C** may bind capabilities. The entanglement of effect tracking with such features is the central challenge in analysing the differences between such effect systems.

Modal effect types provide a new foundation for effect systems. As demonstrated in Part I, **MET** *decouples* effect tracking from standard type and term constructs via modalities. For instance, an application function in **MET** has a plain function type $(\text{Int} \rightarrow 1) \rightarrow \text{Int} \rightarrow 1$. This type imposes no restriction on how effects from the context may be used. To control the use of effects, we can add modalities to the type. For example, the type $[\text{yield}](\text{Int} \rightarrow 1) \rightarrow \text{Int} \rightarrow 1$ restricts the argument function to using only the operation `yield` by wrapping it with the *absolute modality* $[\text{yield}]$; the type $[\Box](\text{Int} \rightarrow 1) \rightarrow [\Box](\text{Int} \rightarrow 1)$ restricts both the argument and result functions to be pure.

We focused on introducing the core theory of **MET** in Part I. In this part, we exploit the observation that the decoupling of effect tracking via modalities leads to a tangible increase in flexibility and expressivity compared to typical effect systems whose effect tracking is entangled with other features. Such decoupling allows us to encode a range of effect systems, including those based on rows and capabilities, in a uniform framework.

In this chapter, we introduce the uniform framework **MET**($\mathcal{X}$), a System F-style core calculus based on **MET** (Chapter 4) with extensions for polymorphism (Section 5.4), local labels (Section 5.6), and modality-parameterised handlers (Section 5.8). **MET**($\mathcal{X}$) is parameterised by an *effect structure* $\mathcal{X}$. An effect structure, inspired by prior work on abstracting row types [Morris and McKinna 2019; Hubers and Morris 2023; Yoshioka

et al. 2024], defines the structure of effect collections. MET hardwires the underlying effect structure to scoped rows [Leijen 2005]. In contrast, $\text{MET}(\mathcal{X})$ allows us to smoothly account for the different treatments of effect collections adopted by different effect systems, such as sets [Bauer and Pretnar 2014; Brachthäuser et al. 2022], simple rows [Morris and McKinnon 2019], and scoped rows [Leijen 2005, 2017; Lindley et al. 2017]. Parameterising by the effect structure enables us to separate the bureaucracy of managing effect collections from our main concern, which is how to use modalities to encode different effect tracking mechanisms. In Chapter 7, we will show how our uniform framework $\text{MET}(\mathcal{X})$ enables encodings of different effect systems.

In the rest of this chapter, we present the syntax, typing rules, and operational semantics of $\text{MET}(\mathcal{X})$. For ease of reference, we give its complete specification, even though many components were introduced separately in Part I.

6.1 Syntax

Figure 6.1 gives the syntax of $\text{MET}(\mathcal{X})$. We highlight syntax relevant to modal effect types and their extensions in grey.

Types	$A, B ::= 1 \mid A \rightarrow B \mid \mu A$ $\mid \alpha \mid \forall \alpha^K. A$	Terms $M, N ::= () \mid x \mid \lambda x^A. M \mid M N$ $\mid \Lambda \alpha^K. V \mid M A \mid \text{mod}_\mu V$ $\mid \text{let}_v \text{mod}_\mu x = V \text{ in } M$ $\mid \text{do } \ell M$ $\mid \text{local } \ell : A \rightarrow B \text{ in } M$ $\mid \text{handle}^\mu M \text{ with } H$
Modalities	$\mu, \nu ::= [E] \mid \langle D \rangle$	Values $V, W ::= () \mid x \mid \lambda x^A. M \mid \text{mod}_\mu V$ $\mid \text{let}_v \text{mod}_\mu x = V \text{ in } W$ $\mid \Lambda \alpha^K. V \mid V A$
Extensions	$D ::= \cdot \mid \ell, D \mid \varepsilon, D$	Handlers $H ::= \{\text{return } x \mapsto N, \ell p r \mapsto M\}$
Effect Contexts	$E, F ::= \cdot \mid \ell, E \mid \varepsilon, E$	
Kinds	$K ::= \text{Abs} \mid \text{Any} \mid \text{Effect}$	
Contexts	$\Gamma ::= \cdot \mid \Gamma, \alpha : K \mid \Gamma, \mu$ $\mid \Gamma, x :_\mu A \mid \Gamma, \ell : A \rightarrow B$	
Label Contexts	$\Sigma ::= \cdot \mid \Sigma, \ell : A \rightarrow B$	

Figure 6.1: Syntax of $\text{MET}(\mathcal{X})$.

As mentioned above, $\text{MET}(\mathcal{X})$ is a System F-style core calculus based on MET (Chapter 4) with three extensions from Chapter 5: polymorphism (Section 5.4), local labels (Section 5.6), and modality-parameterised handlers (Section 5.8). $\text{MET}(\mathcal{X})$ also generalises MET by leaving the structure of effect collections (i.e., extensions and effect contexts) unspecified, as discussed in Section 6.2.

We treat $\text{let}_v \text{mod}_\mu x = V \text{ in } W$ as a value, similar to the fact that we treat $V A$ as a value, which is useful for the encoding of System C in Section 7.3. This is reminiscent of the notion of complex values in call-by-push-value [Levy 2004]. We also extend the

operational semantics of $\text{MET}(\mathcal{X})$ in Section 6.6 to allow reduction of values in $\text{mod}_\mu V$ and $\text{let}_v \text{mod}_\mu x = V$ in M .

We let A range over both value types A and effect contexts E , and let α range over type variables for them in type abstraction $\Lambda \alpha^K. V$ and type application $M A$.

We make several simplifications to MET in order to focus on the core ideas of our encodings of different effect systems. We assume that each handler only handles one operation, and fix a global context Σ which associates each global operation label with its type. An entry $\ell : A \rightarrow B$ indicates that the operation ℓ takes an argument of type A and returns a value of type B . We also support local labels which are introduced by $\text{local } \ell : A \rightarrow B$ in M and maintained in the context Γ . We do not distinguish between local and global labels syntactically.

6.2 Effect structures

An effect structure defines the structure of effect collections, that is, extensions and effect contexts in $\text{MET}(\mathcal{X})$. Extensions D and effect contexts E are both syntactically defined as lists of labels and effect variables. We overload commas for list concatenation, e.g., D, E and E, F both denote list concatenation. As usual, list concatenation is associative but not commutative. The kinding, equivalence, and subtyping (or subeffecting) relations for them are determined by an effect structure $\mathcal{X}$.

Definition 2 (Effect structure). *An effect structure $\mathcal{X}$ is a tuple $\langle \cdot, \equiv \rangle$ of two relations.*

- $\Gamma \vdash D : \text{Effect}$ is a kinding relation which defines well-formed extensions. This relation is preserved by concatenation D, D' . That is, if $\Gamma \vdash D : \text{Effect}$ and $\Gamma \vdash D' : \text{Effect}$, then $\Gamma \vdash D, D' : \text{Effect}$.
- $\Gamma \vdash D \equiv D'$ is an equivalence relation for well-formed extensions.

Our definition of an effect structure $\mathcal{X}$ is minimal and includes only the definitions of kinding and equivalence relations for extensions D . We can naturally derive the kinding relation $\Gamma \vdash E : \text{Effect}$, equivalence relation $\Gamma \vdash E \equiv E'$, and subeffecting relation $\Gamma \vdash E \leq E'$ for effect contexts as follows.

$$\begin{array}{c}
\frac{}{\Gamma \vdash \cdot : \text{Effect}} \quad \frac{\Gamma \ni \varepsilon : \text{Effect}}{\Gamma \vdash \varepsilon : \text{Effect}} \quad \frac{\Gamma \vdash D : \text{Effect} \quad \Gamma \vdash E : \text{Effect}}{\Gamma \vdash D, E : \text{Effect}} \\
\frac{}{\Gamma \vdash \cdot \equiv \cdot} \quad \frac{}{\Gamma \vdash \varepsilon \equiv \varepsilon} \quad \frac{\Gamma \vdash D_1 \equiv D_2 \quad \Gamma \vdash E_1 \equiv E_2}{\Gamma \vdash D_1, E_1 \equiv D_2, E_2} \quad \frac{\Gamma \vdash E, E' \equiv F}{\Gamma \vdash E \leq F}
\end{array}$$

The kinding and equivalence relations for effect contexts are defined inductively. The subeffecting relation is more interesting. We have $E \leq F$ if there exists an effect

context E' such that E, E' is well-formed and $E, E' \equiv F$. It is easy to verify that this subeffecting relation is a partial order. We often write $:\chi$, $\leq_\chi$, and $\equiv_\chi$ to denote which specific effect structure we refer to. We sometimes omit the context Γ for the equivalence and subeffecting relations for brevity.

We give four examples of effect structures, among which $\mathcal{R}_{\text{scp}}$ and $\mathcal{S}$ are used for the encodings of System F^ϵ and System C in Sections 7.2 and 7.3, respectively.

Definition 3 (Simple Rows). $\mathcal{R}_{\text{simp}} = \langle :_{\mathcal{R}_{\text{simp}}}, \equiv_{\mathcal{R}_{\text{simp}}} \rangle$ defines effect collections as simple rows [Morris and McKinna 2019] of operation labels. Well-formed extensions consist of distinct labels without any effect variable. $D \equiv D'$ if D is identical to D' modulo duplication and reordering of labels. The full definition is given in Figure 6.5.

Definition 4 (Scoped Rows). $\mathcal{R}_{\text{scp}} = \langle :_{\mathcal{R}_{\text{scp}}}, \equiv_{\mathcal{R}_{\text{scp}}} \rangle$ defines effect collections as scoped rows [Leijen 2005] of operation labels. Well-formed extensions may contain duplicate labels without any effect variable. $D \equiv D'$ if D is identical to D' modulo reordering of distinct labels. The full definition is given in Figure 6.4.

Definition 5 (Sets). $\mathcal{S} = \langle :_{\mathcal{S}}, \equiv_{\mathcal{S}} \rangle$ defines effect collections as sets. Well-formed extensions are sets of labels and effect variables. The equivalence relation is set equivalence. The full definition is given in Figure 6.2.

Definition 6 (Multisets). $\mathcal{S}_{\text{mult}} = \langle :_{\mathcal{S}_{\text{mult}}}, \equiv_{\mathcal{S}_{\text{mult}}} \rangle$ defines effect collections as multisets. Well-formed extensions are multisets of labels and effect variables. The equivalence relation is multiset equivalence. The full definition is given in Figure 6.3.

$\Gamma \vdash D : \text{Effect}$				
$\frac{}{\Gamma \vdash \cdot : \text{Effect}}$	$\frac{\Sigma, \Gamma \ni \ell : A \multimap B \quad \Gamma \vdash D : \text{Effect}}{\Gamma \vdash \ell, D : \text{Effect}}$	$\frac{\Gamma \vdash \varepsilon : \text{Effect} \quad \Gamma \vdash D : \text{Effect}}{\Gamma \vdash \varepsilon, D : \text{Effect}}$		
$\Gamma \vdash D \equiv D'$				
$\frac{}{D \equiv D}$	$\frac{D_1 \equiv D_2}{D_2 \equiv D_1}$	$\frac{D_1 \equiv D_2 \quad D_2 \equiv D_3}{D_1 \equiv D_3}$	$\frac{D \equiv D'}{\ell, D \equiv \ell, D'}$	$\frac{D \equiv D'}{\varepsilon, D \equiv \varepsilon, D'}$
$\frac{}{\ell, \ell', D \equiv \ell', \ell, D}$	$\frac{}{\ell, \varepsilon, D \equiv \varepsilon, \ell, D}$	$\frac{}{\varepsilon, \varepsilon', D \equiv \varepsilon', \varepsilon, D}$	$\frac{}{\ell, \ell, D \equiv \ell, D}$	
$\frac{}{\varepsilon, \varepsilon, D \equiv \varepsilon, D}$				

Figure 6.2: The effect structure $\mathcal{S}$ (sets).

$$\begin{array}{c}
 \boxed{\Gamma \vdash D : \text{Effect}} \\
 \frac{}{\Gamma \vdash \cdot : \text{Effect}} \quad \frac{\Sigma, \Gamma \ni \ell : A \rightarrow B \quad \Gamma \vdash D : \text{Effect}}{\Gamma \vdash \ell, D : \text{Effect}} \quad \frac{\Gamma \vdash \varepsilon : \text{Effect} \quad \Gamma \vdash D : \text{Effect}}{\Gamma \vdash \varepsilon, D : \text{Effect}} \\
 \\
 \boxed{\Gamma \vdash D \equiv D'} \\
 \frac{}{D \equiv D} \quad \frac{D_1 \equiv D_2}{D_2 \equiv D_1} \quad \frac{D_1 \equiv D_2 \quad D_2 \equiv D_3}{D_1 \equiv D_3} \quad \frac{D \equiv D'}{\ell, D \equiv \ell, D'} \quad \frac{D \equiv D'}{\varepsilon, D \equiv \varepsilon, D'} \\
 \frac{}{\ell, \ell', D \equiv \ell', \ell, D} \quad \frac{}{\ell, \varepsilon, D \equiv \varepsilon, \ell, D} \quad \frac{}{\varepsilon, \varepsilon', D \equiv \varepsilon', \varepsilon, D}
 \end{array}$$

Figure 6.3: The effect structure $\mathcal{S}_{\text{mult}}$ (multisets).

$$\begin{array}{c}
 \boxed{\Gamma \vdash D : \text{Effect}} \\
 \frac{}{\Gamma \vdash \cdot : \text{Effect}} \quad \frac{\Sigma, \Gamma \ni \ell : A \rightarrow B \quad \Gamma \vdash D : \text{Effect}}{\Gamma \vdash \ell, D : \text{Effect}} \\
 \\
 \boxed{\Gamma \vdash D \equiv D'} \\
 \frac{}{D \equiv D} \quad \frac{D_1 \equiv D_2}{D_2 \equiv D_1} \quad \frac{D_1 \equiv D_2 \quad D_2 \equiv D_3}{D_1 \equiv D_3} \quad \frac{D \equiv D'}{\ell, D \equiv \ell, D'} \quad \frac{\ell \neq \ell'}{\ell, \ell', D \equiv \ell', \ell, D}
 \end{array}$$

Figure 6.4: The effect structure $\mathcal{R}_{\text{scp}}$ (scoped rows).

$$\begin{array}{c}
 \boxed{\Gamma \vdash D : \text{Effect}} \\
 \frac{}{\Gamma \vdash \cdot : \text{Effect}} \quad \frac{\Sigma, \Gamma \ni \ell : A \rightarrow B \quad \Gamma \vdash D : \text{Effect}}{\Gamma \vdash \ell, D : \text{Effect}} \\
 \\
 \boxed{\Gamma \vdash D \equiv D'} \\
 \frac{}{D \equiv D} \quad \frac{D_1 \equiv D_2}{D_2 \equiv D_1} \quad \frac{D_1 \equiv D_2 \quad D_2 \equiv D_3}{D_1 \equiv D_3} \quad \frac{D \equiv D'}{\ell, D \equiv \ell, D'} \quad \frac{}{\ell, \ell', D \equiv \ell', \ell, D} \\
 \frac{}{\ell, \ell, D \equiv \ell, D}
 \end{array}$$

Figure 6.5: The effect structure $\mathcal{R}_{\text{simp}}$ (simple rows).

The effect structure $\mathcal{R}_{\text{scp}}$ corresponds to the treatment of effect collections as scoped rows used in **MET**.

Following [Yoshioka et al. \[2024\]](#), an effect structure that intuitively characterises the notion of a collection of effects should satisfy the following validity conditions.

Definition 7 (Validity Conditions). *Validity conditions for an effect structure $\mathcal{X}$ are*

1. if $E \leq_{\mathcal{X}} \cdot$ then $E = \cdot$, and
2. if $\ell \leq_{\mathcal{X}} \ell', E$ and $\ell \neq \ell'$ then $\ell \leq_{\mathcal{X}} E$.

The validity conditions together ensure that if a label ℓ is a subtype of an effect context E , then it must syntactically appear in the effect context E . The first condition prevents us from claiming that some label is contained in the empty effect context. The second condition prevents us from identifying two syntactically different labels as the same one. All effect structures given above satisfy the validity conditions. Our type soundness and effect safety theorems in Section 6.7 are parameterised over any effect structure satisfying the validity conditions.

6.3 Mode theory

The mode theory of $\text{MET}(\mathcal{X})$ follows that of **MET** in Section 4.3. The main difference is that we use a more general modality transformation rule for relative modalities that is suitable for effect structures beyond the scoped rows used in **MET**.

Modes. Modes of $\text{MET}(\mathcal{X})$ remain effect contexts E as in **MET**. Well-formedness, equivalence, and subeffecting of effect contexts are determined by the effect structure $\mathcal{X}$ as in Section 6.2.

Modalities. Well-formedness and equivalence of modalities are unchanged from **MET**.

$$\frac{\Gamma \vdash E : \text{Effect}}{\Gamma \vdash [E]} \quad \frac{\Gamma \vdash D : \text{Effect}}{\Gamma \vdash \langle D \rangle} \quad \frac{\Gamma \vdash E \equiv F}{\Gamma \vdash [E] \equiv [F]} \quad \frac{\Gamma \vdash D \equiv D'}{\Gamma \vdash \langle D \rangle \equiv \langle D' \rangle}$$

Action and composition of modalities are also unchanged from **MET** and are independent of the effect structure $\mathcal{X}$.

$$\begin{array}{lll} [E](F) = E & \langle D \rangle(F) = D, F \\ \mu \circ [E] = [E] & [E] \circ \langle D \rangle = [D, E] & \langle D_1 \rangle \circ \langle D_2 \rangle = \langle D_2, D_1 \rangle \end{array}$$

Modality transformations. We generalise the modality transformation rules of **MET** in Section 4.3.2 to account for different effect structures $\mathcal{X}$.

$$\text{MT-ABS} \frac{\Gamma \vdash E \leq \mu(F)}{\Gamma \vdash [E] \Rightarrow \mu @ F} \quad \text{MT-REL} \frac{\Gamma \vdash D_1, F \leq D_2, F \text{ for all } E \leq F}{\Gamma \vdash \langle D_1 \rangle \Rightarrow \langle D_2 \rangle @ E}$$

Rule MT-ABS is unchanged from **MET**. Rule MT-REL generalises MT-REL of **MET**. It allows us to transform a relative modality $\langle D_1 \rangle$ to another relative modality $\langle D_2 \rangle$ as long as $D_1, F \leq D_2, F$ for every effect context F larger than E . This rule directly corresponds to Lemma 1. This MT-REL rule is suitable for any effect structure, while the corresponding rule MT-REL in Section 4.3.2, which allows $\langle D_1 \rangle \Rightarrow \langle D_2 \rangle @ E$ if D_1 is equivalent to D_2 , is specific to the treatment of effect collections as scoped rows in **MET**. For instance, with the effect structure $\mathcal{S}$, it is safe to have $\langle \ell \rangle \Rightarrow \langle \rangle @ \ell$ because $\ell, \ell \leq \ell$ holds for sets. The label ℓ provided by the modality on the LHS is already in the effect context. In $\mathcal{S}$ these two occurrences of ℓ cannot be distinguished, unlike in scoped rows.

It is worth noting that with the effect structure $\mathcal{R}_{\text{scp}}$, the MT-REL rule of **MET**($\mathcal{R}_{\text{scp}}$) is more permissive than the MT-REL rule of **MET**. As we discussed in Section 4.3.2, the judgement $\langle \rangle \Rightarrow \langle \ell \rangle @ \cdot$ is not allowed in **MET**, but it is allowed in **MET**($\mathcal{R}_{\text{scp}}$). As a result, we still have type soundness for **MET**($\mathcal{R}_{\text{scp}}$) but not abstraction safety. We deliberately choose this more permissive MT-REL rule since it works for other effect structures and it is necessary for encoding other effect systems. Characterising the notion of abstraction safety in the uniform framework **MET**($\mathcal{X}$) is interesting future work.

6.4 Kinds, contexts, and equivalence

The kinding, well-formedness, and equivalence rules largely follow Section 4.4. The full rules are given in Figure 6.6. Kinding and equivalence for extensions and effect contexts are provided by the effect structure $\mathcal{X}$. The rules for type variables and local labels come from Sections 5.4 and 5.6, respectively. The global label context Σ is fixed, and all its operation signatures are closed and well-formed.

As in Section 4.4, $\text{locks}(-)$ composes all the modalities on the locks in a context.

$$\begin{aligned} \text{locks}(\cdot) &= \langle \rangle & \text{locks}(\Gamma, \blacksquare_\mu) &= \text{locks}(\Gamma) \circ \mu & \text{locks}(\Gamma, x :_\mu A) &= \text{locks}(\Gamma) \\ \text{locks}(\Gamma, \alpha : K) &= \text{locks}(\Gamma) & \text{locks}(\Gamma, \ell : A \twoheadrightarrow B) &= \text{locks}(\Gamma) \end{aligned}$$

6.5 Typing

Figure 6.7 gives the full typing rules for **MET**($\mathcal{X}$). They combine the rules for **MET** from Section 4.5 with the extensions for polymorphism, local labels, and modality-parameterised handlers from Sections 5.4, 5.6 and 5.8. As before, we highlight rules

$$\boxed{\Gamma \vdash A : K}$$

$$\frac{\Gamma \vdash A : \text{Abs}}{\Gamma \vdash A : \text{Any}} \quad \frac{\Gamma \ni \alpha : K}{\Gamma \vdash \alpha : K} \quad \frac{}{\Gamma \vdash 1 : \text{Abs}} \quad \frac{\Gamma \vdash [E] \quad \Gamma \vdash A : \text{Any}}{\Gamma \vdash [E]A : \text{Abs}}$$

$$\frac{\Gamma \vdash \langle D \rangle \quad \Gamma \vdash A : K}{\Gamma \vdash \langle D \rangle A : K} \quad \frac{\Gamma \vdash A : \text{Any} \quad \Gamma \vdash B : \text{Any}}{\Gamma \vdash A \rightarrow B : \text{Any}} \quad \frac{\Gamma, \alpha : K_1 \vdash A : K_2}{\Gamma \vdash \forall \alpha^{K_1}. A : K_2}$$

$$\boxed{\Gamma \vdash A \equiv B}$$

$$\frac{}{\Gamma \vdash \alpha \equiv \alpha} \quad \frac{}{\Gamma \vdash 1 \equiv 1} \quad \frac{\Gamma \vdash \mu \equiv \nu \quad \Gamma \vdash A \equiv B}{\Gamma \vdash \mu A \equiv \nu B} \quad \frac{\Gamma \vdash A \equiv A' \quad \Gamma \vdash B \equiv B'}{\Gamma \vdash A \rightarrow B \equiv A' \rightarrow B'}$$

$$\frac{\Gamma, \alpha : K \vdash A \equiv B}{\Gamma \vdash \forall \alpha^K. A \equiv \forall \alpha^K. B}$$

$$\boxed{\Gamma @ E}$$

$$\frac{}{\cdot @ \cdot} \quad \frac{\Gamma @ E \quad \Gamma \vdash \mu \quad \Gamma \vdash A : K}{\Gamma, x :_{\mu} A @ E} \quad \frac{\Gamma @ F \quad \Gamma \vdash \mu \quad \mu(F) = E}{\Gamma, \mathbf{\mu}_{\mu} @ E}$$

$$\frac{\Gamma @ E}{\Gamma, \alpha : K @ E} \quad \frac{\Gamma @ E \quad \Gamma \vdash A \rightarrow B}{\Gamma, \ell : A \rightarrow B @ E}$$

$$\boxed{\Gamma \equiv \Gamma' @ E}$$

$$\frac{\Gamma @ E}{\Gamma \equiv \Gamma @ E} \quad \frac{\Gamma \equiv \Gamma' @ E}{\Gamma' \equiv \Gamma @ E} \quad \frac{\Gamma \equiv \Gamma_1 @ E \quad \Gamma_1 \equiv \Gamma_2 @ E}{\Gamma \equiv \Gamma_2 @ E}$$

$$\frac{\Gamma \equiv \Gamma' @ E \quad \Gamma \vdash \mu \equiv \mu' \quad \Gamma \vdash A \equiv A'}{\Gamma, x :_{\mu} A \equiv \Gamma', x :_{\mu'} A' @ E} \quad \frac{\Gamma \equiv \Gamma' @ F \quad \mu(F) = E \quad \Gamma \vdash \mu \equiv \mu'}{\Gamma, \mathbf{\mu}_{\mu} \equiv \Gamma', \mathbf{\mu}_{\mu'} @ E}$$

$$\frac{\Gamma \equiv \Gamma' @ E}{\Gamma, \alpha : K \equiv \Gamma', \alpha : K @ E} \quad \frac{\Gamma \equiv \Gamma' @ E \quad \Gamma \vdash A \equiv A' \quad \Gamma \vdash B \equiv B'}{\Gamma, \ell : A \rightarrow B \equiv \Gamma', \ell : A' \rightarrow B' @ E} \quad \frac{}{\Gamma, \mathbf{\mu}_{\langle \rangle} \equiv \Gamma @ E}$$

$$\frac{}{\Gamma, \mathbf{\mu}_{\mu}, \mathbf{\mu}_{\nu} \equiv \Gamma, \mathbf{\mu}_{\mu \circ \nu} @ E}$$

$$\boxed{\Gamma \vdash A \rightarrow B} \quad \boxed{\vdash \Sigma}$$

$$\frac{\Gamma \vdash A : \text{Abs} \quad \Gamma \vdash B : \text{Abs}}{\Gamma \vdash A \rightarrow B} \quad \frac{}{\vdash \cdot} \quad \frac{\vdash \Sigma \quad \cdot \vdash A \rightarrow B}{\vdash \Sigma, \ell : A \rightarrow B}$$

Figure 6.6: Kinding, equivalence, and context well-formedness rules for MET($\mathcal{X}$).

relevant to modal effect types in grey. The typing judgement $\Gamma \vdash M : A @ E$ states that the term M has type A under context Γ and effect context E with well-formedness condition $\Gamma @ E$. We require well-formedness of M , A , and E as well-formedness conditions for the typing judgement.

$\Gamma \vdash (\mu, A) \Rightarrow v$	$\frac{\Gamma \vdash A : \text{Abs}}{\Gamma \vdash (\mu, A) \Rightarrow v}$	$\frac{\Gamma @ F \quad \Gamma \vdash \mu \Rightarrow v @ F}{\Gamma \vdash (\mu, A) \Rightarrow v}$
$\Gamma \vdash M : A @ E$	$\frac{\text{T-VAR} \quad \Gamma \vdash (\mu, A) \Rightarrow \text{locks}(\Gamma')}{\Gamma, x :_{\mu} A, \Gamma' \vdash x : A @ E}$ $\frac{\text{T-MOD} \quad \Gamma, \mu \vdash V : A @ \mu(F)}{\Gamma \vdash \text{mod}_{\mu} V : \mu A @ F}$ $\frac{\text{T-LETMOD} \quad \begin{array}{l} \Gamma, \mu \vdash V : \mu A @ v(F) \\ \Gamma, x :_{v \circ \mu} A \vdash M : B @ F \end{array}}{\Gamma \vdash \text{let}_{\nu} \text{mod}_{\mu} x = V \text{ in } M : B @ F}$	$\frac{\text{T-ABS} \quad \Gamma, x : A \vdash M : B @ E}{\Gamma \vdash \lambda x^A. M : A \rightarrow B @ E}$ $\frac{\text{T-APP} \quad \begin{array}{l} \Gamma \vdash M : A \rightarrow B @ E \\ \Gamma \vdash N : A @ E \end{array}}{\Gamma \vdash M N : B @ E}$ $\frac{\text{T-TABS} \quad \Gamma, \alpha : K \vdash V : A @ E}{\Gamma \vdash \Lambda \alpha^K. V : \forall \alpha^K. A @ E}$ $\frac{\text{T-TAPP} \quad \begin{array}{l} \Gamma \vdash M : \forall \alpha^K. A @ E \\ \Gamma \vdash B : K \end{array}}{\Gamma \vdash M B : A[B/\alpha] @ E}$
$\frac{\text{T-UNIT}}{\Gamma \vdash () : 1 @ E}$	$\frac{\text{T-DO} \quad \Sigma, \Gamma \ni \ell : A \rightarrow B \quad \Gamma \vdash N : A @ \ell, E}{\Gamma \vdash \text{do } \ell N : B @ \ell, E}$	$\frac{\text{T-LOCALEFFECT} \quad \Gamma, \ell : A \rightarrow B \vdash M : A' @ E}{\Gamma \vdash \text{local } \ell : A \rightarrow B \text{ in } M : A' @ E}$
$\frac{\text{T-HANDLE} \quad \begin{array}{l} \mu(F) = E \quad \Gamma \vdash \mu \Rightarrow \langle \rangle @ F \quad \Gamma \vdash \mu \Rightarrow \mu \circ \mu @ F \quad \Gamma, \mu, \mu \langle \ell \rangle \vdash M : A @ \ell, E \\ \Sigma, \Gamma \ni \ell : A' \rightarrow B' \quad \Gamma, \mu, x :_{\mu \circ \langle \ell \rangle} A \vdash N : B @ E \quad \Gamma, \mu, p : A', r :_{\mu} B' \rightarrow B \vdash N' : B @ E \end{array}}{\Gamma \vdash \text{handle}^{\mu} M \text{ with } \{\text{return } x \mapsto N, \ell \text{ } p \text{ } r \mapsto N'\} : B @ F}$		

Figure 6.7: Typing rules for $\text{MET}(\mathcal{X})$.

6.6 Operational semantics

Figure 6.8 gives the full operational semantics of $\text{MET}(\mathcal{X})$. It combines the semantics of MET from Section 4.6 with the rules for polymorphism and local labels from Sections 5.4 and 5.6, and adapts the handler rules from Section 5.8. We carry over the runtime label context Ω from Section 5.6 and omit it from reductions when it is unchanged.

We define value normal forms U which cannot reduce further. We always evaluate terms to value normal forms before substituting them into other terms. This is because we include $\text{let}_{\nu} \text{mod}_{\mu} x = V \text{ in } W$ as values. As a result, for example, $\text{let mod}_{\langle \rangle} x = \text{mod}_{\langle \rangle} f \text{ in } ()$ is a value of unit type but captures a function $f : 1 \rightarrow 1$. Since the type 1 has kind Abs , we can bind this value to a variable and use this variable anywhere. However, we want to avoid substituting such a value into a term wrapped in an absolute modality; otherwise the term after substitution would be ill-typed since f would not be

accessible. Requiring values to be in value normal forms before substitution rules out such issues.

The evaluation contexts permit reduction of values under modality introduction and type abstraction, in type application, and in the bound value of modality elimination.

Value normal forms	$U ::= () \mid x \mid \lambda x^A.M \mid \Lambda \alpha^K.U \mid \text{mod}_\mu U$
Evaluation Contexts	$\mathcal{E} ::= [] \mid \mathcal{E} N \mid U \mathcal{E} \mid \mathcal{E} A \mid \text{mod}_\mu \mathcal{E}$ $\mid \Lambda \alpha^K.\mathcal{E} \mid \text{let}_\nu \text{mod}_\mu x = \mathcal{E} \text{ in } M \mid \text{do } \ell \mathcal{E} \mid \text{handle}^\mu \mathcal{E} \text{ with } H$
E-APP	$(\lambda x^A.M) U \rightsquigarrow M[U/x]$
E-TAPP	$(\Lambda \alpha^K.U) A \rightsquigarrow U[A/\alpha]$
E-LETMOD	$\text{let}_\nu \text{mod}_\mu x = \text{mod}_\mu U \text{ in } M \rightsquigarrow M[U/x]$
E-GEN	$\text{local } \ell : A \rightarrow B \text{ in } M \mid \Omega \rightsquigarrow M[\ell'/\ell] \mid \Omega, \ell' : A \rightarrow B \quad \text{where } \ell' \text{ fresh in } \Omega \text{ and } \Sigma$
E-RET	$\text{handle}^\mu U \text{ with } H \rightsquigarrow N[U/x],$ $\text{where } H = \{\text{return } x \mapsto N, \ell \text{ p } r \mapsto N'\}$
E-OP	$\text{handle}^\mu \mathcal{E}[\text{do } \ell U] \text{ with } H \rightsquigarrow N[U/p, (\lambda y.\text{handle}^\mu \mathcal{E}[y] \text{ with } H)/r]$ $\text{where } \ell \notin \text{bl}(\mathcal{E}) \text{ and } H \ni (\ell \text{ p } r \mapsto N)$
E-LIFT	$\mathcal{E}[M] \mid \Omega \rightsquigarrow \mathcal{E}[N] \mid \Omega' \quad \text{if } M \mid \Omega \rightsquigarrow N \mid \Omega'$

Figure 6.8: Operational semantics for $\text{MET}(\mathcal{X})$.

6.7 Metatheory

To state syntactic type soundness, we first define normal forms.

Definition 8 (Normal Forms). *We say that a term M is in normal form with respect to an effect context E if it is either a value normal form $M = U$ or of the form $M = \mathcal{E}[\text{do } \ell U]$ for $\ell \leq E$ and $\ell \notin \text{bl}(\mathcal{E})$.*

The following theorems together give type soundness and effect safety.

Theorem 8 (Progress). *In $\text{MET}(\mathcal{X})$ where $\mathcal{X}$ satisfies the validity conditions, if $\Omega \mid \Gamma \vdash M : A @ E$ and $\text{dom}(\Gamma) = \emptyset$, then either there exist N and Ω' such that $M \mid \Omega \rightsquigarrow N \mid \Omega'$, or M is in a normal form with respect to E .*

Theorem 9 (Subject Reduction). *In $\text{MET}(\mathcal{X})$, if $\Omega \mid \Gamma \vdash M : A @ E$ and $M \mid \Omega \rightsquigarrow N \mid \Omega'$, then $\Omega' \mid \Gamma \vdash N : A @ E$.*

The condition $\text{dom}(\Gamma) = \emptyset$ rules out free term variables while allowing Γ to contain locks. Their proofs are given in Appendices B.3 and B.4. Finally, we can state effect safety.

Theorem 10 (Effect Safety). *In $\text{MET}(X)$ where X satisfies the validity conditions, if $\cdot \vdash M : A @ \cdot$, $M \mid \cdot \rightsquigarrow^* N \mid \Omega'$, and $N \mid \Omega'$ is irreducible, then N is a value normal form, where $\rightsquigarrow^*$ is the reflexive-transitive closure of $\rightsquigarrow$.*

Proof. Theorem 9 and Theorem 8 show that N is in normal form with respect to the empty effect context $\cdot$. Supposing it has the form $\mathcal{E}[\text{do } \ell \ U]$, we have $\ell \leq \cdot$, which contradicts the first validity condition. Therefore, N is a value normal form. $\square$

7 Encodings

The relation between call-by-value and call-by-name is then studied by giving simulations of each language by the other and interpretations of each calculus in the other.

—Gordon D. Plotkin (1975)

We study the relation between different effect systems by encoding, as *macro translations* [Felleisen 1991], various effect systems based on rows and capabilities into our uniform framework $\text{MET}(\mathcal{X})$. We prove that all our encodings preserve typing. Our encodings do not heavily alter the structure of programs but mostly insert terms for manipulating modalities. As a result, it is relatively obvious that the encodings preserve semantics as well; we formally prove this result for some of our encodings. Our primary case studies are encodings of System F^ϵ [Xie et al. 2020], a core calculus of Koka with a row-based effect system, and of System C [Brachthäuser et al. 2022], a core calculus of Effekt with a capability-based effect system. By encoding effect systems into a uniform framework, we can directly reason about the differences between the effect-tracking mechanisms of different effect systems (Sections 7.1.3 and 7.1.4). Our encodings also offer practical insights for language designers (Section 7.5.1).

We start with an overview of our encodings in Section 7.1. Then we present the encodings of row-based effect systems in Section 7.2 and of capability-based effect systems in Section 7.3. We also present the encoding of System $F^{1\epsilon}$, a fragment of System F^ϵ , into modal effect types without any effect variables in Section 7.4. Section 7.5 discusses insights from our encodings and potential extensions.

7.1 Overview of encodings

We briefly describe the row-based effect system of System F^ϵ [Xie et al. 2020] and the capability-based effect system of System C [Brachthäuser et al. 2022] along with their encodings into $\text{MET}(\mathcal{X})$. We use these encodings to directly compare the different systems in a uniform framework. We specifically consider encodings of the different kinds of effect handlers provided by the different systems. We also briefly discuss the results of encoding other effect systems in $\text{MET}(\mathcal{X})$.

7.1.1 Encoding rows

Koka [Leijen 2026] has an effect system based on scoped rows [Leijen 2005]. System F^ϵ [Xie et al. 2020] is a core calculus underlying Koka. To encode System F^ϵ , we use the effect structure $\mathcal{R}_{\text{scp}}$ of scoped rows.

Function types in System F^ϵ have the form $A \rightarrow^E B$, where E is an effect row that specifies the effects that the function may use. Effect types in System F^ϵ are entangled with function types. The key idea of our encoding is to decouple the effect type E from the function arrow via an absolute modality in $\text{MET}(\mathcal{R}_{\text{scp}})$. Writing $\llbracket - \rrbracket$ for translations, we translate a function type as follows.

$$\llbracket A \rightarrow^E B \rrbracket = \llbracket E \rrbracket (\llbracket A \rrbracket \rightarrow \llbracket B \rrbracket)$$

An effectful function in System F^ϵ is decomposed into an absolute modality and a standard function in $\text{MET}(\mathcal{R}_{\text{scp}})$. For instance, consider the following first-order effectful function in System F^ϵ which invokes the operation `yield`.

$$\text{gen}_{F^\epsilon} \doteq \lambda^{\text{yield}} x^{\text{Int}}. \text{do yield } x : \text{Int} \rightarrow^{\text{yield}} 1$$

We use meta-level macros defined by $\doteq$ in red to refer to code snippets. Each λ -abstraction in System F^ϵ is annotated with an effect row, which is the same as the effect row of its function type. We translate gen_{F^ϵ} into $\text{MET}(\mathcal{R}_{\text{scp}})$ as follows.

$$\llbracket \text{gen}_{F^\epsilon} \rrbracket = \text{mod}_{[\text{yield}]} (\lambda x^{\text{Int}}. \text{do yield } x) : [\text{yield}] (\text{Int} \rightarrow 1)$$

At the term level, we insert a modality introduction $\text{mod}_{[\text{yield}]}$ for the λ -abstraction, corresponding to the type-level modality $[\text{yield}]$. We colour mod in grey in the translations. The black parts remain terms with valid syntax and provide intuition about the translation. The modality $[\text{yield}]$ is different from the effect row annotation in System F^ϵ since it is a first-class type constructor and not part of the function type.

As a more involved example including both higher-order functions and function application, consider the effect-polymorphic application function in System F^ϵ :

$$\text{app}_{F^\epsilon} \doteq \Lambda^\epsilon \text{Effect}. \lambda f^{\text{Int} \rightarrow^\epsilon 1}. \lambda^\epsilon x^{\text{Int}}. f \ x : \forall \epsilon. (\text{Int} \rightarrow^\epsilon 1) \rightarrow \text{Int} \rightarrow^\epsilon 1$$

This function abstracts over an effect variable ϵ which stands for the effects performed by the argument f . Both f and the inner λ -abstraction are annotated with ϵ because f is invoked, so their effects must match. The outer λ -abstraction is pure as partial application is pure. The encoding of app_{F^ϵ} in $\text{MET}(\mathcal{R}_{\text{scp}})$ is as follows.

$$\begin{aligned} \llbracket \text{app}_{F^\epsilon} \rrbracket &= \Lambda^\epsilon \text{Effect}. \text{mod}_{[]} (\lambda f^{[\epsilon] (\text{Int} \rightarrow 1)}. \text{mod}_{[\epsilon]} (\lambda x^{\text{Int}}. \text{let mod}_{[\epsilon]} f' = f \text{ in } f' x)) \\ &: \forall \epsilon \text{Effect}. []([\epsilon] (\text{Int} \rightarrow 1) \rightarrow [\epsilon] (\text{Int} \rightarrow 1)) \end{aligned}$$

Each function arrow is associated with an absolute modality reflecting the effects performed by that function. For the pure function arrow in the middle, we use the empty absolute modality $[]$. The type abstraction $\Lambda \varepsilon$ and quantifier $\forall \varepsilon$ are preserved. We omit kinds when obvious. In the term, in addition to modality introduction, we also insert a modality elimination for f before applying it to x . The use of f' requires that the effect variable ε is present in the effect context.

Our term translation from $\text{System } F^\varepsilon$ to $\text{MET}(\mathcal{R}_{\text{scp}})$ explicitly decouples the effect tracking mechanism of $\text{System } F^\varepsilon$ from function abstraction and application. This reveals the essence of effect tracking in $\text{System } F^\varepsilon$. Each λ -abstraction $\lambda^E x. M$ in $\text{System } F^\varepsilon$ is encoded in $\text{MET}(\mathcal{R}_{\text{scp}})$ by inserting a modality introduction $\text{mod}_{[\llbracket E \rrbracket]}$. This demonstrates that a function in $\text{System } F^\varepsilon$ carries its effects. Each function application $V \ W$ in $\text{System } F^\varepsilon$ is encoded by inserting a modality elimination $\text{let } \text{mod}_{[\llbracket E \rrbracket]} f = \llbracket V \rrbracket \text{ in } f \ \llbracket W \rrbracket$ for the function V of type $A \rightarrow^E B$. This demonstrates that when a function is invoked in $\text{System } F^\varepsilon$, we need to provide all effects it may use, as the elimination of $[\llbracket E \rrbracket]$ and use of f together require $\llbracket E \rrbracket$ to be present in the effect context.

We give the full encoding of $\text{System } F^\varepsilon$ into $\text{MET}(\mathcal{R}_{\text{scp}})$ in Section 7.2.2.

7.1.2 Encoding capabilities

Effekt [Brachthäuser et al. 2026] has an effect system based on capabilities. System C [Brachthäuser et al. 2022] is a core calculus underlying Effekt. Since System C tracks capabilities as sets, we use the effect structure $\mathcal{S}$ of sets to encode it.

Functions in System C are called *blocks*. Blocks are second-class in that they must be fully applied and cannot be returned. Capabilities are introduced as block variables. Unlike row-based effect systems which have a separate notion of operation labels, System C interprets effects as capabilities provided by the context. A capability can only be used if it is in scope.

First-order blocks. Let us start with a simple example. Suppose that we have a capability $y : \text{Int} \Rightarrow 1$ (for yielding integers) in the context. We can construct the following block.

$$y :^* \text{Int} \Rightarrow 1 \vdash \text{gen}_C \doteq \{(x : \text{Int}) \Rightarrow y(x)\} : \text{Int} \Rightarrow 1 \mid \{y\}$$

The star $*$ on the binding of y indicates that this block variable is a capability. Braces delimit blocks. Arguments are wrapped in parentheses. Double arrows emphasise that blocks are second-class. The block applies the capability y from the context to the argument x . The typing judgement tracks a capability set $\{y\}$, which contains all

capabilities that the block may use. The block arrow itself has no capability annotation. The above block is simply encoded as a λ -abstraction in $\text{MET}(\mathcal{S})$.¹

$$y^* : \text{Effect}, \hat{y} : [y^*] \text{Int} \rightarrow 1 \vdash \llbracket \text{gen}_C \rrbracket = \lambda x^{\text{Int}}. \hat{y} x : \text{Int} \rightarrow 1 @ y^*$$

The most interesting aspect of the encoding is how we encode the capability $y : \text{Int} \Rightarrow 1$ in the context. A capability y in System C can appear as both a type and a term. We introduce an effect variable y^* of kind **Effect** to represent it at the type level. We omit kinds in the context when obvious. At the term level, we translate the capability directly to $\hat{y} : [y^*] \text{Int} \rightarrow 1$. The absolute modality ensures that whenever $\hat{y}$ is invoked, the effect variable y^* is present in the effect context. The translation of the block body directly applies $\hat{y}$ to x . The effect variable y^* must be in the effect context specified by $@ y^*$ because $\hat{y}$ is used.

Boxes. In System C we can turn a second-class block into a first-class value by boxing it as follows.

$$y : \text{Int} \Rightarrow 1 \vdash \text{gen}'_C \doteq \text{box } \{(x : \text{Int}) \Rightarrow y(x)\} : \text{Int} \Rightarrow 1 \text{ at } \{y\}$$

This typing judgement has no capability set because it types a value, and values are always pure in System C. The value has type $\text{Int} \Rightarrow 1 \text{ at } \{y\}$, which means it is a boxed block of type $\text{Int} \Rightarrow 1$ with capability set $\{y\}$. The block may only use the capability y . We can unbox a boxed block V via $\text{unbox } V$ which gives back a second-class block. We simply encode boxing and unboxing as modality introduction and elimination in $\text{MET}(\mathcal{S})$. For instance, we encode gen'_C as follows.

$$y^*, \hat{y} : [y^*] \text{Int} \rightarrow 1 \vdash \llbracket \text{gen}'_C \rrbracket = \text{mod}_{[y^*]} (\lambda x^{\text{Int}}. \hat{y} x) : [y^*](\text{Int} \rightarrow 1) @ \cdot$$

The capability set annotation $\text{at } \{y\}$ in the type is encoded as the absolute modality $[y^*]$. The encoding shows the connection between boxes of System C and modalities, supporting the claim of Brachthäuser et al. [2022] that boxes of System C are inspired by modal connectives [Choudhury and Krishnaswami 2020].

Higher-order blocks. The situation becomes more involved when we consider higher-order blocks that take other blocks as arguments. This is because System C entangles the introduction and tracking of capabilities with blocks, especially their construction and application.

¹If we strictly follow the encoding of System C in Section 7.3.2, there would be an extra identity modality for the translated function. This modality is crucial for keeping the encoding systematic. We omit such identity modalities in the overview.

Let us consider the uncurried and curried application functions in System C.

$$\begin{aligned} \text{app}_C &\doteq \{(x : \text{Int}, f : \text{Int} \Rightarrow 1) \Rightarrow f(x)\} && : (\text{Int}, f : \text{Int} \Rightarrow 1) \Rightarrow 1 \\ \text{app}'_C &\doteq \{(f : \text{Int} \Rightarrow 1) \Rightarrow \text{box } \{(x : \text{Int}) \Rightarrow f(x)\}\} && : (f : \text{Int} \Rightarrow 1) \Rightarrow (\text{Int} \Rightarrow 1 \text{ at } \{f\}) \end{aligned}$$

These are block constructions. The first block app_C binds the integer parameter x first because System C requires value parameters like x to appear before block parameters like f in a parameter list. In addition to behaving like standard λ -abstractions, block constructions also play an important role in capability tracking. Specifically:

1. Both app_C and app'_C bind a capability $f : \text{Int} \Rightarrow 1$ for their block bodies. This capability f can also be used in the type as shown in the type of app'_C .
2. For soundness, System C assumes that this new capability f is called directly at least once in the block body even if f may actually not be used. (The capability f is indeed called directly in app_C , whereas in app'_C the call occurs inside a box.) Consequently, the capability f is always added to the capability set of the block body tracked by the typing judgement.
3. In addition to the new capability f , both app_C and app'_C allow any capability from the context to be called as well.

Our encoding of block constructions in $\text{MET}(\mathcal{S})$ takes account of these three constraints and exposes them explicitly via modalities. For instance, app_C is encoded as follows.

$$\begin{aligned} \llbracket \text{app}_C \rrbracket &= \Lambda f^*. \text{mod}_{\langle f^* \rangle} (\lambda x^{\text{Int}}. \lambda f^{[f^*](\text{Int} \rightarrow 1)}. \text{let } \text{mod}_{[f^*]} \hat{f} = f \text{ in } \hat{f} x) \\ &: \forall f^*. \langle f^* \rangle (\text{Int} \rightarrow [f^*](\text{Int} \rightarrow 1) \rightarrow 1) \end{aligned}$$

For (1), in order to allow the term variable f to appear in types, we introduce an effect variable f^* and wrap the type $\text{Int} \rightarrow 1$ of the argument f with an absolute modality $[f^*]$. The effect variable f^* represents the term variable f at the level of types. Additionally, we immediately eliminate the modality of f to $\hat{f}$. As a result, in the context of the application $\hat{f} x$ we have three local bindings f^* , f , and $\hat{f}$. For (2) and (3), we use a relative modality $\langle f^* \rangle$ to wrap the whole function type. The relative modality adds the effect variable f^* to the ambient effect context for the function to use, in accordance with (2). The relative modality also allows the function to use effects from the ambient effect context, in accordance with (3).

The translation of app'_C is similar.

$$\begin{aligned} \llbracket \text{app}'_C \rrbracket &= \Lambda f^*. \text{mod}_{\langle f^* \rangle} (\lambda f^{[f^*](\text{Int} \rightarrow 1)}. \text{let } \text{mod}_{[f^*]} \hat{f} = f \text{ in } \text{mod}_{[f^*]} (\lambda x. \hat{f} x)) \\ &: \forall f^*. \langle f^* \rangle ([f^*](\text{Int} \rightarrow 1) \rightarrow [f^*](\text{Int} \rightarrow 1)) \end{aligned}$$

In general, the translation of block types from System C to $\text{MET}(\mathcal{S})$ is as follows, where we let A and B range over value types and let T range over block types.

$$\llbracket (\overline{A}, \overline{f : T}) \Rightarrow B \rrbracket = \forall \overline{f^*}. \langle \overline{f^*} \rangle (\llbracket \overline{A} \rrbracket \rightarrow \llbracket \overline{f^*} \rrbracket \llbracket \overline{T} \rrbracket \rightarrow \llbracket \overline{B} \rrbracket)$$

A block type is decomposed into a standard function type with extra modalities and type quantifiers, which makes explicit exactly how System C introduces and tracks capabilities.

Block calls. Blocks must be fully applied. Assuming we have a capability $y : \text{Int} \Rightarrow 1$ in the context, we can apply the blocks app_C and app'_C to the block gen_C as follows.

$$\begin{aligned} y :^* \text{Int} \Rightarrow 1 \vdash \text{app}_C(42, \text{gen}_C) & : 1 & | \{y\} \\ y :^* \text{Int} \Rightarrow 1 \vdash \text{app}'_C(\text{gen}_C) & : \text{Int} \Rightarrow 1 \text{ at } \{y\} & | \{y\} \end{aligned}$$

As blocks must be fully applied, we must additionally pass an integer to app_C , in this case 42. These two programs are block calls. Like block constructions, block calls in System C not only pass arguments to a block but also play an important role in capability tracking. Specifically:

1. Recall that both app_C and app'_C bind a capability f . Consequently, when calling them with gen_C , System C substitutes f with the capability set $\{y\}$ of gen_C in types. This is reflected by $\text{at } \{y\}$ in the result type of the call to app'_C (before substitution it was at $\{f\}$).
2. Recall that System C assumes the capability f bound by app_C and app'_C is called directly. Consequently, the capability set of the whole block call must be extended with the capability set $\{y\}$ of the argument gen_C . This is reflected by the fact that both typing judgements track the capability sets $\{y\}$ even though the application of app'_C does not call y directly.

Our encoding of block calls in $\text{MET}(S)$ takes account of these two constraints and exposes them explicitly via modalities. For instance, our example application of app_C above is encoded as follows.

$$y^*, \hat{y} : [y^*] \text{Int} \rightarrow 1 \vdash \text{let mod}_{\langle y^* \rangle} f = \llbracket \text{app}_C \rrbracket y^* \text{ in } f \ 42 \ (\text{mod}_{[y^*]} \llbracket \text{gen}_C \rrbracket) : 1 @ y^*$$

For (1), recall that in the translation $\llbracket \text{app}_C \rrbracket$ we bind an effect variable f^* to represent the capability f and wrap the argument type with an absolute modality $[f^*]$. Thus for the application of $\llbracket \text{app}_C \rrbracket$, we instantiate the effect variable f^* with y^* and box the argument $\llbracket \text{gen}_C \rrbracket$ with the absolute modality $[y^*]$. For (2), the elimination of the relative modality $\langle y^* \rangle$ of $\llbracket \text{app}_C \rrbracket y^*$ and the use of f ensure that y^* must be present in the effect context.

The translation of the call of app'_C is similar.

$$y^*, \hat{y} : [y^*] \text{Int} \rightarrow 1 \vdash \text{let mod}_{\langle y^* \rangle} f = \llbracket \text{app}'_C \rrbracket y^* \text{ in } f \ (\text{mod}_{[y^*]} \llbracket \text{gen}_C \rrbracket) : [y^*](\text{Int} \rightarrow 1) @ y^*$$

As with the encoding in Section 7.1.1, the encoding of System C in $\text{MET}(S)$ helps elucidate exactly how the capability tracking in System C is entangled with constructs

like block constructions and calls. Modality introduction and elimination reveal the hidden mechanisms.

We give the full encoding of System C into $\text{MET}(\mathcal{S})$ in Section 7.3.2.

7.1.3 Comparing rows and capabilities

As a uniform framework, $\text{MET}(\mathcal{X})$ allows us to directly compare how effect tracking differs in different effect systems without dealing with the subtleties in their typing and reduction rules.

For instance, let us compare the encoding of function types and polymorphic types in System F^ϵ with the encoding of block types and box types in System C.

$$\begin{aligned}
 \text{System } F^\epsilon \text{ to } \text{MET}(\mathcal{R}_{\text{scp}}) : \quad & \llbracket A \rightarrow^E B \rrbracket = \llbracket E \rrbracket (\llbracket A \rrbracket \rightarrow \llbracket B \rrbracket) \\
 & \llbracket \forall \epsilon. A \rrbracket = \forall \epsilon. \llbracket A \rrbracket \\
 \text{System C to } \text{MET}(\mathcal{S}) : \quad & \llbracket (\overline{A}, \overline{f} : \overline{T}) \Rightarrow B \rrbracket = \forall \overline{f^*}. \langle \overline{f^*} \rangle (\llbracket \overline{A} \rrbracket \rightarrow \llbracket \overline{f^*} \rrbracket \llbracket \overline{T} \rrbracket \rightarrow \llbracket B \rrbracket) \\
 & \llbracket T \text{ at } C \rrbracket = \llbracket C \rrbracket \llbracket T \rrbracket
 \end{aligned}$$

From the encodings we can immediately observe two key differences between System F^ϵ and System C.

1. The encoding of function types in System F^ϵ is wrapped with an absolute modality, whereas the encoding of a block type in System C is wrapped with a relative modality. The encoding of box types in System C is wrapped with an absolute modality. The different modalities reveal a fundamental difference between the meanings of functions in System F^ϵ and blocks in System C: functions in System F^ϵ can only use those effects specified in their types, whereas blocks in System C can use arbitrary effects from the context unless they are boxed.
2. The encoding of block types in System C binds a list of effect variables and wraps each block argument type with an absolute modality of the corresponding effect variable, whereas the encoding of a function type in System F^ϵ is much less involved. Only the encoding of polymorphic types in System F^ϵ binds effect variables. The difference in the treatment of argument types reveals that capabilities in System C act as an implicit form of parametric polymorphism, abstracting the capabilities used by each block variable. This explains why capability-based effect systems do not require explicit effect variables in many cases where row-based effect systems do.

7.1.4 Encoding effect handlers

We have seen how effectful functions in **System F^ε** and **System C** are encoded in $\text{MET}(\mathcal{X})$. These are the most important parts of our encodings, as most effect systems track effects by giving different interpretations to functions. Although all effect systems discussed in this chapter support effect handlers, the same ideas apply equally to traditional effect systems for languages with only built-in effects. Nonetheless, the encodings of effect handlers in **System F^ε** and **System C** are interesting and reveal fundamental differences between the typing and semantics of effect handlers in these two calculi.

In this section, we show how effect handlers in **System F^ε** and **System C** are encoded. We use the following handler in $\text{MET}(\mathcal{X})$ for the `yield` operation as an example.

$$\text{sum}_{\text{MET}(\mathcal{X})} \doteq \text{handle } (\text{do yield } 42; \text{do yield } 37; 0) \text{ with } \{\text{yield } p \ r \mapsto p + r \ ()\}$$

This handler handles the `yield` operation by adding all yielded integers together. We omit the return clause, which is simply $\text{return } x \mapsto x$, for simplicity.

Encoding effect handlers in **System F^ε.** **System F^ε** does not use the `handle` with syntax. Instead, a handler in **System F^ε** is defined as a handler value, which is a function that takes an argument to handle. Consider the following polymorphic handler for the `yield` operation.

$$\text{sum}_{\text{F}^\epsilon} \doteq \Lambda \epsilon. \text{handler } \{\text{yield } p \ r \mapsto p + r \ ()\} : \forall \epsilon. (1 \rightarrow^{\text{yield}, \epsilon} \text{Int}) \rightarrow^\epsilon \text{Int}$$

The term $\text{sum}_{\text{F}^\epsilon}$ is polymorphic over other effects ϵ that it does not handle. The handler syntax defines a handler, which is a function that takes an argument of type $1 \rightarrow^{\text{yield}, \epsilon} \text{Int}$, calls this argument with unit and handles the `yield` operation. The continuation r has type $1 \rightarrow^\epsilon \text{Int}$ as it may use effects abstracted by ϵ . For instance, we can apply $\text{sum}_{\text{F}^\epsilon}$ using the following expression, which reduces to 79.

$$\text{sum}_{\text{F}^\epsilon} E (\lambda x^1. \text{do yield } 42; \text{do yield } 37; 0)$$

We can easily encode $\text{sum}_{\text{F}^\epsilon}$ in $\text{MET}(\mathcal{R}_{\text{scp}})$ as a polymorphic function whose body uses the `handle` with syntax to handle the argument. The main difficulty is that for the handler clause, the continuation r should have type $\llbracket 1 \rightarrow^\epsilon \text{Int} \rrbracket = [\epsilon](1 \rightarrow \text{Int})$ following the translation of function types in Section 7.1.1. However, the naive typing rule of handlers in Section 4.5 only allows us to give a function type to r with no modality. We use the extension of *modality-parameterised handlers* in Section 5.8 to resolve this issue. The handler syntax is annotated with a modality μ as `handleμ M with H`. The continuation r in the handler clause of H now has the plain type $A \rightarrow B$ and carries the modality μ .

on its binder. The translated return clause uses its binder directly. With the modality-parameterised handler, we can translate sum_{F} as follows, omitting the details of the translation of the handler clause, which we name H' .

$$\begin{aligned} \llbracket \text{sum}_{\text{F}} \rrbracket &= \Lambda \varepsilon. \text{mod}_{[\varepsilon]} (\lambda f^{[\text{yield}, \varepsilon]} (1 \rightarrow \text{Int}). \text{handle}^{[\varepsilon]} (\text{let } \text{mod}_{[\text{yield}, \varepsilon]} f' = f \text{ in } f' ()) \text{ with } H') \\ &: \forall \varepsilon. [\varepsilon]([\text{yield}, \varepsilon](1 \rightarrow \text{Int}) \rightarrow \text{Int}) \end{aligned}$$

We eliminate the modality of the argument f before applying and handling f . The type translation follows the translation given in Section 7.1.1. We formally define the translation of handlers in Section 7.2.2.

Encoding effect handlers in System C. System C adopts named handlers. Instead of using operation labels to identify which operation we want to invoke and handle, in System C each handler binds a fresh capability in the scope of the handler and handles the use of this capability. For instance, we can define a named handler and use it to handle a computation as follows.

$$\vdash \text{sum}_{\text{C}} \doteq \text{try } \{y^{\text{Int} \Rightarrow 1} \Rightarrow y(42); y(37); 0\} \text{ with } \{p \ r \mapsto p + r()\} : \text{Int} \mid \text{C}$$

Handlers in System C use the try with syntax. This handler introduces a capability y of type $\text{Int} \Rightarrow 1$ in the scope between try and with. We use the capability y to yield integers 42 and 37. These two uses of y are handled by the handler, whose operation clause is similar to what we have seen before, except that it uses a capability in place of an operation label.

The semantics of named handlers in System C differs from that of the standard effect handlers of Plotkin and Pretnar [2013]. Named handlers have a generative semantics [Biernacki et al. 2020] which dynamically generates a fresh runtime label for each capability introduced by a handler. Dynamic generation guarantees the uniqueness of runtime labels, which ensures that all uses of a capability must be handled by the handler that introduces the capability.

To encode the named handlers of System C into $\text{MET}(\mathcal{X})$, we need to resolve this semantic gap. Adding named handlers to $\text{MET}(\mathcal{X})$ would work but is rather heavyweight. We observe that the essence of named handlers is actually a way to dynamically generate labels. We use the extension of *local labels* from Section 5.6, which decouples dynamic generation of labels from named handlers. The syntax $\text{local } \ell : A \rightarrow B \text{ in } M$ introduces a local label in the scope of M . The type system ensures that the local label ℓ cannot escape from M . The semantics generates a fresh label to replace the local label ℓ . With local labels, we can encode sum_{C} as follows, omitting the handler H' , which contains an operation clause for ℓ_y translated from the handler of sum_{C} .

$$\begin{aligned} \vdash \llbracket \text{sum}_{\text{C}} \rrbracket &= \text{local } \ell_y : \text{Int} \rightarrow 1 \text{ in } \text{handle}^{\llbracket \text{C} \rrbracket} \\ &(\lambda y^{[\ell_y]} (\text{Int} \rightarrow 1). \text{let } \text{mod}_{[\ell_y]} \hat{y} = y \text{ in } \hat{y} 42; \hat{y} 37; 0) (\text{mod}_{[\ell_y]} (\lambda x^{\text{Int}}. \text{do } \ell_y x)) \text{ with } H' : \text{Int} @ \llbracket \text{C} \rrbracket \end{aligned}$$

We introduce a local label ℓ_y for the handler. We use the term $\text{mod}_{[\ell_y]}(\lambda x^{\text{Int}}.\text{do } \ell_y x)$ which invokes the operation ℓ_y to simulate the capability introduced by the named handler in sum_C . The translation of the handled computation binds this function to y , eliminates the modality of y to $\hat{y}$, and uses $\hat{y}$ to yield integers 42 and 37. As in the encoding of effect handlers in System F^ϵ , we also use our modality-parameterised handlers here and annotate `handle` with the modality $[\![C]\!]$.

Our translation $[\![\text{sum}_C]\!]$ is simplified for clarity; it is actually the result of reducing the full translation of sum_C by a few steps. We give the full translation in Section 7.3.2.

Comparing encodings of effect handlers. Our encodings of effect handlers in System F^ϵ and System C elucidate how effect handlers differ in these two effect systems.

1. The System C encoding requires local labels, whereas the System F^ϵ encoding does not, which reveals the syntactic difference that capabilities in System C have scopes whereas operation labels in System F^ϵ do not, and the semantic difference that System C generates fresh runtime labels for effect handlers, whereas System F^ϵ does not.
2. The System F^ϵ encoding performs operations directly, whereas the System C encoding wraps operation invocations into a function (such as the term $\text{mod}_{[\ell_y]}(\lambda x^{\text{Int}}.\text{do } \ell_y x)$ in $[\![\text{sum}_C]\!]$) and passes this function to the handled computation. This difference shows how, in a capability-based effect system such as System C, operations are not directly invoked via their labels but are instead explicitly invoked and passed around as blocks at the term level.

7.1.5 More encodings

The encodings of System F^ϵ and System C illustrate the core idea of using modalities to encode and compare effect systems with different foundations in $\text{MET}(\mathcal{X})$. However, $\text{MET}(\mathcal{X})$ can be used for much more than encoding these two effect systems. In Sections 7.2.4, 7.3.4 and 7.4.2, we give three more representative encodings of effect systems into $\text{MET}(\mathcal{X})$, including

- System Ξ [Brachthäuser et al. 2020], an early core calculus for Effekt based on capabilities,
- System $F^{\epsilon+\text{sn}}$ [Xie et al. 2022], a core calculus formalising scope-safe named handlers of Koka, and
- System $F^{1\epsilon}$, a practical fragment of System F^ϵ that can be encoded into modal effect types without using effect variables.

These results further demonstrate the expressiveness of $\text{MET}(\mathcal{X})$ as a general framework for encoding, comparing, and analysing effect systems. We further discuss practical language design insights arising from our encodings in Section 7.5.1.

7.2 Rows as modal effects

In this section, we first briefly present System F^ϵ [Xie et al. 2020], a System F-style core calculus formalising the row-based effect system of Koka [Leijen 2026], and show how to encode it into $\text{MET}(\mathcal{R}_{\text{scp}})$. Then we present System $F^{\epsilon+\text{sn}}$, a variant of System F^ϵ which formalises named handlers in Koka, and show how to encode it into $\text{MET}(\mathcal{S})$.

7.2.1 System F^ϵ

Figure 7.1 gives the full syntax and typing rules of System F^ϵ . Unlike Xie et al. [2022], our version of System F^ϵ is fine-grain call-by-value [Levy et al. 2003]. Effect rows E are scoped rows [Leijen 2005] with an optional tail effect variable ϵ . As in $\text{MET}(\mathcal{X})$, we assume a fixed global label context Σ . By convention, we write ϵ for effect variables and α for value type variables. We omit their kinds, Effect and Value, when obvious. In type abstractions and applications, we let A range over both value types and effect rows, and let α range over their type variables.

Typing judgements in System F^ϵ include $\Gamma \vdash V : A$ for values and $\Gamma \vdash M : A \mid E$ for computations, where the latter tracks effects E . The typing rules are standard for a System F-style calculus with effect handlers and a row-based effect system [Hillerström and Lindley 2016; Leijen 2017]. The operational semantics is also standard and is given in Figure 7.2.

Rule T-ABS introduces a λ -abstraction. Rule T-DO invokes an operation ℓ . Rule T-HANDLER introduces a handler as a function that takes a function argument of type $1 \rightarrow^{\ell, E} A$ as in Section 7.1.4. Xie et al. [2020] do not include a return clause in handlers for System F^ϵ . Figure 7.2 also defines the runtime handler construct and evaluation contexts. The runtime handler construct is typed as follows.

$$\frac{\text{T-HANDLE} \quad \Sigma \ni \ell : A' \twoheadrightarrow B' \quad \Gamma \vdash M : A \mid \ell, E \quad \Gamma, p : A', r : B' \rightarrow^E A \vdash N : A \mid E}{\Gamma \vdash \text{handle } M \text{ with } \{\ell \ p \ r \mapsto N\} : A \mid E}$$

7 Encodings

Value Types	$A, B ::= 1 \mid \alpha \mid A \rightarrow^E B \mid \forall \alpha^K. A$	Values	$V, W ::= () \mid x \mid \lambda^E x^A. M$
Effect Rows	$E, F ::= \cdot \mid \varepsilon \mid \ell, E$		$\mid \Lambda \alpha^K. V \mid \text{handler } H$
Kind	$K ::= \text{Effect} \mid \text{Value}$	Computations	$M, N ::= \text{return } V \mid V W \mid V A$
Contexts	$\Gamma ::= \cdot \mid \Gamma, x : A \mid \Gamma, \alpha : K$		$\mid \text{do } \ell V \mid \text{let } x = M \text{ in } N$
Label Contexts	$\Sigma ::= \cdot \mid \Sigma, \ell : A \rightarrow B$	Handlers	$H ::= \{\ell \ p \ r \mapsto N\}$

$\boxed{\Gamma \vdash V : A}$

$\frac{}{\Gamma \vdash () : 1}$	$\frac{\text{T-VAR} \quad \Gamma \ni x : A}{\Gamma \vdash x : A}$	$\frac{\text{T-ABS} \quad \Gamma, x : A \vdash M : B \mid E}{\Gamma \vdash \lambda^E x^A. M : A \rightarrow^E B}$	$\frac{\text{T-TABS} \quad \Gamma, \alpha : K \vdash V : A}{\Gamma \vdash \Lambda \alpha^K. V : \forall \alpha^K. A}$
$\frac{\text{T-HANDLER} \quad \Sigma \ni \ell : A' \rightarrow B' \quad \Gamma, p : A', r : B' \rightarrow^E A \vdash N : A \mid E}{\Gamma \vdash \text{handler } \{\ell \ p \ r \mapsto N\} : (1 \rightarrow^{\ell, E} A) \rightarrow^E A}$			

$\boxed{\Gamma \vdash M : A \mid E}$

$\frac{\text{T-VALUE} \quad \Gamma \vdash V : A}{\Gamma \vdash \text{return } V : A \mid E}$	$\frac{\text{T-LET} \quad \Gamma \vdash M : A \mid E \quad \Gamma, x : A \vdash N : B \mid E}{\Gamma \vdash \text{let } x = M \text{ in } N : B \mid E}$
$\frac{\text{T-APP} \quad \Gamma \vdash V : A \rightarrow^E B \quad \Gamma \vdash W : A}{\Gamma \vdash V W : B \mid E}$	$\frac{\text{T-TAPP} \quad \Gamma \vdash V : \forall \alpha^K. B \quad \Gamma \vdash A : K}{\Gamma \vdash V A : B[A/\alpha] \mid E}$
$\frac{\text{T-DO} \quad \Sigma \ni \ell : A \rightarrow B \quad \Gamma \vdash V : A}{\Gamma \vdash \text{do } \ell V : B \mid \ell, E}$	

Figure 7.1: Syntax and typing rules for System F^ϵ .

Computations	$M ::= \dots \mid \text{handle } M \text{ with } H$
Evaluation Contexts	$\mathcal{E} ::= [\] \mid \text{let } x = \mathcal{E} \text{ in } N \mid \text{handle } \mathcal{E} \text{ with } H$
E-TAPP	$(\Lambda \alpha^K. V) T \rightsquigarrow V[T/\alpha]$
E-APP	$(\lambda x^A. M) V \rightsquigarrow M[V/x]$
E-HANDLER	$\text{handler } H \ V \rightsquigarrow \text{handle } V \ () \text{ with } H,$
E-RET	$\text{handle } (\text{return } V) \text{ with } H \rightsquigarrow \text{return } V$
E-OP	$\text{handle } \mathcal{E}[\text{do } \ell V] \text{ with } H \rightsquigarrow N[V/p, (\lambda y. \text{handle } \mathcal{E}[\text{return } y] \text{ with } H)/r],$ where $\ell \notin \text{bl}(\mathcal{E})$ and $H \ni (\ell \ p \ r \mapsto N)$
E-LIFT	$\mathcal{E}[M] \rightsquigarrow \mathcal{E}[N], \quad \text{if } M \rightsquigarrow N$

Figure 7.2: Operational semantics of System F^ϵ .

$$\begin{array}{ll}
\llbracket - \rrbracket : \text{Kind} \rightarrow \text{Kind} & \llbracket - \rrbracket : \text{Type} \rightarrow \text{Type} \\
\llbracket \text{Effect} \rrbracket = \text{Effect} & \llbracket 1 \rrbracket = 1 \\
\llbracket \text{Value} \rrbracket = \text{Abs} & \llbracket \alpha \rrbracket = \alpha \\
\llbracket - \rrbracket : \text{Effect Row} \rightarrow \text{Effect Context} & \llbracket A \rightarrow^E B \rrbracket = \llbracket [E] \rrbracket (\llbracket A \rrbracket \rightarrow \llbracket B \rrbracket) \\
\llbracket \cdot \rrbracket = \cdot & \llbracket \forall \alpha^K. A \rrbracket = \forall \alpha^{\llbracket K \rrbracket}. \llbracket A \rrbracket \\
\llbracket \varepsilon \rrbracket = \varepsilon & \llbracket - \rrbracket : \text{Context} \rightarrow \text{Context} \\
\llbracket \ell, E \rrbracket = \ell, \llbracket E \rrbracket & \llbracket \cdot \rrbracket = \cdot \\
\llbracket - \rrbracket : \text{Label Context} \rightarrow \text{Label Context} & \llbracket \Gamma, x : A \rrbracket = \llbracket \Gamma \rrbracket, x : \llbracket A \rrbracket \\
\llbracket \cdot \rrbracket = \cdot & \llbracket \Gamma, \alpha : K \rrbracket = \llbracket \Gamma \rrbracket, \alpha : \llbracket K \rrbracket \\
\llbracket \Sigma, \ell : A \rightarrow B \rrbracket = \llbracket \Sigma \rrbracket, \ell : \llbracket A \rrbracket \rightarrow \llbracket B \rrbracket & \llbracket \Gamma \rrbracket_E = \llbracket \Gamma \rrbracket, \blacksquare_{\llbracket [E] \rrbracket} \\
\llbracket - \rrbracket : \text{Value / Handler} \rightarrow \text{Term} & \llbracket - \rrbracket : \text{Computation} \rightarrow \text{Term} \\
\llbracket () \rrbracket = () & \llbracket \text{return } V \rrbracket = \llbracket V \rrbracket \\
\llbracket x \rrbracket = x & \llbracket \text{let } x = M \text{ in } N \rrbracket = \text{let } x = \llbracket M \rrbracket \text{ in } \llbracket N \rrbracket \\
\llbracket \Lambda \alpha^K. V \rrbracket = \Lambda \alpha^{\llbracket K \rrbracket}. \llbracket V \rrbracket & \llbracket V A \rrbracket = \llbracket V \rrbracket \llbracket A \rrbracket \\
\llbracket \lambda^E x^A. M \rrbracket = \text{mod}_{\llbracket [E] \rrbracket} (\lambda x^{\llbracket A \rrbracket}. \llbracket M \rrbracket) & \llbracket (V : A \rightarrow^E B) W \rrbracket = \text{let mod}_{\llbracket [E] \rrbracket} x = \llbracket V \rrbracket \text{ in } x \llbracket W \rrbracket \\
\llbracket \text{handler } H : (1 \rightarrow^{\ell, E} A) \rightarrow^E A \rrbracket = \text{mod}_{\llbracket [E] \rrbracket} (\lambda f. \text{handle}^{\llbracket [E] \rrbracket} (\text{let mod}_{\llbracket [E] \rrbracket} f' = f \text{ in } f'()) \text{ with } \llbracket H^E \rrbracket) & \llbracket \text{do } \ell V \rrbracket = \text{do } \ell \llbracket V \rrbracket \\
\llbracket \{ \ell p r \mapsto N \}^E \rrbracket = \{ \text{return } x \mapsto x, \ell p r \mapsto \text{let } r' = \text{mod}_{\llbracket [E] \rrbracket} r \text{ in } \llbracket N \rrbracket[r'/r] \} &
\end{array}$$

Figure 7.3: An encoding of System F^e in $\text{MET}(\mathcal{R}_{\text{scp}})$.

7.2.2 Encoding System F^e into $\text{MET}(\mathcal{R}_{\text{scp}})$

Figure 7.3 encodes System F^e in $\text{MET}(\mathcal{R}_{\text{scp}})$. The translation is mostly straightforward.

For kinds, we translate the effect kind **Effect** to the effect kind **Effect** and the value kind **Value** to the kind **Abs**. We always translate values in System F^e into values of kind **Abs** in $\text{MET}(\mathcal{R}_{\text{scp}})$.

For types, we translate an effectful function type $A \rightarrow^E B$ into a function type with an absolute modality $\llbracket [E] \rrbracket (\llbracket A \rrbracket \rightarrow \llbracket B \rrbracket)$.

For contexts, we homomorphically translate each entry. For a typing judgement $\Gamma \vdash M : A \mid E$ in System F^e , we pass the effect context E to the translation of its context Γ as $\llbracket \Gamma \rrbracket_E$, which adds a lock $\blacksquare_{\llbracket [E] \rrbracket}$ to the translated context. This lock is used to maintain the invariant of $\text{MET}(X)$ typing judgements that $\llbracket \Gamma \rrbracket_E @ \llbracket [E] \rrbracket$.

For terms, the translation is type-directed and essentially defined on typing judgements. We annotate components of a term with their types as necessary. We highlight modality-relevant syntax of the term translation in *grey*. The grey parts show how modalities decouple effect tracking. The black parts themselves remain valid programs after type erasure.

The translation of lambda abstraction $\lambda^E x^A. M$ introduces an absolute modality by $\text{mod}_{\llbracket [E] \rrbracket}$, and the translation of function application $V W$ first eliminates the modality of $\llbracket V \rrbracket$ by $\text{let mod}_{\llbracket [E] \rrbracket} x = \llbracket V \rrbracket$ before applying it. Examples of translations of lambda abstraction and application can be found in Section 7.1.1 as $\llbracket \text{gen}_{F^e} \rrbracket$ and $\llbracket \text{app}_{F^e} \rrbracket$.

Translations of type abstraction, type application, operation invocation, and let-binding are homomorphic. Let-binding in $\text{MET}(X)$ is syntactic sugar defined in the standard way as $\text{let } x = M \text{ in } N \doteq (\lambda x. N) M$. The translation of $\text{return } V$ is simply $\llbracket V \rrbracket$.

A handler value handler H of type $(1 \rightarrow^{\ell, E} A) \rightarrow^E A$ is translated to a higher-order function that handles the application of its function argument f . We eliminate the modality of f before applying it to $()$ since f has type $\llbracket \ell, E \rrbracket (1 \rightarrow \llbracket A \rrbracket)$. We introduce a modality $\text{mod}_{\llbracket E \rrbracket}$ for the whole translated function since handler H is an effectful function with effect E . In the return clause of $\llbracket H \rrbracket$, the modality $\llbracket \ell, E \rrbracket$ appears on the binder of x , so no modality elimination is needed. The operation clause of $\llbracket H \rrbracket$ demonstrates why we must use modality-parameterised handlers. Note that rule T-HANDLER of System F^ϵ gives the continuation r in H the type $B' \rightarrow^E A$. By annotating handle with $\llbracket E \rrbracket$, the modality $\llbracket E \rrbracket$ appears on the binder of the continuation r in $\llbracket H \rrbracket$, while r has type $\llbracket B' \rrbracket \rightarrow \llbracket A \rrbracket$. We therefore rebox r and bind the result to r' before translating N , so that r' has type $\llbracket E \rrbracket (\llbracket B' \rrbracket \rightarrow \llbracket A \rrbracket)$, which is equal to $\llbracket B' \rightarrow^E A \rrbracket$. The runtime handler construct is translated as follows. It is used in the proof of semantics preservation in Appendix C.1.

$$\begin{aligned} \llbracket - \rrbracket &: \text{Computation} \rightarrow \text{Term} \\ \llbracket \text{handle } M \text{ with } H : A \mid E \rrbracket &= \text{handle}^{\llbracket E \rrbracket} \llbracket M \rrbracket \text{ with } \llbracket H \rrbracket \end{aligned}$$

Translations of evaluation contexts are analogous to the translations of their corresponding terms.

We now give the full translation of the handler sum_{F^ϵ} of Section 7.1.4.

$$\begin{aligned} \llbracket \text{sum}_{F^\epsilon} \rrbracket &= \Lambda \varepsilon. \text{mod}_{\llbracket \varepsilon \rrbracket} (\lambda f^{\llbracket \text{yield}, \varepsilon \rrbracket (1 \rightarrow \text{Int})}. \text{handle}^{\llbracket \varepsilon \rrbracket} (\text{let } \text{mod}_{\llbracket \text{yield}, \varepsilon \rrbracket} f' = f \text{ in } f' ()) \text{ with} \\ &\quad \{\text{return } x \mapsto x, \ell \ p \ r \mapsto \text{let } r' = \text{mod}_{\llbracket \varepsilon \rrbracket} r \text{ in } p + (\text{let } \text{mod}_{\llbracket \varepsilon \rrbracket} r'' = r' \text{ in } r'' ())\}) \\ &: \forall \varepsilon. \llbracket \varepsilon \rrbracket (\llbracket \text{yield}, \varepsilon \rrbracket (1 \rightarrow \text{Int}) \rightarrow \text{Int}) \end{aligned}$$

We have the following type and semantics preservation theorems with proofs in Appendix C.1.

Theorem 11 (Type Preservation). *If $\Gamma \vdash M : A \mid E$ in System F^ϵ , then $\llbracket \Gamma \rrbracket_E \vdash \llbracket M \rrbracket : \llbracket A \rrbracket @ \llbracket E \rrbracket$ in $\text{MET}(\mathcal{R}_{\text{scp}})$. Similarly for typing judgements of values.*

Theorem 12 (Semantics Preservation). *If M is well-typed and $M \rightsquigarrow N$ in System F^ϵ , then $\llbracket M \rrbracket \rightsquigarrow^* \llbracket N \rrbracket$ in $\text{MET}(\mathcal{R}_{\text{scp}})$ where $\rightsquigarrow^*$ denotes the reflexive-transitive closure of $\rightsquigarrow$.*

7.2.3 System $F^{\epsilon+\text{sn}}$

Xie et al. [2022] extend Koka with named handlers and formalise this extension in the core calculus System $F^{\epsilon+\text{sn}}$, which is based on System F^ϵ . System $F^{\epsilon+\text{sn}}$ allows each

handler to bind a handler name that can be used to invoke operations. A handler name is similar to a capability in System C, but it is a first-class value. For instance, we can define a named handler in System $F^{\epsilon+sn}$ as follows.

$$sum_{F^{\epsilon+sn}} \doteq \Lambda \epsilon. nhandler \{ yield \ p \ r \mapsto p + r \ () \} : \forall \epsilon. (\forall a. ev \ yield^a \rightarrow^{yield^a, \epsilon} Int) \rightarrow^{\epsilon} Int$$

This handler is similar to the handler $sum_{F^{\epsilon}}$ in Section 7.1.4. The main difference is that the argument takes a value of type $ev \ yield^a$. This is a first-class handler name with which we can invoke the yield operation. For example, we can apply $sum_{System \ F^{\epsilon+sn}}$ as follows.

$$sum_{F^{\epsilon+sn}} E (\Lambda a. \lambda h^{ev \ yield^a}. h \ 42; h \ 37; 0)$$

Instead of using the label `yield` to invoke the operation as in the application of $sum_{F^{\epsilon}}$ in Section 7.1.4, we directly apply the handler name h to arguments. This is reminiscent of the handler $sum_{System \ C}$ in Section 7.1.4 where we invoke the operation by calling the capability introduced by the handler. This program reduces to 79. The scope variable a ensures scope safety of the handler name, similar to the technique used by `runST` in Haskell [Launchbury and Peyton Jones 1995]. We give the full syntax, typing rules, and operational semantics of System $F^{\epsilon+sn}$ below.

Figure 7.4 gives the syntax of System $F^{\epsilon+sn}$. Our presentation of System $F^{\epsilon+sn}$ is fine-grain call-by-value. We highlight new parts compared to System F^{ϵ} . Each effect label ℓ is annotated with a scope variable a . As before, we omit kinds when obvious from the variable names.

Value Types	$A, B ::= 1 \mid \alpha \mid A \rightarrow^E B \mid \forall \alpha^K. A \mid \text{ev } \ell^a$
Kind	$K ::= \text{Value} \mid \text{Effect} \mid \text{Scope}(\ell)$
Effect Rows	$E ::= \cdot \mid \epsilon \mid \ell^a, E$
Contexts	$\Gamma ::= \cdot \mid \Gamma, x : A \mid \Gamma, \alpha : K$
Values	$V, W ::= () \mid x \mid \lambda^E x^A. M \mid \Lambda \alpha^K. V \mid \text{nhandler } H$
Computations	$M, N ::= \text{return } V \mid V \ W \mid V \ A \mid \text{let } x = M \text{ in } N \mid \text{do } V \ W$
Handlers	$H ::= \{ \ell \ p \ r \mapsto M \}$
Label Contexts	$\Sigma ::= \cdot \mid \Sigma, \ell : A \rightarrow B$

Figure 7.4: Syntax of System $F^{\epsilon+sn}$.

Unlike Xie et al. [2022], the kind $\text{Scope}(\ell)$ of scope variables is annotated with an effect label ℓ . For $a : \text{Scope}(\ell)$, each appearance of a in effect rows must be associated with this label ℓ as ℓ^a . This annotation is important to rule out well-typed but meaningless terms in System $F^{\epsilon+sn}$. Not every well-typed term in System $F^{\epsilon+sn}$ is meaningful in the sense that we can find an appropriate evaluation context to fully apply its abstractions and handle its effects. For example, a function of type $\forall a. ev \ \ell_1^a \times ev \ \ell_2^a \rightarrow^{\ell_1^a, \ell_2^a} 1$ cannot be applied and handled when $\ell_1 \neq \ell_2$, because named handlers cannot provide two

evidence values of types $\text{ev } \ell_1^a$ and $\text{ev } \ell_2^a$ with the same scope variable a but different operations ℓ_1 and ℓ_2 . (This type becomes meaningful with umbrella effects in Xie et al. [2022].) Each handler introduces its own scope variable a with some fixed operation label ℓ . Annotating the kind Scope with an operation label ℓ solves the problem of attaching the same scope variable a to different operation labels.

Figure 7.5 gives the typing rules. Rule T-NAMEDHANDLER introduces a named handler as a function, whose argument takes a handler name of the evidence type $\text{ev } \ell^a$. An evidence type specifies an effect ℓ and a scope variable a . The use of rank-2 polymorphism guarantees that the handler name of type $\text{ev } \ell^a$ cannot escape the scope of the handler. Rule T-DOName invokes an operation via a handler name M of the evidence type $\text{ev } \ell^a$.

$\Gamma \vdash V : A$			
T-UNIT $\frac{}{\Gamma \vdash () : 1}$	T-VAR $\frac{\Gamma \ni x : A}{\Gamma \vdash x : A}$	T-ABS $\frac{\Gamma, x : A \vdash M : B \mid E}{\Gamma \vdash \lambda^E x^A. M : A \rightarrow^E B}$	T-TABS $\frac{\Gamma, \alpha : K \vdash V : A}{\Gamma \vdash \Lambda \alpha^K. V : \forall \alpha^K. A}$
T-NAMEDHANDLER $\frac{\Sigma \ni \ell : A' \rightarrow B' \quad \Gamma, p : A', r : B' \rightarrow^E A \vdash N : A \mid E}{\Gamma \vdash \text{nhandler } \{\ell \ p \ r \mapsto N\} : (\forall a^{\text{Scope}(\ell)}. \text{ev } \ell^a \rightarrow^{\ell^a, E} A) \rightarrow^E A}$			
$\Gamma \vdash M : A \mid E$			
T-VALUE $\frac{\Gamma \vdash V : A}{\Gamma \vdash \text{return } V : A \mid E}$	T-APP $\frac{\Gamma \vdash V : A \rightarrow^E B \quad \Gamma \vdash W : A}{\Gamma \vdash V W : B \mid E}$	T-TAPP $\frac{\Gamma \vdash V : \forall \alpha^K. B \quad \Gamma \vdash A : K}{\Gamma \vdash V A : B[A/\alpha] \mid E}$	
T-LET $\frac{\Gamma \vdash M : A \mid E \quad \Gamma, x : A \vdash N : B \mid E}{\Gamma \vdash \text{let } x = M \text{ in } N : B \mid E}$		T-DOName $\frac{\Sigma \ni \ell : A \rightarrow B \quad \Gamma \vdash V : \text{ev } \ell^a \quad \Gamma \vdash W : A}{\Gamma \vdash \text{do } V W : B \mid \ell^a, E}$	

Figure 7.5: Typing rules for System F^{e+sn} .

Figure 7.6 gives the operational semantics of System F^{e+sn} , including the definitions of runtime constructs and evaluation contexts. As in the operational semantics of $\text{MET}(\mathcal{X})$ and System C, reduction rules in System F^{e+sn} are also of the form $M \mid \Omega \rightsquigarrow N \mid \Omega'$. The most interesting rule is E-GEN, which reduces a handler application to a runtime handle construct and passes a runtime evidence value ev_h to the argument. This rule generates a fresh marker h and a fresh scope variable a . The runtime context Ω associates each dynamically generated marker h with its operation label ℓ and its dynamically generated scope variable a .

Unlike Xie et al. [2022], our E-NOP rule for System F^{e+sn} has the condition $h \notin \text{BH}(\mathcal{E})$, which ensures that there is no other handler in $\mathcal{E}$ with the marker h . This condition

is necessary to guarantee that the current handler is the nearest one for the marker h , because even for named handlers it is still possible to duplicate the same handler at runtime, as observed by [Biernacki et al. \[2020\]](#).

Runtime Markers	h
Runtime Contexts	$\Omega ::= \cdot \mid \Omega, h : \ell^a$
Computations	$M ::= \dots \mid \text{handle}_h M \text{ with } H$
Values	$V ::= \dots \mid \text{ev}_h$
Evaluation Contexts	$\mathcal{E} ::= [\] \mid \text{let } x = \mathcal{E} \text{ in } N \mid \text{handle}_h \mathcal{E} \text{ with } H$
E-TAPP	$(\Lambda \alpha^K.V) A \rightsquigarrow V[A/\alpha]$
E-APP	$(\lambda x^A.M) V \rightsquigarrow M[V/x]$
E-GEN	$\text{nhandler } H \ V \mid \Omega \rightsquigarrow \text{handle}_h (\text{let } x = V \ a \text{ in } x \ \text{ev}_h) \text{ with } H \mid \Omega, h : \ell^a$ where a, h fresh and $H \ni (\ell \ p \ r \mapsto N)$
E-NRET	$\text{handle}_h (\text{return } V) \text{ with } H \rightsquigarrow \text{return } V$
E-NOF	$\text{handle}_h \mathcal{E}[\text{do ev}_h V] \text{ with } H \rightsquigarrow N[V/p, (\lambda y. \text{handle}_h \mathcal{E}[\text{return } y] \text{ with } H)/r],$ where $h \notin \text{BH}(\mathcal{E})$ and $H \ni (\ell \ p \ r \mapsto N)$
E-LIFT	$\mathcal{E}[M] \mid \Omega \rightsquigarrow \mathcal{E}[N] \mid \Omega', \text{ if } M \mid \Omega \rightsquigarrow N \mid \Omega'$

Figure 7.6: Operational semantics for System F^{e+sn} .

The meta-function $\text{BH}(\mathcal{E})$ gives the set of runtime markers bound by handlers in $\mathcal{E}$:

$$\begin{aligned} \text{BH}([\]) &= \emptyset \\ \text{BH}(\text{let } x = \mathcal{E} \text{ in } N) &= \text{BH}(\mathcal{E}) \\ \text{BH}(\text{handle}_h \mathcal{E} \text{ with } H) &= \text{BH}(\mathcal{E}) \cup \{h\}. \end{aligned}$$

Typing rules for runtime constructs are as follows.

$$\begin{array}{c} \text{T-EVIDENCE} \\ \frac{\Omega \ni h : \ell^a}{\Omega \mid \Gamma \vdash \text{ev}_h : \text{ev } \ell^a} \end{array} \quad \begin{array}{c} \text{T-HANDLENAME} \\ \frac{\Omega \ni h : \ell^a \quad \Sigma \ni \ell : A' \twoheadrightarrow B' \quad \Omega \mid \Gamma, p : A', r : B' \rightarrow^E A \vdash N : A \mid E}{\Omega \mid \Gamma \vdash \text{handle}_h M \text{ with } \{\ell \ p \ r \mapsto N\} : A \mid E} \end{array}$$

7.2.4 Encoding System F^{e+sn} into $\text{MET}(\mathcal{S})$

As with the encoding of named handlers in System $\mathcal{C}$, we can encode a named handler of System F^{e+sn} by introducing a local label ℓ_a and using the term $\text{mod}_{[\ell_a]} (\lambda x. \text{do } \ell_a \ x)$ to simulate the handler name. We use the effect structure $\mathcal{S}$ instead of $\mathcal{R}_{\text{scp}}$ because handler names are unique. The theory $\mathcal{S}$ allows multiple effect variables, which we use to encode scope variables. We give the full encoding below and prove its type and semantics preservation in Appendix C.4.

Figure 7.7 gives the translation of System F^{e+sn} into $\text{MET}(\mathcal{S})$. The evidence type $\text{ev } \ell^a$ with $\Sigma \ni \ell : A \twoheadrightarrow B$ is translated to a function type $[a](\llbracket A \rrbracket \rightarrow \llbracket B \rrbracket)$. Correspondingly,

an operation invocation $\text{do } V \ W$ for $V : \text{ev } \ell^a$ is translated in the same way as a function application. A named handler $\text{nhandler } H$ is translated to a function that takes a function argument f and handles it with a handler in $\text{MET}(S)$. We pass the term $\text{mod}_{[a]} (\lambda x. \text{do}_a x)$ to the argument f to simulate the handler name of type $\text{ev } \ell^a$ in System $F^{\epsilon+\text{sn}}$.

$$\begin{array}{ll}
\llbracket - \rrbracket : \text{Kind} \rightarrow \text{Kind} & \llbracket - \rrbracket : \text{Effect Row} \rightarrow \text{Effect Context} \\
\llbracket \text{Value} \rrbracket = \text{Abs} & \llbracket \cdot \rrbracket = \cdot \\
\llbracket \text{Effect} \rrbracket = \text{Effect} & \llbracket \varepsilon \rrbracket = \varepsilon \\
\llbracket \text{Scope}(\ell) \rrbracket = \text{Effect} & \llbracket \ell^a, E \rrbracket = a, \llbracket E \rrbracket \\
\\
\llbracket - \rrbracket : \text{Type} \rightarrow \text{Type} & \llbracket - \rrbracket : \text{Context} \rightarrow \text{Context} \\
\llbracket 1 \rrbracket = 1 & \llbracket \cdot \rrbracket = \cdot \\
\llbracket \alpha \rrbracket = \alpha & \llbracket \Gamma, x : A \rrbracket = \llbracket \Gamma \rrbracket, x : \llbracket A \rrbracket \\
\llbracket A \rightarrow^E B \rrbracket = \llbracket \llbracket E \rrbracket \rrbracket (\llbracket A \rrbracket \rightarrow \llbracket B \rrbracket) & \llbracket \Gamma, \alpha : K \rrbracket = \llbracket \Gamma \rrbracket, \alpha : \llbracket K \rrbracket \\
\llbracket \forall \alpha^K. A \rrbracket = \forall \alpha^{\llbracket K \rrbracket}. \llbracket A \rrbracket & \llbracket \Gamma \rrbracket_E = \llbracket \Gamma \rrbracket, \blacksquare_{\llbracket \llbracket E \rrbracket \rrbracket} \\
\llbracket \text{ev } \ell^a \rrbracket = \llbracket a \rrbracket (\llbracket A \rrbracket \rightarrow \llbracket B \rrbracket) & \llbracket - \rrbracket : \text{Label Context} \rightarrow \text{Label Context} \\
\text{where } \Sigma \ni \ell : A \twoheadrightarrow B & \llbracket \cdot \rrbracket = \cdot \\
& \llbracket \Sigma, \ell : A \twoheadrightarrow B \rrbracket = \llbracket \Sigma \rrbracket, \ell : \llbracket A \rrbracket \twoheadrightarrow \llbracket B \rrbracket \\
\\
\llbracket - \rrbracket : \text{Value / Computation} \rightarrow \text{Term} \\
\llbracket () \rrbracket = () \\
\llbracket x \rrbracket = x \\
\llbracket \Lambda \alpha^K. V \rrbracket = \Lambda \alpha^{\llbracket K \rrbracket}. \llbracket V \rrbracket \\
\llbracket \text{return } V \rrbracket = \llbracket V \rrbracket \\
\llbracket \text{let } x = M \text{ in } N \rrbracket = \text{let } x = \llbracket M \rrbracket \text{ in } \llbracket N \rrbracket \\
\llbracket V \ A \rrbracket = \llbracket V \rrbracket \ \llbracket A \rrbracket \\
\llbracket V^{A \rightarrow^E B} \ W \rrbracket = \text{let mod}_{\llbracket \llbracket E \rrbracket \rrbracket} x = \llbracket V \rrbracket \text{ in } x \ \llbracket W \rrbracket \\
\llbracket \text{do } V^{\text{ev } \ell^a} \ W \rrbracket = \text{let mod}_{[a]} x = \llbracket V \rrbracket \text{ in } x \ \llbracket W \rrbracket \\
\llbracket \text{nhandler } H \rrbracket = \text{mod}_{\llbracket \llbracket E \rrbracket \rrbracket} (\lambda f. \text{local } \ell_a : \llbracket A' \rrbracket \twoheadrightarrow \llbracket B' \rrbracket \text{ in handle}^{\llbracket \llbracket E \rrbracket \rrbracket} \\
\llbracket (\forall a^\ell. \text{ev } \ell^a \rightarrow^{\ell^a, E} A) \rightarrow^E A \rrbracket = \text{mod}_{\llbracket \llbracket E \rrbracket \rrbracket} (\lambda f. \text{local } \ell_a : \llbracket A' \rrbracket \twoheadrightarrow \llbracket B' \rrbracket \text{ in handle}^{\llbracket \llbracket E \rrbracket \rrbracket} \\
\quad (\text{let } x = f \ \ell_a \text{ in} \\
\quad \quad \text{let mod}_{[\ell_a, \llbracket \llbracket E \rrbracket \rrbracket]} f' = x \text{ in} \\
\quad \quad f' (\text{mod}_{[\ell_a]} (\lambda x^{\llbracket A' \rrbracket}. \text{do } \ell_a \ x))) \\
\quad \text{with } \llbracket H^{a, E} \rrbracket) \\
\llbracket H^{a, E} \rrbracket = \{ \text{return } x \mapsto x, \\
\quad \ell_a \ p \ r \mapsto \text{let } r' = \text{mod}_{\llbracket \llbracket E \rrbracket \rrbracket} r \text{ in } \llbracket N \rrbracket[r'/r] \}
\end{array}$$

Figure 7.7: An encoding of System $F^{\epsilon+\text{sn}}$ in $\text{MET}(S)$.

The following theorems show that our encoding preserves types and operational semantics.

Theorem 13 (Type Preservation). *If $\Gamma \vdash M : A \mid E$ in System $F^{\epsilon+\text{sn}}$, then $\llbracket \Gamma \rrbracket_E \vdash \llbracket M \rrbracket : \llbracket A \rrbracket @ \llbracket \llbracket E \rrbracket \rrbracket$ in $\text{MET}(S)$. Similarly for typing judgements of values.*

Theorem 14 (Semantics Preservation). *Suppose no two entries of Ω contain the same scope variable. If M is well-typed and $M \mid \Omega \rightsquigarrow N \mid \Omega'$ in **System** $F^{\epsilon+sn}$, then $\llbracket M \rrbracket \mid \llbracket \Omega \rrbracket \rightsquigarrow^* \llbracket N \rrbracket \mid \llbracket \Omega' \rrbracket$ in $\text{MET}(\mathcal{S})$, where $\rightsquigarrow^*$ denotes the reflexive-transitive closure of $\rightsquigarrow$.*

Translations of runtime constructs for **System** $F^{\epsilon+sn}$ are as follows. Note that we translate dynamically generated scope variables a to their corresponding dynamically generated labels ℓ_a , where $\Omega \ni h : \ell^a$. The translations of terms do not use h , as the scope variable a is also dynamically generated and suffices to assign operations uniquely to handlers. The translations are used in the proof of semantics preservation in Appendix C.4.

$$\begin{aligned}
\llbracket - \rrbracket & : \text{Runtime Context} \rightarrow \text{Runtime Context} \\
\llbracket \cdot \rrbracket & = \cdot \\
\llbracket \Omega, h : \ell^a \rrbracket & = \llbracket \Omega \rrbracket, \ell_a : \llbracket A \rrbracket \rightarrow \llbracket B \rrbracket \text{ where } \Sigma \ni \ell : A \rightarrow B \\
\llbracket - \rrbracket & : \text{Term} \rightarrow \text{Term} \\
\llbracket V \ a \rrbracket & = \llbracket V \rrbracket \ell_a \text{ where } \Omega \ni h : \ell^a \\
\llbracket \text{ev}_h \rrbracket & = \text{mod}_{[\ell_a]} (\lambda x^{\llbracket A \rrbracket}. \text{do } \ell_a \ x) \\
& \quad \text{where } \Omega \ni h : \ell^a \text{ and } \Sigma \ni \ell : A \rightarrow B \\
\llbracket \text{handle}_h \ M \text{ with } H : A \mid E \rrbracket & = \text{handle}^{\llbracket E \rrbracket} \llbracket M \rrbracket \text{ with } \llbracket H \rrbracket \\
& \quad \text{where } \Omega \ni h : \ell^a \text{ and } \Sigma \ni \ell : A \rightarrow B
\end{aligned}$$

Translations of evaluation contexts are analogous to the translations of their corresponding terms.

7.3 Capabilities as modal effects

In this section, we present **System C** [Brachthäuser et al. 2022], a core calculus formalising the capability-based effect system of Effekt [Brachthäuser et al. 2026], and show how to encode it into $\text{MET}(\mathcal{S})$. We refer to Brachthäuser et al. [2022] for a complete introduction to **System C**.

7.3.1 System C

Figure 7.8 gives the syntax and typing rules for **System C**, which is fine-grain call-by-value [Levy et al. 2003] and distinguishes between first-class values V , blocks P (second-class functions), and computations M .

We have three typing judgements for values, blocks, and computations, respectively. Judgements for blocks $\Gamma \vdash P : T \mid C$ and computations $\Gamma \vdash M : A \mid C$ explicitly track a capability set C , which contains the capabilities in Γ that may be used.

Value Types	$A, B ::= 1 \mid T \text{ at } C$	Blocks	$P, Q ::= f \mid \{(x : \overline{A}, \overline{f} : \overline{T}) \Rightarrow M\}$
Block Types	$T ::= (\overline{A}, \overline{f} : \overline{T}) \Rightarrow B$		$\mid \text{unbox } V$
Capability Sets	$C ::= \{\overline{f}\}$	Computations	$M, N ::= \text{return } V \mid P(\overline{V}, \overline{Q})$
Contexts	$\Gamma ::= \cdot \mid \Gamma, x : A \mid \Gamma, f :^C T \mid \Gamma, f :^* T$		$\mid \text{let } x = M \text{ in } N$
Values	$V, W ::= x \mid () \mid \text{box } P$		$\mid \text{def } f = P \text{ in } N$
Handlers	$H ::= \{p \ r \mapsto N\}$		$\mid \text{try } \{f^{(A) \Rightarrow B} \Rightarrow M\} \text{ with } H$

$\Gamma \vdash V : A$	$\Gamma \vdash P : T \mid C$
-----------------------	------------------------------

T-UNIT $\frac{}{\Gamma \vdash () : 1}$	T-VAR $\frac{\Gamma \ni x : A}{\Gamma \vdash x : A}$	T-BOX $\frac{\Gamma \vdash P : T \mid C}{\Gamma \vdash \text{box } P : T \text{ at } C}$	T-TRANSPARENT $\frac{\Gamma \ni f :^C T}{\Gamma \vdash f : T \mid C}$	T-TRACKED $\frac{\Gamma \ni f :^* T}{\Gamma \vdash f : T \mid \{f\}}$
--	--	--	---	---

T-UNBOX $\frac{\Gamma \vdash V : T \text{ at } C}{\Gamma \vdash \text{unbox } V : T \mid C}$	T-BLOCK $\frac{\Gamma, x : \overline{A}, \overline{f} :^* \overline{T} \vdash M : B \mid C \cup \{\overline{f}\}}{\Gamma \vdash \{(x : \overline{A}, \overline{f} : \overline{T}) \Rightarrow M\} : (\overline{A}, \overline{f} : \overline{T}) \Rightarrow B \mid C}$	T-BSUB $\frac{\Gamma \vdash P : T \mid C' \quad C' \subseteq C}{\Gamma \vdash P : T \mid C}$
--	--	--

$\Gamma \vdash M : A \mid C$

T-VALUE $\frac{\Gamma \vdash V : A}{\Gamma \vdash \text{return } V : A \mid \cdot}$	T-LET $\frac{\Gamma \vdash M : A \mid C \quad \Gamma, x : A \vdash N : B \mid C'}{\Gamma \vdash \text{let } x = M \text{ in } N : B \mid C \cup C'}$	T-CALL $\frac{\Gamma \vdash P : (\overline{A}_i, \overline{f}_j : \overline{T}_j) \Rightarrow B \mid C \quad \Gamma \vdash V_i : \overline{A}_i \quad \Gamma \vdash Q_j : \overline{T}_j \mid C_j}{\Gamma \vdash P(\overline{V}_i, \overline{Q}_j) : B[\overline{C}_j / \overline{f}_j] \mid C \cup \overline{C}_j}$
---	--	--

T-SUB $\frac{\Gamma \vdash M : A \mid C' \quad C' \subseteq C}{\Gamma \vdash M : A \mid C}$	T-DEF $\frac{\Gamma \vdash P : T \mid C' \quad \Gamma, f :^C T \vdash M : A \mid C}{\Gamma \vdash \text{def } f = P \text{ in } M : A \mid C}$	T-HANDLE $\frac{\Gamma, f :^* (A') \Rightarrow B' \vdash M : A \mid C \cup \{f\} \quad \Gamma, p : A', r :^C (B') \Rightarrow A \vdash N : A \mid C}{\Gamma \vdash \text{try } \{f^{(A') \Rightarrow B'} \Rightarrow M\} \text{ with } \{p \ r \mapsto N\} : A \mid C}$
---	--	---

Figure 7.8: Syntax and typing rules for System C. We mostly follow the syntax of [Brachthäuser et al. \[2022\]](#). The main difference is that we write $\Rightarrow$ for block types to emphasise that they are second-class.

The typing rules of System C are much more involved than those of System F^e as capability tracking is deeply entangled with term constructs such as block constructions (T-BLOCK), block calls (T-CALL), block bindings (T-DEF), and uses of block variables (T-TRANSPARENT and T-TRACKED). Due to space constraints, we focus on explaining these key rules.

There are two rules for uses of block variables as there are two forms of block variable bindings in contexts. A *tracked* binding $f :^* T$ stands for a capability. Rule T-TRACKED tracks f itself in the singleton capability set $\{f\}$. A *transparent* binding $f :^C T$ stands for a user-defined block whose capability set C is known. Rule T-TRANSPARENT tracks C as the capability set.

Rules T-DEF and T-BLOCK both bind block variables. Rule T-DEF binds a block P as a transparent block variable $f :^{C'} T$ where C' is the capability set of P . Rule T-BLOCK binds a list of tracked block variables (capabilities) $\overline{f} :^* T$ whose concrete capability sets are unknown until called. The rule T-BLOCK reflects the roles that block constructions play in capability tracking, as discussed in Section 7.1.2. For instance, all capabilities $\overline{f}$ are added to the capability set of the block body M .

Rule T-CALL fully applies a block P to values $\overline{V}_i$ and blocks $\overline{Q}_j$. The rule reflects the roles that block calls play in capability tracking, as discussed in Section 7.1.2. It replaces each block variable f_j (recall that these variables are bound as $f_j :^* T$ in rule T-BLOCK) with the capability set C_j of the block Q_j in type B . The capability set of the call is the union of the capability sets of P and all its block arguments because all these arguments might be invoked.

Rule T-HANDLE defines a named handler which introduces a capability $f : (A') \Rightarrow B'$ into the scope of M . An operation invoked by calling f in M is handled by this handler. The capability f is added to the capability set of M . The continuation r is introduced as a transparent binding with capability set C as it may only use capabilities in C provided by the context.

System C adopts named handlers and a generative semantics with a reduction relation $M \mid \Omega \rightsquigarrow N \mid \Omega'$, where $\Omega ::= \cdot \mid \ell : (A) \Rightarrow B$ is a context for runtime operation labels, similar to $\text{MET}(\mathcal{X})$. The most interesting reduction rule is E-GEN, which uses a runtime capability value cap_ℓ with a runtime label ℓ to replace a capability f introduced by a handler. The full operational semantics and runtime constructs are given in Figure 7.9.

Figure 7.9 also defines the syntax of runtime constructs for System C. Reduction in System C is defined on both terms and blocks, since we have $\text{unbox } V$ which can reduce. Since System C uses block variables f as both term-level and type-level variables, we need to substitute them separately. We follow Brachthäuser et al. [2022] in overloading the notion of substitution. We write C/f for substitution in types and P/f for substitution in terms.

Typing rules for runtime constructs are as follows.

$$\begin{array}{c}
\text{T-CAP} \\
\frac{\Omega \ni \ell : A \Rightarrow B}{\Omega \mid \Gamma \vdash \text{cap}_\ell : A \Rightarrow B \mid \{\ell\}}
\end{array}
\qquad
\begin{array}{c}
\text{T-HANDLE} \\
\frac{\Omega \ni \ell : A' \Rightarrow B' \quad \Omega \mid \Gamma \vdash M : A \mid C \cup \{\ell\} \quad \Omega \mid \Gamma, p : A', r :^C (B') \Rightarrow A \vdash N : A \mid C}{\Omega \mid \Gamma \vdash \text{try}_\ell M \text{ with } \{p \mapsto N\} : A \mid C}
\end{array}$$

Runtime Labels	ℓ
Runtime Contexts	$\Omega ::= \cdot \mid \Omega, \ell : A \Rightarrow B$
Capability Sets	$C ::= \cdot \mid \{\ell\} \mid \{f\} \mid C \cup C'$
Blocks	$P, Q ::= \dots \mid \text{cap}_\ell$
Computations	$M ::= \dots \mid \text{try}_\ell M \text{ with } H$
Evaluation Contexts	$\mathcal{E} ::= [\] \mid \text{let } x = \mathcal{E} \text{ in } N \mid \text{def } f = \mathcal{E} \text{ in } N \mid \mathcal{E}(\overline{V}, \overline{Q}) \mid \text{try}_\ell \mathcal{E} \text{ with } H$
E-BOX	$\text{unbox } (\text{box } P) \rightsquigarrow P$
E-LET	$\text{let } x = \text{return } V \text{ in } N \rightsquigarrow N[V/x]$
E-DEF	$\text{def } f = P \text{ in } N \rightsquigarrow N[P/f]$
E-CALL	$\{(x : A, f : T) \Rightarrow M\}(\overline{V}, \overline{Q}) \mid \Omega \rightsquigarrow M[\overline{V}/x, \overline{Q}/f, \overline{C}/f] \mid \Omega$ where $\overline{\Omega} \mid \cdot \vdash \overline{Q} : T \mid \overline{C}$
E-GEN	$\text{try } \{f^{A \Rightarrow B} \Rightarrow M\} \text{ with } H \mid \Omega \rightsquigarrow \text{try}_\ell M[\text{cap}_\ell/f, \{\ell\}/f] \text{ with } H \mid \Omega, \ell : A \Rightarrow B$ where ℓ fresh
E-RET	$\text{try}_\ell (\text{return } V) \text{ with } H \rightsquigarrow \text{return } V$
E-OP	$\text{try}_\ell \mathcal{E}[\text{cap}_\ell(V)] \text{ with } H \rightsquigarrow N[V/p, \{(y) \Rightarrow \text{try}_\ell \mathcal{E}[\text{return } y] \text{ with } H\}/r],$ where $\ell \notin \text{bl}(\mathcal{E})$ and $H = \{p \ r \mapsto N\}$
E-LIFT	$\mathcal{E}[M] \mid \Omega \rightsquigarrow \mathcal{E}[N] \mid \Omega', \text{ if } M \mid \Omega \rightsquigarrow N \mid \Omega'$

Figure 7.9: Operational semantics for System C.

7.3.2 Encoding System C in MET(S)

Figure 7.10 encodes System C in MET(S). The term translation is type-directed and defined on typing judgements. We annotate components of a term with their types and capability sets as necessary. In the term translation, we highlight syntax relevant to modalities and type abstraction in grey. The grey parts show how modalities decouple capability tracking. The black parts remain valid programs after type erasure. The encoding is unavoidably more involved than that of System F^c because of the deeper entanglement of capability tracking with blocks. As in Section 7.3.1, we focus on explaining the encoding of block-relevant constructs.

For block constructions and block calls, we have explained their encodings in detail in Section 7.1.2, using the block constructions and calls app_C and app'_C as examples.

A block binding $\text{def } f = P \text{ in } N$ not only binds f to a block P but also annotates the binding $f :^{C'} T$ with the capability set C' of the block P as shown by rule T-DEF in Figure 7.8. For instance, we can bind the block gen_C in Section 7.1.2 to f and apply it to 42. Its typing derivation is as follows.

$$\frac{y :^* \text{Int} \Rightarrow 1 \vdash \text{gen}_C : \text{Int} \Rightarrow 1 \mid \{y\} \quad y :^* \text{Int} \Rightarrow 1, f :^{\{y\}} \text{Int} \Rightarrow 1 \vdash f(42) : 1 \mid \{y\}}{y :^* \text{Int} \Rightarrow 1 \vdash \text{def } f = \text{gen}_C \text{ in } f(42) : 1 \mid \{y\}}$$

$$\begin{array}{ll}
\llbracket - \rrbracket : \text{Cap Set} \rightarrow \text{Effect Context} & \llbracket - \rrbracket : \text{Context} \rightarrow \text{Context} \\
\llbracket \{f\} \rrbracket = \overline{f^*} & \llbracket \cdot \rrbracket = \cdot \\
\llbracket - \rrbracket : \text{Value / Block Type} \rightarrow \text{Type} & \llbracket \Gamma, x : A \rrbracket = \llbracket \Gamma \rrbracket, x : \llbracket A \rrbracket \\
\llbracket 1 \rrbracket = 1 & \llbracket \Gamma, f :^* T \rrbracket = \llbracket \Gamma \rrbracket, f^*, \hat{f} :_{\llbracket f^* \rrbracket} \llbracket T \rrbracket \\
\llbracket T \text{ at } C \rrbracket = \llbracket [C] \rrbracket \llbracket T \rrbracket & \llbracket \Gamma, f :^C T \rrbracket = \llbracket \Gamma \rrbracket, \hat{f} :_{\llbracket [C] \rrbracket} \llbracket T \rrbracket \\
\llbracket (\overline{A}, \overline{f} : \overline{T}) \Rightarrow B \rrbracket = \forall \overline{f^*}. \langle \overline{f^*} \rangle (\llbracket \overline{A} \rrbracket \rightarrow \langle \overline{f^*} \rangle \llbracket \overline{T} \rrbracket \rightarrow \llbracket B \rrbracket) & \llbracket \Gamma \rrbracket_C = \llbracket \Gamma \rrbracket, \hat{\bullet} :_{\llbracket [C] \rrbracket} \\
\llbracket - \rrbracket : \text{Value} \rightarrow \text{Term} & \llbracket - \rrbracket : \text{Block} \rightarrow \text{Term} \\
\llbracket () \rrbracket = () & \llbracket f \rrbracket = \hat{f} \\
\llbracket x \rrbracket = x & \llbracket \{(x : \overline{A}, \overline{f} : \overline{T}) \Rightarrow M\} \rrbracket = \Lambda \overline{f^*}. \overline{\text{mod}}_{\langle \overline{f^*} \rangle} (\lambda x \llbracket \overline{A} \rrbracket \overline{\text{mod}}_{\llbracket f^* \rrbracket} \llbracket \overline{T} \rrbracket \llbracket M \rrbracket) \\
\llbracket \text{box } P : T \text{ at } C \rrbracket = \overline{\text{mod}}_{\llbracket [C] \rrbracket} \llbracket P \rrbracket & \llbracket \text{let mod}_{\llbracket f^* \rrbracket} \hat{f} = f \text{ in } \llbracket M \rrbracket \rrbracket \\
& \llbracket \text{unbox } V : T \mid C \rrbracket = \text{let mod}_{\llbracket [C] \rrbracket} x = \llbracket V \rrbracket \text{ in } x \\
\\
\llbracket - \rrbracket : \text{Computation / Handler} \rightarrow \text{Term} & \\
\llbracket \text{return } V \rrbracket = \llbracket V \rrbracket & \\
\llbracket \text{let } x = M \text{ in } N \rrbracket = \text{let } x = \llbracket M \rrbracket \text{ in } \llbracket N \rrbracket & \\
\llbracket \text{def } f = P : T \mid C \text{ in } N \rrbracket = \text{let } f = \overline{\text{mod}}_{\llbracket [C] \rrbracket} \llbracket P \rrbracket \text{ in let mod}_{\llbracket [C] \rrbracket} \hat{f} = f \text{ in } \llbracket N \rrbracket & \\
\llbracket P(\overline{V}_i, \overline{Q}_j : \overline{T}_j \mid \overline{C}_j) \rrbracket = \text{let mod}_{\langle \llbracket [C] \rrbracket \rangle} x = \llbracket P \rrbracket \llbracket \overline{C}_j \rrbracket \text{ in } x \llbracket \overline{V}_i \rrbracket (\overline{\text{mod}}_{\llbracket [C] \rrbracket} \llbracket \overline{Q}_j \rrbracket) & \\
\llbracket \text{try } \{f^{(A') \Rightarrow B'} \Rightarrow M\} \rrbracket = \text{local } \ell_f : \llbracket A' \rrbracket \rightarrow \llbracket B' \rrbracket \text{ in let}_{\llbracket [C] \rrbracket} \overline{\text{mod}}_{\langle \ell_f \rangle} g = & \\
\llbracket \text{with } H : A \mid C \rrbracket \rrbracket = (\Lambda f^*. \overline{\text{mod}}_{\langle f^* \rangle} (\lambda f. \text{let mod}_{\llbracket f^* \rrbracket} \hat{f} = f \text{ in } \llbracket M \rrbracket)) \ell_f & \\
& \text{in handle}_{\llbracket [C] \rrbracket} (g (\overline{\text{mod}}_{\langle \ell_f \rangle} (\overline{\text{mod}}_{\langle \rangle} (\lambda x \llbracket A' \rrbracket}. \text{do } \ell_f x)))) \text{ with } \llbracket H^{f, C} \rrbracket \\
\llbracket \{p \mapsto N\}^{f, C} \rrbracket = \{\text{return } x \mapsto x, & \\
\ell_f \ p \mapsto \text{let } r' = \overline{\text{mod}}_{\llbracket [C] \rrbracket} (\overline{\text{mod}}_{\langle \rangle} r) \text{ in let mod}_{\llbracket [C] \rrbracket} \hat{r} = r' \text{ in } \llbracket N \rrbracket\} &
\end{array}$$

Figure 7.10: An encoding of System C in MET(S).

The binding of f in the second premise is annotated with its capability set $\{y\}$ since gen_C uses the capability y . We cannot simply encode such a transparent binding by ignoring its capability-set annotation. Instead, we use an absolute modality to simulate this annotation. To encode the binding of f , we wrap the translated block gen_C in the absolute modality $[y]$. The full translation of the above term is as follows, where we include the identity modality omitted in Section 7.1.2.

$$\text{let } f = \overline{\text{mod}}_{\llbracket y^* \rrbracket} (\overline{\text{mod}}_{\langle \rangle} (\lambda x^{\text{Int}}. \hat{y} \ x)) \text{ in let mod}_{\llbracket y^* \rrbracket} \hat{f} = f \text{ in let mod}_{\langle \rangle} f' = \hat{f} \text{ in } f' \ 42$$

We eliminate the modality $[y^*]$ of f and bind it to $\hat{f}$, reminiscent of how we translate block arguments bound by block constructions. In general, for a transparent block variable binding $f :^C T$ in the context, we only keep the binding $\hat{f} :_{\llbracket [C] \rrbracket} \llbracket T \rrbracket$. The intermediate binding f above is introduced locally by the term translation and is not retained by the context translation.

The translation of block-variable uses is simple. We translate each f to its hatted version $\hat{f}$. This simplicity stems from the fact that we eagerly eliminate the modality of

each f after it is introduced, for example, in the translations of block constructions and block bindings.

The translation of named handlers $\text{try } \{f^{(A') \Rightarrow B'} \Rightarrow M\}$ with H is different from the translation of sum_C in Section 7.1.4. The full translation of sum_C is as follows, where we make the omitted identity modalities explicit.

$$\begin{aligned} & \text{local } \ell_y : \text{Int} \rightarrow 1 \text{ in let}_{\llbracket C \rrbracket} \text{mod}_{\langle \ell_y \rangle} g = (\Lambda y^*. \text{mod}_{\langle y^* \rangle} (\lambda y. \text{let mod}_{[y^*]} \hat{y} = y \text{ in } \hat{y} 42; \hat{y} 37; 0)) \ell_y \\ & \text{in handle}_{\llbracket C \rrbracket} (g (\text{mod}_{\langle \ell_y \rangle} (\text{mod}_{\langle \rangle} (\lambda x^{\text{Int}}. \text{do } \ell_y x)))) \\ & \text{with } \{\text{return } x \mapsto x, \ell_y p \ r \mapsto \text{let } r' = \text{mod}_{\llbracket C \rrbracket} (\text{mod}_{\langle \rangle} r) \text{ in} \\ & \quad \text{let mod}_{\llbracket C \rrbracket} \hat{r} = r' \text{ in } p + (\text{let mod}_{\langle \rangle} r'' = \hat{r} \text{ in } r'' ())\} \end{aligned}$$

The main difference is that, instead of directly using the local label ℓ_y for the handled computation, we introduce an effect variable y^* first and substitute it with ℓ_y . This extra layer of abstraction is necessary to keep the translation systematic, because our translations of types and terms consistently translate a capability y to an effect variable y^* . After reducing the type application and the binding of g in the translation above, we obtain the translation of sum_C in Section 7.1.4.

In the return clause, the modality $[\ell_y, \llbracket C \rrbracket]$ appears on the binder of x , so no modality elimination is needed. In the operation clause, the modality $\llbracket C \rrbracket$ appears on the binder of r , while r has a plain function type. Using a modality-parameterised handler is important because in sum_C , the continuation r is a transparent binding of the form $r :^C 1 \rightarrow \text{Int}$ as shown by the typing rule T-HANDLE of System C in Section 7.3.1. We therefore give r the identity modality used by translated block types, rebox it with $\llbracket C \rrbracket$, and eliminate the latter to bind $\hat{r}$, consistently with the translation of transparent bindings.

For contexts, we translate each entry. For a variable binding $x : A$, we translate it homomorphically. For a transparent binding of a block variable $f :^C T$, we translate it to the term variable $\hat{f}$ as discussed in the translation of def above. For a tracked binding of a block variable $f :^* T$, we translate it to an effect variable f^* and a term variable $\hat{f}$ as discussed in Section 7.1.2.

We have the following type preservation theorem with its proof in Appendix C.2.

Theorem 15 (Type Preservation). *If $\Gamma \vdash M : A \mid C$ in System C, then $\llbracket \Gamma \rrbracket_C \vdash \llbracket M \rrbracket : \llbracket A \rrbracket @ \llbracket C \rrbracket$ in $\text{MET}(\mathcal{S})$. Similarly for typing judgements of values and blocks.*

Unlike the previous encodings, we do not have a one-to-many correspondence between the reduction steps in System C and $\text{MET}(\mathcal{S})$ such as Theorems 12 and 14. It is still possible to establish a semantics preservation theorem up to the reduction steps on values in $\text{MET}(\mathcal{S})$. We leave the formal statement and proof of this theorem to future work.

7.3.3 System Ξ

System Ξ [Brachthäuser et al. 2020] is an early core calculus for the Effekt language. System Ξ is essentially a fragment of System C without boxes. As a result, in System Ξ capabilities can never appear in types since we cannot box a second-class block into a first-class value. System Ξ does not track capabilities in typing judgements. There is no danger of capability leakage since capabilities are second-class. Figure 7.11 gives the syntax and typing rules of System Ξ .

Value Types	$A, B ::= 1$	Computations	$M, N ::= \text{return } V \mid \text{let } x = M \text{ in } N$
Block Types	$T ::= (\overline{A}, \overline{T}) \Rightarrow B$		$\mid \text{def } f = P \text{ in } N \mid P(\overline{V}, \overline{Q})$
Contexts	$\Gamma ::= \cdot \mid \Gamma, x : A \mid \Gamma, f : T$		$\mid \text{try } \{f^{A \Rightarrow B} \Rightarrow M\} \text{ with } H$
Values	$V, W ::= x \mid ()$	Handlers	$H ::= \{p \ r \mapsto N\}$
Blocks	$P, Q ::= f \mid \{(x : A, f : T) \Rightarrow M\}$		

$\boxed{\Gamma \vdash V : A}$	$\boxed{\Gamma \vdash P : T}$	$\boxed{\Gamma \vdash M : A}$
-------------------------------	-------------------------------	-------------------------------

$\frac{}{\Gamma \vdash () : 1}$	$\frac{}{\Gamma \vdash x : A}$	$\frac{}{\Gamma \vdash f : T}$	$\frac{}{\Gamma \vdash \{(x : A, f : T) \Rightarrow M\} : (\overline{A}, \overline{T}) \Rightarrow B}$
$\frac{}{\Gamma \vdash \text{return } V : A}$	$\frac{}{\Gamma \vdash \text{let } x = M \text{ in } N : B}$	$\frac{}{\Gamma \vdash \text{def } f = P \text{ in } N : B}$	
$\frac{}{\Gamma \vdash P : (\overline{A}_i, \overline{T}_j) \Rightarrow B}$	$\frac{}{\Gamma \vdash Q_j : T_j}$	$\frac{}{\Gamma \vdash \text{try } \{f^{A' \Rightarrow B'} \Rightarrow M\} \text{ with } \{p \ r \mapsto N\} : A}$	

Figure 7.11: Syntax and typing rules for System Ξ .

The operational semantics of System Ξ is almost identical to that of System C, except that it omits the E-Box rule and capability-set substitutions C/f .

7.3.4 Encoding System Ξ in $\text{MET}(\mathcal{S})$

While our encoding of System C in Section 7.3.2 directly gives an encoding of System Ξ in $\text{MET}(\mathcal{S})$, it introduces unnecessary complexity. Since capabilities never appear in types in System Ξ , we do not need to introduce an effect variable f^* for each capability f in the encoding. It turns out that we can simply encode second-class blocks in System Ξ as first-class functions in $\text{MET}(\mathcal{S})$ without introducing any extra term constructs. For instance, a block $\{(x : A, f : T) \Rightarrow M\}$ is encoded as a function $\lambda x^{\llbracket A \rrbracket} f^{\llbracket T \rrbracket} . \llbracket M \rrbracket$ by changing only

the notation. Figure 7.12 gives the encoding of System Ξ into $\text{MET}(S)$. This encoding is straightforward and does not require any non-identity modalities. We mostly just translate the syntax of second-class blocks in System Ξ to first-class functions in $\text{MET}(S)$. For named handlers, we introduce local labels.

$$\begin{array}{ll}
\llbracket - \rrbracket : \text{Type} \rightarrow \text{Type} & \llbracket - \rrbracket : \text{Value} \rightarrow \text{Term} \\
\llbracket 1 \rrbracket = 1 & \llbracket x \rrbracket = x \\
\llbracket (\overline{A}, \overline{T}) \Rightarrow B \rrbracket = \llbracket \overline{A} \rrbracket \rightarrow \llbracket \overline{T} \rrbracket \rightarrow \llbracket B \rrbracket & \llbracket () \rrbracket = () \\
\\
\llbracket - \rrbracket : \text{Context} \rightarrow \text{Context} & \llbracket - \rrbracket : \text{Block} \rightarrow \text{Term} \\
\llbracket \cdot \rrbracket = \cdot & \llbracket f \rrbracket = f \\
\llbracket \Gamma, x : A \rrbracket = \llbracket \Gamma \rrbracket, x : \llbracket A \rrbracket & \llbracket \{(x : \overline{A}, f : \overline{T}) \Rightarrow M\} \rrbracket = \lambda x^{\llbracket A \rrbracket} \overline{f}^{\llbracket T \rrbracket} . \llbracket M \rrbracket \\
\llbracket \Gamma, f : T \rrbracket = \llbracket \Gamma \rrbracket, f : \llbracket T \rrbracket & \\
\\
\llbracket - \rrbracket : \text{Computation} \rightarrow \text{Term} & \\
\llbracket \text{return } V \rrbracket = \llbracket V \rrbracket & \\
\llbracket \text{let } x = M \text{ in } N \rrbracket = \text{let } x = \llbracket M \rrbracket \text{ in } \llbracket N \rrbracket & \\
\llbracket \text{def } f = P \text{ in } N \rrbracket = \text{let } f = \llbracket P \rrbracket \text{ in } \llbracket N \rrbracket & \\
\llbracket P(\overline{V}, \overline{Q}) \rrbracket = \llbracket P \rrbracket \llbracket \overline{V} \rrbracket \llbracket \overline{Q} \rrbracket & \\
\llbracket \text{try } \{f^{A \Rightarrow B} \Rightarrow M\} \text{ with } H \rrbracket = \text{local } \ell_f : \llbracket A \rrbracket \rightarrow \llbracket B \rrbracket \text{ in} & \\
\quad \text{handle } (\lambda f. \llbracket M \rrbracket) (\lambda x^{\llbracket A \rrbracket} . \text{do } \ell_f x) \text{ with } \llbracket H^f \rrbracket & \\
\llbracket \{p \ r \mapsto N\}^f \rrbracket = \{\text{return } x \mapsto x, \ell_f \ p \ r \mapsto \llbracket N \rrbracket\} &
\end{array}$$

Figure 7.12: An encoding of System Ξ in $\text{MET}(S)$.

We translate runtime contexts as follows.

$$\begin{array}{ll}
\llbracket \cdot \rrbracket & = \cdot \\
\llbracket \Omega, \ell : A \Rightarrow B \rrbracket & = \llbracket \Omega \rrbracket, \ell : \llbracket A \rrbracket \rightarrow \llbracket B \rrbracket
\end{array}$$

We translate runtime constructs used in the operational semantics as follows.

$$\begin{array}{ll}
\llbracket \text{cap}_\ell \rrbracket & = \lambda x^{\llbracket A \rrbracket} . \text{do } \ell \ x \quad \text{where } \Omega \ni \ell : A \Rightarrow B \\
\llbracket \text{try}_\ell M \text{ with } H \rrbracket & = \text{handle } \llbracket M \rrbracket \text{ with } \llbracket H^\ell \rrbracket \\
\llbracket \{p \ r \mapsto N\}^\ell \rrbracket & = \{\text{return } x \mapsto x, \ell \ p \ r \mapsto \llbracket N \rrbracket\}
\end{array}$$

We have the following theorems, whose proofs are in Appendix C.3.

Theorem 16 (Type Preservation). *If $\Gamma \vdash M : A$ in System Ξ , then $\llbracket \Gamma \rrbracket \vdash \llbracket M \rrbracket : \llbracket A \rrbracket @ \cdot$ in $\text{MET}(S)$. Similarly for values and blocks.*

Theorem 17 (Semantics Preservation). *If M is well-typed and $M \mid \Omega \rightsquigarrow N \mid \Omega'$ in System Ξ , then $\llbracket M \rrbracket \mid \llbracket \Omega \rrbracket \rightsquigarrow^* \llbracket N \rrbracket \mid \llbracket \Omega' \rrbracket$ in $\text{MET}(S)$.*

7.4 Implicit effect polymorphism as simply-typed MET

Even without effect variables, MET is sufficiently expressive to encode programs from conventional row-based effect systems, provided effect variables on function arrows always refer to the lexically closest one. This is an important special case, since most functions in practice use at most one effect variable. For example, as of August 2026, the Koka [Leijen 2026] repository contains 583 effectful functions across 134 files, but only 86 functions across 5 files use more than one effect variable; almost all of these are internal primitives for handlers that are not exposed to programmers. Also, almost all programs in the Frank [Convent et al. 2020] repository make no mention of effect variables, relying on syntactic sugar to hide the single effect variable. We formally characterise and prove this claim about the expressiveness of MET.

7.4.1 System $F^{1\epsilon}$

We first define System $F^{1\epsilon}$, the fragment of System F^ϵ with *implicit effect variables* but explicit concrete effects. Effect abstraction omits the bound effect variable. Effect application gives the concrete prefix of the instantiation row, but omits its tail effect variable, which is always the lexically closest one.² As a result, this fragment guarantees that in each scope we can only refer to the lexically closest effect variable.

Figure 7.13 gives the full syntax and typing rules of System $F^{1\epsilon}$. We need no effect variables in System $F^{1\epsilon}$, but we retain effect variables in grey to make the connection to System F^ϵ explicit. For clarity, we also wrap effect rows with grey braces. We add Λ to effect variable bindings in contexts to indicate effect variable bindings. All effect variables are grey and can be erased, whereas the concrete effects at an effect application remain explicit. Everything in red is only used for the encoding. We get a fragment of System F^ϵ by dropping everything in red, dropping the effect row on the value typing judgement, and making everything else black.

For simplicity, we assume that operation argument and result types are not function types, although they may be effect abstractions. For instance, $\ell : (\forall \epsilon. 1 \rightarrow^{\{\epsilon\}} 1) \rightarrow 1$ is allowed, but $\ell : (1 \rightarrow^{\{\epsilon\}} 1) \rightarrow 1$ is not. This restriction ensures our invariant that only the lexically closest effect variable can be referred to; otherwise, an operation call could carry an effect variable ϵ to a scope where ϵ is no longer the lexically closest effect variable.

Typing judgements include $\Gamma \vdash V : A \mid \{E, \epsilon\}$ for values and $\Gamma \vdash M : A \mid \{E, \epsilon\}$ for computations. The effect variable ϵ in the effect row is always the lexically closest effect

²This is different from the notion of implicit polymorphism in the sense of Hindley-Milner type inference, where type abstraction and application are fully inferred. In System $F^{1\epsilon}$, we still keep the syntax for effect abstraction and application as in System F , but erase effect variables from them.

Value Types	$A, B ::= 1 \mid A \rightarrow^{\{E, \varepsilon\}} B \mid \forall \varepsilon. A$	Values	$V, W ::= () \mid x \mid \lambda^{\{E, \varepsilon\}} x^A. M$
Effects	$E, F ::= \cdot \mid \ell, E$		$\mid \Lambda \varepsilon. V \mid \text{handler } H$
Contexts	$\Gamma ::= \cdot \mid \Gamma, x : A$	Computations	$M, N ::= \text{return } V \mid V W \mid V \{E, \varepsilon\}$
	$\mid \Gamma, \Lambda \varepsilon \mid \Gamma, \blacklozenge_E$		$\mid \text{do } \ell V \mid \text{let } x = M \text{ in } N$
Label Contexts	$\Sigma ::= \cdot \mid \Sigma, \ell : A \rightarrow B$	Handlers	$H ::= \{\ell p r \mapsto N\}$

$\Gamma \vdash V : A \mid \{E, \varepsilon\}$

T1-UNIT	T1-VAR	T1-ABS
$\Gamma \vdash () : 1 \mid \{E, \varepsilon\}$	$\Lambda \notin \Gamma_2 \text{ or } A = \forall \varepsilon'. A' \text{ or } A = 1$	$\Gamma, \blacklozenge_F, x : A \vdash M : B \mid \{E, \varepsilon\}$
	$\Gamma_1, x : A, \Gamma_2 \vdash x : A \mid \{E, \varepsilon\}$	$\Gamma \vdash \lambda^{\{E, \varepsilon\}} x^A. M : A \rightarrow^{\{E, \varepsilon\}} B \mid \{F, \varepsilon\}$
T1-EABS	T1-HANDLER	
$\Gamma, \Lambda \varepsilon', \blacklozenge_E \vdash V : A \mid \{\varepsilon'\}$	$\Sigma \ni \ell : A' \rightarrow B' \quad \Gamma, \blacklozenge_F, p : A', r : B' \rightarrow^{\{E, \varepsilon\}} A \vdash N : A \mid \{E, \varepsilon\}$	
$\Gamma \vdash \Lambda \varepsilon'. V : \forall \varepsilon'. A \mid \{E, \varepsilon\}$	$\Gamma \vdash \text{handler } \{\ell p r \mapsto N\} : (1 \rightarrow^{\{\ell, E, \varepsilon\}} A) \rightarrow^{\{E, \varepsilon\}} A \mid \{F, \varepsilon\}$	

$\Gamma \vdash M : A \mid \{E, \varepsilon\}$

T1-VALUE	T1-LET
$\Gamma \vdash V : A \mid \{E, \varepsilon\}$	$\Gamma \vdash M : A \mid \{E, \varepsilon\} \quad \Gamma, x : A \vdash N : B \mid \{E, \varepsilon\}$
$\Gamma \vdash \text{return } V : A \mid \{E, \varepsilon\}$	$\Gamma \vdash \text{let } x = M \text{ in } N : B \mid \{E, \varepsilon\}$
T1-APP	T1-EAPP
$\Gamma \vdash V : A \rightarrow^{\{E, \varepsilon\}} B \mid \{E, \varepsilon\} \quad \Gamma \vdash W : A \mid \{E, \varepsilon\}$	$\Gamma \vdash V : \forall \varepsilon'. A \mid \{E, \varepsilon\}$
$\Gamma \vdash V W : B \mid \{E, \varepsilon\}$	$\Gamma \vdash V \{F, \varepsilon\} : A[\{F, \varepsilon\}/\varepsilon'] \mid \{E, \varepsilon\}$
	T1-DO
	$\Sigma \ni \ell : A \rightarrow B \quad \Gamma \vdash V : A \mid \{\ell, E, \varepsilon\}$
	$\Gamma \vdash \text{do } \ell V : B \mid \{\ell, E, \varepsilon\}$

Figure 7.13: Syntax and typing rules for System F^{1e} .

variable. Unlike System F^e , we also keep the effect variable for value typing judgements to ensure that when defining a function in T1-ABS, we refer to the lexically closest effect variable. Notice that these effect variables are grey, indicating that they are irrelevant to the typing of System F^{1e} . The red effect rows on value typing judgements record the ambient effect rows at which values are translated. Red markers $\blacklozenge_E$ in contexts indicate the transitions of ambient effect rows as in rules T1-ABS, T1-EABS, and T1-HANDLER. These red rows and the red markers are auxiliary annotations used only by the encoding. Because of these red markers, contexts are ordered.

The context entry $\Lambda \varepsilon$ introduced by T1-EABS indicates an effect-variable binding. We retain the black Λ so that we know an effect variable is bound here, whereas the concrete effect variable name is not needed and is therefore grey. Rule T1-VAR prevents

variables from crossing an effect-variable binding entry (Λ), unless their type is an effect abstraction or 1. This condition ensures that we cannot use variables that refer to effect variables that are not the lexically closest. In rule T1-EAPP, we instantiate an effect abstraction with an effect row whose concrete prefix F is explicit and whose tail is the implicit, lexically closest effect variable. The other typing rules are restrictions of the corresponding rules of System F^ϵ . The operational semantics of System $F^{1\epsilon}$ follows the operational semantics of System F^ϵ in Figure 7.2, except that it erases effect variables.

The instantiation operator $[\{E, \epsilon\}/\epsilon']$ implements part of standard type substitution when the grey effect variables are taken into account:

$$\begin{aligned} 1[\{E, \epsilon\}/\epsilon'] &= 1 & (\forall \epsilon''. A)[\{E, \epsilon\}/\epsilon'] &= \forall \epsilon''. A \\ (A \rightarrow^{\{F, \epsilon'\}} B)[\{E, \epsilon\}/\epsilon'] &= A[\{E, \epsilon\}/\epsilon'] \rightarrow^{\{F, E, \epsilon\}} B[\{E, \epsilon\}/\epsilon'] \end{aligned}$$

Two cases cannot arise because of the invariant that only the lexically closest effect variable can be referred to, so we do not include them here. First, when instantiating an effect abstraction, we never need to further substitute in A , which can mention only ϵ'' , whose binding is closer than that of ϵ' . Second, when instantiating a function type, the effect variable on the function arrow is always the same as the effect variable being instantiated, because this is the lexically closest effect variable that can be referred to.

For example, the following System $F^{1\epsilon}$ handler sums all yielded integers.

$$\begin{aligned} \text{asSum} &: \forall \epsilon. (1 \rightarrow^{\{\text{Gen Int}, \epsilon\}} \text{Int}) \rightarrow^{\{\epsilon\}} \text{Int} \\ \text{asSum} &= \Lambda \epsilon. \text{handler } \{\text{yield } x \ r \mapsto x + r \ ()\} \end{aligned}$$

7.4.2 Encoding System $F^{1\epsilon}$ in $\text{MET}(\mathcal{R}_{\text{scp}})$

We now give compositional translations of types, contexts, and terms from System $F^{1\epsilon}$ into $\text{MET}(\mathcal{R}_{\text{scp}})$. The encoding relies on the mask extension of MET from Section 5.1. Our framework does not include masks, because masks are designed for scoped rows. Our target calculus is $\text{MET}(\mathcal{R}_{\text{scp}})$ with the mask extension.

Figure 7.14 gives the type, context, label-context, and term translations. For label contexts, the restriction on operation signatures ensures that their translations do not depend on the effect context, so we use the empty context. We always translate a type to a modal type. For the unit type, we insert the identity modality for uniformity. For a function arrow, the relative modality $\langle E - F | F - E \rangle$ transforms the outside effect context E to the effect context F of the function body. For effect abstraction, the empty absolute modality simulates entering a scope with a fresh effect variable. For contexts, we translate term bindings by moving top-level modalities to their binders, and translate red markers to their corresponding locks. The sequence $\Lambda, \blacklozenge_F$ becomes an absolute lock

since it is introduced by effect abstraction; other markers become relative locks since they are introduced by lambda abstraction or handlers.

$$\begin{array}{l}
\llbracket - \rrbracket_E : \text{Value Type} \rightarrow \text{Type} \\
\llbracket 1 \rrbracket_E = \langle \rangle_1 \\
\llbracket A \rightarrow^F B \rrbracket_E = \langle E - F \mid F - E \rangle (\llbracket A \rrbracket_F \rightarrow \llbracket B \rrbracket_F) \\
\llbracket \forall. A \rrbracket_E = \square \llbracket A \rrbracket.
\end{array}
\quad
\begin{array}{l}
\llbracket - \rrbracket_E : \text{Context} \rightarrow \text{Context} \\
\llbracket \cdot \rrbracket_E = \blacksquare_{[E]} \\
\llbracket \Gamma, x : A \rrbracket_E = \llbracket \Gamma \rrbracket_E, x ;_\mu A' \quad \text{if } \llbracket A \rrbracket_E = \mu A' \\
\llbracket \Gamma, \Lambda, \blacklozenge_F \rrbracket_E = \llbracket \Gamma \rrbracket_F, \blacksquare_{[]} \\
\llbracket \Gamma, \blacklozenge_F \rrbracket_E = \llbracket \Gamma \rrbracket_F, \blacksquare_{\langle F-E \mid E-F \rangle} \quad \text{if } \Gamma \neq \Gamma', \Lambda
\end{array}$$

$$\begin{array}{l}
\llbracket - \rrbracket : \text{Label Context} \rightarrow \text{Label Context} \\
\llbracket \cdot \rrbracket = \cdot \\
\llbracket \Sigma, \ell : A \rightarrow B \rrbracket = \llbracket \Sigma \rrbracket, \ell : \llbracket A \rrbracket. \rightarrow \llbracket B \rrbracket.
\end{array}$$

$$\begin{array}{l}
\llbracket - \rrbracket : \text{Value} \rightarrow \text{Term} \\
\llbracket () \rrbracket = \text{mod}_{\langle \rangle} () \\
\llbracket x : A \mid E \rrbracket = \text{mod}_{\text{top}(\llbracket A \rrbracket_E)} x \\
\llbracket \lambda^F x^A. M : B \mid E \rrbracket = \text{mod}_{\langle E-F \mid F-E \rangle} (\lambda x^{\llbracket A \rrbracket_F}. \text{let mod}_{\text{top}(\llbracket A \rrbracket_F)} x = x \text{ in } \llbracket M \rrbracket) \\
\llbracket \Lambda. V \rrbracket = \text{mod}_{[]} \llbracket V \rrbracket \\
\llbracket \text{handler } H : (1 \rightarrow^{\ell, E} A) \rightarrow^E A \mid F \rrbracket = \text{mod}_{\langle F-E \mid E-F \rangle} (\lambda f^{\llbracket 1 \rightarrow^{\ell, E} A \rrbracket_E}. \text{let mod}_{\langle \ell \rangle} f' = f \text{ in} \\
\quad \text{handle } f' (\text{mod}_{\langle \rangle} ()) \text{ with } \llbracket H : A \mid E \rrbracket) \\
\llbracket \{ \ell \ p \ r \mapsto N \} : A \mid E \rrbracket = \{ \text{return } x \mapsto \text{let}_{\langle \ell \rangle} \text{mod}_{\text{top}(\llbracket A \rrbracket_{\ell, E})} y = x \text{ in } \text{mod}_{\text{top}(\llbracket A \rrbracket_E)} y, \\
\quad \ell \ p \ r \mapsto \text{let mod}_{\text{top}(\llbracket A' \rrbracket_E)} p' = p \text{ in } \llbracket N \rrbracket[p'/p] \} \\
\text{where } \Sigma \ni \ell : A' \rightarrow B'.
\end{array}$$

$$\begin{array}{l}
\llbracket - \rrbracket : \text{Computation} \rightarrow \text{Term} \\
\llbracket \text{return } V \rrbracket = \llbracket V \rrbracket \\
\llbracket \text{let } x = M : A \mid E \text{ in } N \rrbracket = (\lambda x. \text{let mod}_{\text{top}(\llbracket A \rrbracket_E)} x = x \text{ in } \llbracket N \rrbracket) \llbracket M \rrbracket \\
\llbracket V \ W \rrbracket = \text{let mod}_{\langle \rangle} f = \llbracket V \rrbracket \text{ in } f \llbracket W \rrbracket \\
\llbracket (V : \forall. A) F \mid E \rrbracket = \text{let mod}_{[]} x = \llbracket V \rrbracket \text{ in } \text{inst}(x; A; F; E) \\
\llbracket \text{do } \ell \ V \rrbracket = \text{do } \ell \llbracket V \rrbracket
\end{array}$$

Figure 7.14: Encoding of System F^{1e} in $\text{MET}(\mathcal{R}_{\text{scp}})$.

As with all other encodings in this chapter, the term translations are defined on typing judgements. We annotate terms with their types and effect contexts only when the translations need them. All terms are translated to terms whose types have a top-level modality. We use $\text{top}(\mu A) = \mu$ to extract a top-level modality. We eliminate top-level modalities when variables are bound and reintroduce them when variables are used, as shown in the translations of variables, abstractions, and let-bindings. The translation of application eliminates the identity modality of the function. The translation of effect abstraction introduces the empty absolute modality.

Effect application is the most interesting case. It eliminates the empty absolute modality introduced by effect abstraction and then uses the meta-function inst to adjust the top-level modality with respect to the ambient effects E and the instantiation effects F . The goal of this meta-function is to get a value with the appropriate top-level modality

that matches the translation of the instantiated type $A[F/]$ at the ambient effect context E . The meta-function inst is defined as:

$$\begin{aligned} \text{inst}(x; 1; F; E) &= x \\ \text{inst}(x; \forall. A; F; E) &= x \\ \text{inst}(x; A \rightarrow^{E'} B; F; E) &= \text{let}_{\langle E-F \mid F-E \rangle} \text{mod}_{\langle E' \rangle} y = x \text{ in } \text{mod}_{\langle E-(E',F) \mid (E',F)-E \rangle} y \end{aligned}$$

The unit and effect-abstraction cases of inst are trivial because their translations do not depend on the ambient effect context. The only non-trivial case is for a function type. In this case, the variable x is bound at modality $[]$ with type $\llbracket A \rightarrow^{E'} B \rrbracket = \langle E' \rangle (\llbracket A \rrbracket_{E'} \rightarrow \llbracket B \rrbracket_{E'})$. Our goal, the translation of the instantiated function type, is:

$$\llbracket A[F/] \rightarrow^{E',F} B[F/] \rrbracket_E = \langle E - (E', F) \mid (E', F) - E \rangle (\llbracket A[F/] \rrbracket_{E',F} \rightarrow \llbracket B[F/] \rrbracket_{E',F})$$

By an induction on types, it is easy to see that we have $\llbracket A \rrbracket_{E'} = \llbracket A[F/] \rrbracket_{E',F}$ and $\llbracket B \rrbracket_{E'} = \llbracket B[F/] \rrbracket_{E',F}$. Thus we only need to reconstruct the appropriate top-level modality, which is exactly what the modality introduction and elimination in the function type case of inst do. To see how it works, let $\xi = \langle E - F \mid F - E \rangle$, $\mu = \langle E' \rangle$, and $\nu = \langle E - (E', F) \mid (E', F) - E \rangle$. We eliminate the modality μ of the translated function type and reintroduce the modality ν . The modality ξ changes the ambient effect context E to the instantiation effect context F , so that the effect context for the argument and result types of y is E', F as required, not E', E . This ξ modality guarantees the term is well-typed as we have the transformation $\xi \circ \mu \Rightarrow \nu @ E$.

The translation of a handler value follows its presentation in **System F^e** as a higher-order function. The modality of the function argument f is eliminated before f is invoked. The translation of handler clauses generates a return clause that changes the type of the returned value from $\llbracket A \rrbracket_{\ell,E}$ to $\llbracket A \rrbracket_E$. In the operation clause, the modality of the operation argument p is eliminated according to its translated type. The continuation r needs no elimination: the identity modality on its binder is exactly the top-level modality of $\llbracket B' \rightarrow^E A \rrbracket_E$. The runtime handler construct is translated homomorphically as:

$$\llbracket \text{handle } M \text{ with } H : A \mid E \rrbracket = \text{handle } \llbracket M \rrbracket \text{ with } \llbracket H : A \mid E \rrbracket.$$

As an example of type translation, the type of `asSum` from Section 7.4.1 is translated to

$$[](\langle \text{Gen Int} \rangle (\langle 1 \rightarrow \langle \text{Int} \rangle \rightarrow \langle \text{Int} \rangle).$$

Omitting introduction and elimination of identity modalities, its term translation is

$$\llbracket \text{asSum} \rrbracket = \text{mod}_{[]} (\lambda f. \text{let mod}_{\langle \text{Gen Int} \rangle} f' = f \text{ in} \\ \text{handle } f' () \text{ with } \{\text{return } x \mapsto x, \text{yield } p \ r \mapsto p + r ()\})$$

We have the following type preservation theorem.

Theorem 18 (Type Preservation). *If $\Gamma \vdash M : A \mid \{E, \varepsilon\}$, then $\llbracket \Gamma \rrbracket_E \vdash \llbracket M \rrbracket : \llbracket A \rrbracket_E @ E$ in $\text{MET}(\mathcal{R}_{\text{scp}})$. Similarly, if $\Gamma \vdash V : A \mid \{E, \varepsilon\}$, then $\llbracket \Gamma \rrbracket_E \vdash \llbracket V \rrbracket : \llbracket A \rrbracket_E @ E$ in $\text{MET}(\mathcal{R}_{\text{scp}})$.*

Its proof is in Appendix C.5.

7.5 Discussion

In this section, we highlight practical language design insights gleaned from our encodings, discuss the extensibility of our encodings, and outline potential extensions to our framework $\text{MET}(\mathcal{X})$.

7.5.1 Insights for language design

In Section 7.1.3 and Section 7.1.4, we demonstrated how our encodings provide a direct way to compare System F^e and System C. Moreover, our encodings can help inform language design choices based on the following observations.

1. Our encodings together demonstrate that modal effect types are as expressive as the row-based and capability-based effect systems we consider.
2. The encoding of System Ξ (Section 7.3.4) implies that we need not sacrifice first-class functions in order to obtain the benefits of the contextual effect polymorphism of Effekt.
3. The encodings of System C (Section 7.3.2), System Ξ (Section 7.3.4), and System $F^{e+\text{sn}}$ (Section 7.2.4) demonstrate that we can use local labels, the minimal extension introduced in Section 5.6, to simulate the relatively heavyweight feature of named handlers in Effekt and Koka.
4. The encoding of System $F^{e+\text{sn}}$ (Section 7.2.4) further demonstrates that the first-class handler names of Koka offer no extra expressiveness over the second-class local labels of $\text{MET}(\mathcal{X})$.
5. The encoding of System C (Section 7.3.2) shows that instead of having a built-in form of capabilities which can appear at both term and type levels, as in Effekt

and Scala [Boruch-Gruszecki et al. 2023], we can simulate such capabilities by introducing an effect variable for each argument and wrapping the argument in an absolute modality with the corresponding effect variable.

6. The encoding of System $F^{1\epsilon}$ (Section 7.4.2) shows that modal effect types without any effect variables are sufficient to enable modular reuse of effectful programs in many practical cases as characterised by the fragment System $F^{1\epsilon}$ of System F^ϵ .

7.5.2 Extensibility

Our encodings focus on the core idea. None of our encodings considers data types. Most of our encodings, except those in Section 7.2, do not consider polymorphism over value types. It is straightforward to add these features to the encodings in Sections 7.2 and 7.3. Data types and polymorphism are simply translated homomorphically. For the encoding of System $F^{1\epsilon}$ in Section 7.4, encoding data types and polymorphism is more subtle, which we discuss below.

Recall that in Section 7.4.2 we always translate types to modal types and greedily eliminate modalities and lazily introduce them for variables. For variables of data types such as a pair $A \times B$, we just need to further destruct the pair before greedy modality elimination and reconstruct the pair upon lazy modality introduction. This is because the translations of its components may produce terms of type $\mu A'$ and $\nu B'$ with different modalities which require separate modality eliminations. For variables of recursive data types, we need to destruct only to the extent that the data type is unfolded in the function body, where we may treat recursive invocations as opaque. While this requires a higher-order translation, it does not require destructing the recursive data type and eliminating its modalities more than a small number of times.

The translation is complicated because we use let-style modality elimination following MTT. For modalities with certain structures (right adjoints), it is possible to use Fitch-style modality elimination [Clouston 2018], which allows modalities to be eliminated directly without binding [Gratzer 2023; Shulman 2023]. We are interested in exploring whether we can extend MET to use Fitch-style modality elimination and thus give a compositional local encoding for recursive data types. Fortunately, these issues appear to cause problems only in the encoding, not in practice. Functional programs typically use pattern matching in a structured way that plays nicely with the automatic modality elimination of bidirectional typing.

Extending the encoding with polymorphism for value types is tricky because the source calculus System $F^{1\epsilon}$ is not stable under value type substitution. For instance,

the following substitution breaks the condition that function arrows only refer to the lexically closest effect variable:

$$(\forall \varepsilon'. \alpha)[(1 \rightarrow^{\{\varepsilon\}} 1)/\alpha] = \forall \varepsilon'. 1 \rightarrow^{\{\varepsilon\}} 1.$$

This exemplifies the fragility of the syntactic approach of Frank. It is still possible to define the encoding by forcing the substituted type to satisfy the lexical restriction. We leave the full development of such an encoding as future work.

7.5.3 Potential extensions to $\mathbf{MET}(\mathcal{X})$

We discuss three potential extensions to $\mathbf{MET}(\mathcal{X})$ that would enable encodings of more effect systems. We leave their full development as future work.

Effect kinds. We can extend the effect structure to abstract over effect kinds instead of having a single kind $\mathbf{Effect}$. The augmented definition of an effect structure is a triple $\mathcal{X} = \langle \mathbb{R}, :, \equiv \rangle$ where the new component $\mathbb{R}$ is a set of effect kinds. We must extend the kinding and equivalence relations accordingly. As an example of this extension, in order to characterise Rémy-style row types [Rémy 1989], which use a kind system to ensure that there are no duplicate labels, we can declare $\mathbb{R} = \{\mathbf{Row}_{\mathcal{L}} \mid \mathcal{L}\}$, where $\mathcal{L}$ is the set of labels that must not occur in the row. As another example, this extension enables us to combine different effect structures by assigning a kind to each theory. For instance, we can declare two kinds $\mathbf{Set}$ and $\mathbf{Row}$ for theories $\mathcal{S}$ and $\mathcal{R}_{\text{scp}}$, respectively, and then give local labels the kind $\mathbf{Set}$ and global labels the kind $\mathbf{Row}$. We can then treat local labels as sets and global labels as scoped rows.

Presence types. We can associate operation labels in extensions and effect contexts with presence types [Rémy 1994]. Furthermore, instead of predefining the operation types for labels, we can assign operation types to labels in extensions and effect contexts in the manner of Tang et al. [2025]. For instance, the syntax of extensions could be extended to $D ::= \cdot \mid \ell : P, D \mid \varepsilon, D$, where P is a presence type typically defined as $P ::= - \mid \text{Pre}(A \twoheadrightarrow B) \mid \theta$. A label can be absent ($-$), present with a type ($\text{Pre}(A \twoheadrightarrow B)$), or polymorphic over its presence (θ).

Masks. $\mathbf{MET}(\mathcal{X})$ does not include the mask operator and the mask modality $\langle L \rangle$ from Section 5.1. Masking Biernacki et al. [2018]; Convent et al. [2020] is useful for effect systems based on scoped rows where duplicated labels indicate nested handlers for the same operation label. However, it is unclear what masking means for other effect

structures, which is why we do not include it in $\text{MET}(\mathcal{X})$. An interesting direction for future work is to extend $\text{MET}(\mathcal{X})$ with a suitable notion of an abstract mask operator and extend the syntax of relative modalities to $\langle L|D \rangle$, where L is a mask and D is an extension. This extension will require extending the effect structure to define the kinding and equivalence relations of masks.

Part III

Maetel the Language

We design a language Maetel with modal effect types, whose central feature is the Fresco type inference algorithm for inferring first-class modal types, where modal types can be implicitly introduced, eliminated, and used for instantiation.

Chapter [8](#) introduces the Fresco type system for first-class polymorphism. Chapter [9](#) presents the type inference algorithm of Fresco and proves its soundness and completeness. Chapter [10](#) extends Fresco to infer modal effect types.

8 A haunted bidirectional type system

Now, I call out to the spirits of this place.

—Arthas, *Warcraft III* (2002)

I like my type systems to be bidirectional.

—Conor Titania McBride (2018)

Consider the following application of a map function to a Church-encoded list

$$\text{churchMap } (\lambda f.(f \ 42, f \ \text{true})) \ \text{churchIds}$$

where `churchMap` and `churchIds` have the standard types

$$\begin{aligned} \text{churchMap} &: \forall a\ b.(a \rightarrow b) \rightarrow \text{CList } a \rightarrow \text{CList } b \\ \text{churchIds} &: \text{CList } (\forall a.a \rightarrow a) \end{aligned}$$

with the type alias $\text{CList } a = \forall b.b \rightarrow (a \rightarrow b \rightarrow b) \rightarrow b$ for the Church encoding.

This simple example turns out to be surprisingly challenging for many type inference systems. We identify two key challenges for type checking our example:

1. instantiating the type variable a of `churchMap` with the polymorphic type $\forall a.a \rightarrow a$,
2. assigning the polymorphic type $\forall a.a \rightarrow a$ to the parameter f of the first argument term $\lambda f.(f \ 42, f \ \text{true})$.

ML-style type inference [Damas and Milner 1982; Milner et al. 1990] fails immediately since it only supports rank-1 polymorphism, which restricts universal quantifiers to the top level of types, while the types of `churchMap` and `churchIds` have universal quantifiers nested inside function arrows. Bidirectional type inference [Peyton Jones et al. 2007; Dunfield and Krishnaswami 2013] nicely supports *higher-rank polymorphism*, allowing quantifiers to appear flexibly in types, as in the types of `churchMap` and `churchIds`. Moreover, bidirectional type inference allows local type information to flow from functions to arguments, which would address challenge (2) provided challenge (1) had already been addressed. However, most existing work on bidirectional type inference

with higher-rank polymorphism is restricted to monomorphic instantiation, i.e., instantiation with monomorphic types (monotypes), and so does not address challenge (1), which requires polymorphic instantiation, i.e., instantiation with polymorphic types (polytypes). We refer to the combination of higher-rank polymorphism *and* polymorphic instantiation as *first-class polymorphism* (FCP).

Type inference for FCP is a hot topic, with many different approaches, including PolyML [Garrigue and Rémy 1999], LTI [Pierce and Turner 2000], MLF [Le Botlan and Rémy 2003], HML [Leijen 2009], Boxy [Vytiniotis et al. 2006], HMF [Leijen 2008], QML [Russo and Vytiniotis 2009], GI [Serrano et al. 2018], QuickLook [Serrano et al. 2020], FCIF [Kovács 2020], FreezeML [Emrich et al. 2020, 2022], SuperF [Parreaux et al. 2024], LCTI [Xue et al. 2026], and ATIA [Morris 2026]. However, almost all of these approaches still fail to type check the above example.¹ Many of these approaches handle polymorphic instantiation as in challenge (1) by using their own heuristics or constraint solving algorithms to determine that `churchMap` should be instantiated with the polytype $\forall a. a \rightarrow a$ as required by the argument `churchIds`. However, unlike bidirectional type inference, they cannot make use of local type information flow to address challenge (2). Some approaches, such as QuickLook [Serrano et al. 2020], the algorithm behind the `ImpredicativeTypes` extension in GHC, do adopt bidirectional typing techniques to support information flow from functions to arguments. This enables QuickLook to address challenge (2), but it fails to address challenge (1), because it can make only limited use of information about arguments to guide polymorphic instantiation.

Our two key challenges exemplify the need for different directions of type information flow between functions and arguments. To address challenge (1), type information must flow from the second argument `churchIds` to the function `churchMap` for polymorphic instantiation. To address challenge (2), type information must flow from the instantiated function to the first argument $\lambda f. (f\ 42, f\ \text{true})$ in order to assign a polymorphic type to f . However, combining these two directions of type information flow to obtain both of their advantages is much more subtle than it may seem. Several existing type inference systems emphasise mixed directions of type information flow [Vytiniotis et al. 2006; Serrano et al. 2020; Odersky et al. 2001; Xue et al. 2026]. However, as far as we are aware, none of them suffices to type check the above example due to limitations on how different directions of type information flow can be mixed. Moreover, they often rely on ad hoc algorithmic procedures to propagate local type information from arguments to functions, and thus lack declarative specifications.

¹The exception is SuperF [Parreaux et al. 2024], which extends surface type syntax with constraints, but lacks completeness results.

In this chapter, we propose a novel bidirectional type system *Fresco*² for FCP that utilises mixed directions of type information flow. Our central idea is to allow local type information to first flow from arguments to functions, addressing challenge (1), and then flow back from functions to arguments, addressing challenge (2). *Fresco* enables this mixed information flow by haunting the type system. Instead of relying on ad hoc mechanisms for collecting local type information from arguments, *Fresco* uses *skeleton inference*, which mimics type inference, to infer *skeletons* of arguments. Skeletons extend types with *ghosts* that represent unknown information. An ordinary ghost λ represents an unknown type, while a universal ghost Λ represents unknown universal quantifiers. For instance, a skeleton $\Lambda.\lambda \rightarrow \text{Int}$ represents a possibly polymorphic function type with result type Int . Skeletons enable mixed information flow between functions and arguments: known information in an argument skeleton is passed to its function; unknown information denoted by ghosts in the skeleton is then filled in by the function and passed back to the argument.

In order to provide fine-grained control over which information is considered known and thus flows from arguments to functions, we introduce a term-level freezing operator inspired by FreezeML [Emrich et al. 2020]. Freezing a term fixes its skeleton to be exactly what is inferred from the term itself.

In order to succinctly distinguish synthesised (known from the term) and inherited (known from the environment) information within a type, we follow Odersky et al. [2001] in ascribing *colours* to types. This enables us to concisely express the declarative type system in a form similar to more conventional bidirectional type systems, making *Fresco* easier to understand and reason about.

A fundamental guiding principle of *Fresco* is that “polymorphism should never be guessed out of thin air”. When instantiating a polymorphic type, *Fresco* only guesses a universal quantifier if this quantifier is determined by known information collected from the environment, such as via skeleton inference. Inspired by classic ideas on provenance in database systems [Buneman et al. 2001] and Barendregt’s proof for finiteness of developments [Barendregt 1985], we formalise this principle in a declarative specification by tagging each guessed quantifier.

In the rest of this chapter, we first give a high-level overview of *Fresco* in Section 8.1, and then present a declarative type system for *Fresco*. We give a type inference algorithm for *Fresco* in Chapter 9 and prove its soundness and completeness with respect to the declarative type system. We focus on type inference for FCP in this chapter and Chapter 9. In Chapter 10, we extend *Fresco* to infer modal effect types for our language Maetel.

²The name *Fresco* is a loose acronym of the key notions “FREezing, Skeletons, Colours, and ghOsts”.

8.1 Overview

In this section, we give a high-level overview of Fresco. We begin with a brief recap of conventional bidirectional type systems. Then we explain the key technical ideas of Fresco one by one, including skeletons and ghosts, colours and modes, polymorphic subtyping with tags, and freezing. Finally we provide more examples and discuss the limitations of Fresco.

8.1.1 Conventional bidirectional type systems

Conventional bidirectional type systems split typing into inference mode I , where we infer the type of a term, and checking mode C , where we check a term against a given type. A function application rule typically has the following shape.

$$\frac{\Gamma \vdash_I M : A \quad \text{inst}(A) = A_1 \rightarrow B \quad \Gamma \vdash_C N : A_1}{\Gamma \vdash_I M N : B}$$

The application itself is in inference mode. We first infer the type A of the function, then instantiate its type to a function type $A_1 \rightarrow B$, and finally check the argument against the argument type A_1 . Thus information flows from functions to arguments. Such information flow naturally solves challenge (2) at the beginning of this chapter, assuming the argument type is known. In the motivating example, if we are given the type $(\forall a. a \rightarrow a) \rightarrow \text{Int} \times \text{Bool}$, type checking the first argument proceeds as follows.

$$\frac{\Gamma, f : \forall a. a \rightarrow a \vdash_C (f \ 42, f \ \text{true}) : \text{Int} \times \text{Bool}}{\Gamma \vdash_C \lambda f. (f \ 42, f \ \text{true}) : (\forall a. a \rightarrow a) \rightarrow \text{Int} \times \text{Bool}}$$

The checking rule of lambda abstraction assigns the known polytype $\forall a. a \rightarrow a$ to the parameter f .

The missing step is the polymorphic instantiation in challenge (1). Many existing bidirectional type systems restrict attention to higher-rank polymorphism [Dunfield and Pfenning 2004; Dunfield and Krishnaswami 2013; Cui et al. 2023; Zhao and d. S. Oliveira 2022; Jiang et al. 2025] and only allow instantiation with monotypes. We cannot simply extend such systems with arbitrary implicit polymorphic instantiation as this is an undecidable problem [Chrzszcz 1998; Tiuryn and Urzyczyn 1996]. However, in our example the second argument `churchIds` provides the information to instantiate `churchMap` with: the polytype $\forall a. a \rightarrow a$. Thus, in order to address challenge (1), we naturally wish information to flow from arguments to functions. Fresco enables such information flow via skeletons and ghosts.

8.1.2 Skeletons and ghosts

In order to perform polymorphic instantiation, we must collect information from the argument `churchIds` of type `CList` $(\forall a.a \rightarrow a)$, which expands to $\forall b.b \rightarrow ((\forall a.a \rightarrow a) \rightarrow b \rightarrow b) \rightarrow b$. Directly using the argument type is, however, not always sufficient. Even though this works in the motivating example, in general, further implicit instantiation or generalisation may occur on the argument, before its type can be matched against the argument type of the function. For instance, in `foo churchIds` where `foo` : $(A \rightarrow ((\forall a.a \rightarrow a) \rightarrow A \rightarrow A) \rightarrow A) \rightarrow B$, the type of `churchIds` is instantiated with A . As a result, we should make sure that the information we collect from the argument is compatible with the types of the argument after any instantiation or type abstraction.

Fresco uses *skeletons* to represent the information collected from arguments. Skeletons extend types with *ghosts*, including both ordinary ghosts $\mathbb{A}$, which represent unknown types, and universal ghosts $\mathbb{A}$, which represent unknown (possibly empty) sequences of universal quantifiers. For instance, we infer the following skeleton for `churchIds`.

$$\mathbb{A}.\mathbb{A} \rightarrow ((\forall a.a \rightarrow a) \rightarrow \mathbb{A} \rightarrow \mathbb{A}) \rightarrow \mathbb{A}$$

This skeleton is consistent with any possible type that `churchIds` may have. Note that $\mathbb{A}$ is not a binder itself, and all instances of ghosts in a skeleton are unrelated. The four appearances of $\mathbb{A}$ can match any other types we use to instantiate `churchIds`. The top-level $\mathbb{A}$ can match any number of universal quantifiers introduced by type abstraction on `churchIds`. As a result, in the example `foo churchIds` defined above, the skeleton of `churchIds` is consistent with the argument type of `foo`. Meanwhile, the skeleton still shows that list elements have type $\forall a.a \rightarrow a$.

As another example, the skeleton of the first argument $\lambda f.(f\ 42, f\ \text{true})$ to `churchMap` is $\mathbb{A}.\mathbb{A} \rightarrow \mathbb{A}.\mathbb{A} \times \mathbb{A}$. It tells us that the term should have a possibly polymorphic function type with an unknown argument type and a possibly polymorphic product return type. Again, this skeleton is consistent with any possible type for a lambda abstraction of this shape. In particular, it does not commit the type of f to be any specific type, but leaves it as a ghost $\mathbb{A}$. Later, this information can flow back from the argument type of `churchMap` to the lambda abstraction and resolve challenge (2), as illustrated in Section 8.1.1.

Skeletons only carry polymorphic information that we are certain about. For instance, the skeleton of `id` cannot be $\forall \alpha.\alpha \rightarrow \alpha$, because the quantifier $\forall \alpha$ may be instantiated. On the other hand, the quantifier $\forall \alpha$ in the skeleton of `churchIds` is allowed, because it is in a guarded position (under function arrows) which cannot be instantiated.

Skeleton inference rules of Fresco correspond one-to-one with the syntax-directed typing rules of Fresco with systematic minimal changes (Section 8.7). We believe that

skeleton inference smoothly scales to support other typing features. We provide some evidence for this belief by extending our implementation of Fresco with effects and handlers along with a modal effect type system [Tang et al. 2025; Tang and Lindley 2026].

8.1.3 Colours

Information provided by skeletons of arguments is known. When typing the function, we want to synthesise its type guided by such information inherited from the arguments. We use colours, following coloured local type inference (CLTI) [Odersky et al. 2001], to distinguish between synthesised and inherited parts of a type. Unlike in CLTI [Odersky et al. 2001], we allow monotypes to be freely guessed as they can be solved efficiently by unification. As a result, we need only ascribe colours to universal quantifiers, which cannot be guessed. For instance, we can fill in the ghosts of the skeleton of `churchIds` to obtain a coloured type (which we call a *bidirectional* type) as follows for the function `churchMap`.

$$\forall b.b \rightarrow ((\forall a.a \rightarrow a) \rightarrow b \rightarrow b) \rightarrow b$$

The quantifier $\forall a$ is inherited (blue) since it is provided by the skeleton of the argument `churchIds`. The quantifier $\forall b$ is synthesised (red) since it has to be inferred from the function `churchMap` itself; the skeleton of `churchIds` only provides a universal ghost $\mathbb{A}$ in this position. In general, a blue quantifier is inherited from the context and a red quantifier is synthesised from the term itself. Monotypes such as function arrows and type variables are always uncoloured. When writing a monotype with colours, we mean the same type as the uncoloured one. For instance, Int is the same as Int .

Type metavariables are classified into different syntactic categories by their colours. A red A (a synthesised type) has only red quantifiers, a blue A (an inherited type) has only blue quantifiers, and a grey A (a bidirectional type) may have both red and blue quantifiers. A black A has uncoloured quantifiers and only appears in term annotations and contexts. For the same type metavariable A in different colours, these coloured versions are all structurally equal, differing only in the colours of quantifiers. In other words, type metavariables are *colour blind*. We also colour skeletons and have the same conventions for P , P , P and P .

We define the above operation of filling in the ghosts as a skeleton refinement relation $P \blacktriangleright A$. Here, P is an inherited skeleton, and A is a bidirectional type with inherited information from P and new synthesised information. For instance, the above skeleton refinement is written as:

$$\mathbb{A}.\mathbb{A} \rightarrow ((\forall a.a \rightarrow a) \rightarrow \mathbb{A} \rightarrow \mathbb{A}) \rightarrow \mathbb{A} \blacktriangleright \forall b.b \rightarrow ((\forall a.a \rightarrow a) \rightarrow b \rightarrow b) \rightarrow b$$

The refinement relation characterises what other types a skeleton represents. We cannot fill in ghosts with arbitrary types. In $P \blacktriangleright A$, we disallow use of variables introduced by inherited quantifiers in P to fill in ghosts. For instance, $\forall a. \lambda \rightarrow a \blacktriangleright \forall a. a \rightarrow a$ is invalid, even though the result type is well-scoped, because we use the bound variable a to fill in a ghost. This restriction guarantees that when a quantifier is known, all appearances of the bound variable must also be known, since these variable appearances also describe the shape of the polytype.

8.1.4 Modes

In order to type check the motivating example, we need to utilise the information from the second argument. However, a standard application rule such as the one in Section 8.1.1 only takes one argument into account. Some previous work [Pierce and Turner 2000; Serrano et al. 2020] gives the application spine $M \bar{N}$ a special typing rule. However, it is not stable under term transformations such as adding a dummy lambda abstraction, e.g., $(\lambda x. M) \bar{N}$. We adopt a more general approach to dealing with argument terms. As shown in Section 8.1.1, bidirectional typing typically has two modes I and C. Inspired by counters in Contextual Typing [Xue and Oliveira 2024], we add a function mode form F_m (where m ranges over modes), for terms in function position. Our modes are primarily used to track local program structure, unlike traditional bidirectional typing where modes fully determine the direction of information flow. For instance, the typing judgement for churchMap in our running example has the form $\Gamma \vdash_{FFI} \text{churchMap} : A$ for some type A . The mode FFI merely tells us that it is applied to two arguments, and that the whole application is in an inference position with no input type. Information flow is determined by colours in the bidirectional type A .

8.1.5 Flowing back and forth

With all the above ideas (skeletons, ghosts, colours, and modes), we now show the key rule of Fresco, the typing rule for function application. This rule enables information to flow back and forth between functions and arguments.

$$\frac{\Gamma \vdash_c N \Rightarrow P \text{ ①} \quad \Gamma \vdash P \blacktriangleright A \text{ ②} \quad \Gamma \vdash_{Fm} M : A \rightarrow B \text{ ③} \quad \Gamma \vdash_c N : A \text{ ④}}{\Gamma \vdash_m M N : B}$$

Though the rule is declarative, it is easiest to explain via an algorithmic intuition. Skeleton inference ① first infers a synthesised skeleton P for the argument N , which carries the polytype information gleaned from N itself. The skeleton P is red since it is known from the argument N itself. Since the argument skeleton is now known, skeleton refinement ② refines inherited P to a bidirectional type A , which is used as

the argument type of the function M in the typing judgement ③. The skeleton P is blue since for the function M , it is inherited from the context (the argument N). The occurrences of the grey A in ② and ③ denote the same type with the same structure and colours. Inherited quantifiers in A come from the argument N in ①, while synthesised quantifiers in A are inferred from the function M in ③. Finally, since the argument type is fully known, we type check the argument N against the type A in ④. The type A is blue since it is inherited from the context (the function M). In summary, information first flows from the argument to the function, and then flows back from the function to the argument. This differs from the application rule in Section 8.1.1, where information only flows from function to argument.

It is worth noting that skeleton inference ① uses checking mode C. This is because N is in a checking position where potential instantiation or type abstraction may happen. Skeleton inference takes this into account and infers a skeleton that is consistent with any possible type of N .

As an example, we show how the running example is typed using the application rule. With $P_{ids} = \mathbb{A}.\mathbb{A} \rightarrow ((\forall a.a \rightarrow a) \rightarrow \mathbb{A} \rightarrow \mathbb{A}) \rightarrow \mathbb{A}$ and $\text{CList } (\forall a.a \rightarrow a) = \forall b.b \rightarrow ((\forall a.a \rightarrow a) \rightarrow b \rightarrow b) \rightarrow b$, the typing of the outer application goes as follows.

$$\frac{\begin{array}{c} \Gamma \vdash_C \text{churchIds} \Rightarrow P_{ids} \quad \Gamma \vdash P_{ids} \blacktriangleright \text{CList } (\forall a.a \rightarrow a) \\ \Gamma \vdash_{FI} \text{churchMap } (\lambda f.(f \ 42, f \ \text{true})) : \text{CList } (\forall a.a \rightarrow a) \rightarrow \text{CList } (\text{Int} \times \text{Bool}) \\ \Gamma \vdash_C \text{churchIds} : \text{CList } (\forall a.a \rightarrow a) \end{array}}{\Gamma \vdash_{FI} \text{churchMap } (\lambda f.(f \ 42, f \ \text{true})) \ \text{churchIds} : \text{CList } (\text{Int} \times \text{Bool})}$$

In each premise, we highlight the part that demonstrates how information flows from the argument to the function and back. We infer the skeleton P_{ids} of churchIds , refine it to the bidirectional type $\text{CList } (\forall a.a \rightarrow a)$, use this type as the argument type for the inner application at mode FI, and finally type check churchIds . The typing of the inner application goes as follows.

$$\frac{\begin{array}{c} \Gamma \vdash_C (\lambda f.(f \ 42, f \ \text{true})) \Rightarrow \mathbb{A}.\mathbb{A} \rightarrow \mathbb{A}.\mathbb{A} \times \mathbb{A} \\ \Gamma \vdash \mathbb{A}.\mathbb{A} \rightarrow \mathbb{A}.\mathbb{A} \times \mathbb{A} \blacktriangleright (\forall c.c \rightarrow c) \rightarrow \text{Int} \times \text{Bool} \\ \Gamma \vdash_{FFI} \text{churchMap} : ((\forall c.c \rightarrow c) \rightarrow \text{Int} \times \text{Bool}) \rightarrow \text{CList } (\forall a.a \rightarrow a) \rightarrow \text{CList } (\text{Int} \times \text{Bool}) \\ \Gamma \vdash_C \lambda f.(f \ 42, f \ \text{true}) : (\forall c.c \rightarrow c) \rightarrow \text{Int} \times \text{Bool} \end{array}}{\Gamma \vdash_{FI} \text{churchMap } (\lambda f.(f \ 42, f \ \text{true})) : \text{CList } (\forall a.a \rightarrow a) \rightarrow \text{CList } (\text{Int} \times \text{Bool})}$$

In this case the skeleton $\mathbb{A}.\mathbb{A} \rightarrow \mathbb{A}.\mathbb{A} \times \mathbb{A}$ of the argument $\lambda f.(f \ 42, f \ \text{true})$ does not really provide any polymorphic information. The function churchMap infers the quantifier $\forall c$, which is then used as inherited information to type check the argument.

8.1.6 Polymorphic subtyping with tags

We have seen how information flows back and forth. Now we focus on how the polymorphic instantiation of `churchMap` is performed in its typing judgement:

$$\Gamma \vdash_{\text{FFI}} \text{churchMap} : ((\forall c. c \rightarrow c) \rightarrow \text{Int} \times \text{Bool}) \rightarrow \text{CList } (\forall c. c \rightarrow c) \rightarrow \text{CList } (\text{Int} \times \text{Bool})$$

The second argument type carries the inherited quantifier $\forall c$ from the skeleton of `churchIds`. Recall that the type of `churchMap` is $\forall a b. (a \rightarrow b) \rightarrow \text{CList } a \rightarrow \text{CList } b$. We want to instantiate the type of `churchMap` to the bidirectional type above. As usual, we combine instantiation together with type abstraction into a subtyping relation. For the example here, the subtyping is:

$$\begin{aligned} \Gamma \vdash_{\text{FFI}} \forall a b. (a \rightarrow b) \rightarrow \text{CList } a \rightarrow \text{CList } b \leqslant \\ ((\forall c. c \rightarrow c) \rightarrow \text{Int} \times \text{Bool}) \rightarrow \text{CList } (\forall c. c \rightarrow c) \rightarrow \text{CList } (\text{Int} \times \text{Bool}) \end{aligned}$$

Intuitively, this judgement upcasts the LHS to the RHS guided by inherited information on the RHS. We must instantiate a to a polymorphic type $\forall c. c \rightarrow c$ as determined by the inherited information from the second argument. In a traditional bidirectional type system such as that of [Dunfield and Krishnaswami \[2013\]](#), the declarative subtyping rule guesses a monomorphic type for instantiation, which is solved by unification. In *Fresco*, though we can still guess monotypes, we cannot guess polytypes arbitrarily. Otherwise, for instance, `single id` would have possible types `List` $(\forall a. a \rightarrow a)$ and `List` $(A \rightarrow A)$ with no best choice. Instead, we want to guess polytypes only based on the inherited information we know. However, it is challenging to distil the idea of “never guess polymorphism out of thin air” in a declarative way. This is one of the main reasons that many type systems for FCP lack declarative specification and completeness results. For instance, *QuickLook* [[Serrano et al. 2020](#)], which also collects information from arguments to guide polymorphic instantiation of functions, only has an algorithmic specification of instantiation.

We propose a novel approach to characterise the idea of “never guess polymorphism out of thin air” in a declarative way, inspired by work on provenance tracking [[Buneman et al. 2001](#)] and residuals in lambda calculus [[Barendregt 1985](#)]. Within a polytype we guess for instantiation, we tag each quantifier with a unique tag, and track whether each quantifier is determined by inherited information via its tag. For instance, in the subtyping judgement above, for the type variables a and b we guess $\varphi_a = \forall^\epsilon c. c \rightarrow c$ and $\varphi_b = \text{Int} \times \text{Bool}$. The tag ϵ is fresh. After substituting φ_a for a , the second argument type of the function type becomes $\text{CList } (\forall^\epsilon c. c \rightarrow c)$, which matches the argument type on the RHS:

$$\epsilon \mid \Gamma \vdash \text{CList } (\forall^\epsilon c. c \rightarrow c) \sim \text{CList } (\forall c. c \rightarrow c).$$

This *consistency* judgement determines the tag ϵ via the inherited quantifier $\forall c$ on the RHS. The judgement includes a tag context (here the singleton ϵ).

Tags share information about whether a guessed quantifier is determined across different appearances of the same quantifier. A guessed quantifier is determined as long as one of its appearances is determined by inherited information. For instance, in the working example, φ_a appears twice in the instantiated type, but it is determined fully by its second appearance in the second argument type $\text{CList } (\forall^\epsilon c. c \rightarrow c)$. This improves over prior work such as CLTI [Odersky et al. 2001] and Boxy types [Vytiniotis et al. 2006]. These approaches also support mixed information flow through ideas similar to colours and skeletons. However, they have a stricter restriction on polymorphic instantiation, requiring all appearances of the guessed instantiation to be determined, rather than just one.

We now present a negative example where subtyping fails as it would require guessing polymorphism out of thin air. Consider `single id`. We may have the invalid subtyping judgement:

$$\Gamma \vdash_{\text{FI}} \forall a. a \rightarrow \text{List } a \leq (\forall b. b \rightarrow b) \rightarrow \text{List } (\forall b. b \rightarrow b)$$

Guessing $\varphi_a = \forall^\epsilon b. b \rightarrow b$ for the instantiation, we have the subtyping judgement:

$$\Gamma \vdash_{\text{FI}} (\forall^\epsilon b. b \rightarrow b) \rightarrow \text{List } (\forall^\epsilon b. b \rightarrow b) \leq (\forall b. b \rightarrow b) \rightarrow \text{List } (\forall b. b \rightarrow b)$$

We cannot determine the tag ϵ via this judgement, as both appearances of $\forall^\epsilon b$ correspond to synthesised $\forall b$ on the RHS. As a result, in `Fresco`, `List (forall a. a -> a)` is not a valid type for `single id`. This solves the no best choice problem.

Our algorithm for subtyping does not guess types for instantiation. Instead, for an instantiation, it uses a *look* judgement to inspect the RHS for the quantified variable. *Look* is separated into constraint collection and constraint solving. In the motivating example, constraint collection for a extracts from the correspondence between `CList a` and `CList (forall c. c -> c)` that the type for a should be consistent with $\forall c. c \rightarrow c$. Constraint solving then determines the instantiation for a as $\forall c. c \rightarrow c$. One advantage of *look* is that it is local: it only happens in subtyping for solving instantiation, and it does not introduce any unification variable to the context. As a result, our algorithm only does standard global unification for monotypes.

8.1.7 Freezing

The motivating example does not require freezing. The skeleton of `churchIds` already provides enough information. Freezing is useful when argument skeletons do not provide enough information. For instance, in `single id`, `Fresco` does not infer the type `List (forall a. a -> a)`. The reason is that the skeleton of `id` is $\lambda a. \lambda b. a \rightarrow b$. We cannot give `id` a

more concrete skeleton such as $\forall a.a \rightarrow a$, as a skeleton should be consistent with any possible type of the term (Section 8.1.2).

Freezing enables a programmer to fine-tune the information flowing from arguments to functions. Freezing a term disables any potential instantiation or type abstraction on it, regardless of the context. We write $\llbracket M \rrbracket$ for freezing a term M . For instance, $\llbracket \text{id} \rrbracket$ has skeleton $\forall a.a \rightarrow a$. As a result, Fresco infers the type `List` $(\forall a.a \rightarrow a)$ for `single` $\llbracket \text{id} \rrbracket$ via the following derivation.

$$\frac{\begin{array}{c} \Gamma \vdash_C \llbracket \text{id} \rrbracket \Rightarrow \forall b.b \rightarrow b \quad \Gamma \vdash \forall b.b \rightarrow b \blacktriangleright \forall b.b \rightarrow b \\ \Gamma \vdash_{FI} \text{single} : (\forall b.b \rightarrow b) \rightarrow \text{List} (\forall b.b \rightarrow b) \quad \Gamma \vdash_C \llbracket \text{id} \rrbracket : \forall b.b \rightarrow b \end{array}}{\Gamma \vdash_I \text{single} \llbracket \text{id} \rrbracket : \text{List} (\forall b.b \rightarrow b)}$$

The polymorphic instantiation of the function `single` succeeds because the first $\forall b$ is inherited.

Though freezing an argument provides more information for the function, it is not always desirable. For instance, `run` $\llbracket \text{id} \rrbracket$ where `run` : $(\text{Int} \rightarrow \text{Int}) \rightarrow \text{Int}$ is ill-typed, because `run` expects a function type $\text{Int} \rightarrow \text{Int}$, yet $\llbracket \text{id} \rrbracket$ disables instantiation on `id`.

We give one more example of freezing based on a variant of our motivating example. With a standard application function `app` : $\forall a b.(a \rightarrow b) \rightarrow a \rightarrow b$, the following application is ill-typed.

`app ( $\lambda f.(f \ 42, f \ \text{true})$ ) id`

The argument `id` does not provide enough information to instantiate `app` with a polytype $\forall a.a \rightarrow a$, unlike `churchIds` in the motivating example. We can make it well-typed by freezing `id` as follows.

`app ( $\lambda f.(f \ 42, f \ \text{true})$ )  $\llbracket \text{id} \rrbracket$`

Now `app` instantiates with $\forall a.a \rightarrow a$, which then flows back to the lambda abstraction and assigns a polytype to f .

8.1.8 Features and design choices

We discuss some important features and design choices of Fresco with examples.

Order irrelevance. Fresco is not sensitive to argument order. This observation is exemplified by our motivating example, where the second argument `churchIds` provides the information to instantiate the function `churchMap`, and this information flows back to the first argument $\lambda f.(f \ 42, f \ \text{true})$. As another simple example, consider two functions `choose` : $\forall a.a \rightarrow a \rightarrow a$ and `auto` : $(\forall b.b \rightarrow b) \rightarrow (\forall b.b \rightarrow b)$. Fresco infers the same type for both `choose auto id` and `choose id auto`. In both cases, `auto` guides the instantiation

of choose with $(\forall b.b \rightarrow b) \rightarrow (\forall b.b \rightarrow b)$, which then flows back to `id` and instantiates `id` with $\forall b.b \rightarrow b$.

Polymorphic instantiation guided by result type. Section 8.1.6 illustrates how argument types guide polymorphic instantiation of functions. Inherited result types can also guide polymorphic instantiation. For instance, for the type checking judgement $\Gamma \vdash_c \text{nil} : \text{List } (\forall b.b \rightarrow b)$ where $\text{nil} : \forall a.\text{List } a$, its subtyping judgement is $\Gamma \vdash_c \forall a.\text{List } a \leq \text{List } (\forall b.b \rightarrow b)$. We guess $\forall^\epsilon b.b \rightarrow b$ for instantiation, which is determined by the consistency $\epsilon \mid \Gamma \vdash \forall^\epsilon b.b \rightarrow b \sim \forall b.b \rightarrow b$.

Flowing back and forth within one argument. We have seen how information flows from one argument to the function, and then flows back to another argument. Fresco also allows information to flow back to the same argument. For instance, consider the application `foo bar` where $\text{foo} : \forall a.((\forall b.b \rightarrow b) \rightarrow a) \rightarrow a$ and $\text{bar} : \forall c.c \rightarrow (\forall d.d \rightarrow d)$. The argument guides the function to instantiate a with $\forall d.d \rightarrow d$, while the function guides the argument to instantiate c with $\forall b.b \rightarrow b$.

Further instantiation. As one would expect, Fresco allows types used for instantiation to be further instantiated. For instance, given functions $\text{head} : \forall a.\text{List } a \rightarrow a$ and $\text{ids} : \text{List } (\forall a.a \rightarrow a)$, Fresco infers the type `Int` for `head ids 42`. The first application `head ids` instantiates `head` with $\forall a.a \rightarrow a$, which is the result type that is further instantiated with `Int` to match `42`.

Shallow subtyping. Fresco only considers shallow subtyping, i.e., instantiation and type abstraction apply only to top-level types of terms. This is a common design choice for FCP type systems, as it enables more polymorphism to be inferred. For instance, in our motivating example, the skeleton of `churchIds` only contains the quantifier because it is in a guarded position (inside a function arrow) and thus can never be instantiated. In contrast, the skeleton of `id` is $\mathbb{A}.\mathbb{A} \rightarrow \mathbb{A}$ without the top-level quantifier, because it can be instantiated. Modes play a central role in subtyping, determining when subtyping is allowed. For instance, in the subtyping judgement

$$\Gamma \vdash_{\text{FC}} \text{Int} \rightarrow \forall a.a \rightarrow a \leq \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}$$

we can still instantiate $\forall a$ without breaking the shallow subtyping restriction as the function mode `F C` specifies that the term is in function position, and we can instantiate the whole function application term of type $\forall a.a \rightarrow a$ with `Int`.

Value restriction. Fresco adopts the value restriction [Milner et al. 1990; Wright 1995]. Type abstraction only applies to syntactic values. For instance, given $\text{poly} : (\forall a. a \rightarrow a) \rightarrow \text{Int}$, the term $\text{poly } (\lambda x. x)$ is well-typed as information flowing from the function to the argument introduces type abstraction on the argument $\lambda x. x$. In contrast, $\text{poly } (\text{id id})$ is ill-typed as we cannot generalise the application id id because it is not a value. To make it well-typed, we can either η -expand the argument to a value as $\text{poly } (\lambda x. \text{id id } x)$, or freeze the second id as $\text{poly } (\text{id } \lfloor \text{id} \rfloor)$. Freezing allows us to directly infer the type $\forall a. a \rightarrow a$ for the argument $\text{id } \lfloor \text{id} \rfloor$ without a type abstraction. The value restriction is helpful for extending Fresco with effects and inference for modal effect types in Chapter 10.

8.1.9 Limitations

We discuss some limitations of Fresco with examples.

Failures. Consider the subtyping judgement $\Gamma \vdash_{\text{FC}} \forall a. 1 \rightarrow a \leq 1 \rightarrow \forall b. b \rightarrow b$. The LHS type is uninhabited. Unfortunately, this subtyping judgement is invalid in Fresco. Suppose we guess $\forall^e c. c \rightarrow c$ as the instantiation of a . After instantiation we have $\Gamma \vdash_c \forall^e c. c \rightarrow c \leq \forall b. b \rightarrow b$. The quantifier $\forall^e c$ is not determined by $\forall b$ because we can still further instantiate it. However, for the original subtyping judgement, the instantiation with $\forall^e c. c \rightarrow c$ is obviously the simplest solution. Extending Fresco to support such subtyping judgements without violating the “never guess polymorphism out of thin air” principle is an interesting direction for future work.

Ghosts are nameless. In Fresco, ghosts are nameless. Being nameless keeps skeletons simple, but also loses the connection between different occurrences of ghosts. For instance, consider the application `compose poly head` with type signatures:

$$\text{compose} : \forall a b c. (b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c \quad \text{head} : \forall d. \text{List } d \rightarrow d \quad \text{poly} : (\forall e. e \rightarrow e) \rightarrow \text{Int}$$

The skeleton of `poly` is its type $(\forall e. e \rightarrow e) \rightarrow \text{Int}$. The skeleton of `head` is $\mathbb{A}. \text{List } \mathbb{A} \rightarrow \mathbb{A}$, which fails to capture the connection between the argument and result types of `head`. The skeleton of `poly` guides instantiation of b with $\forall e. e \rightarrow e$. Type checking `head` would require instantiating a with the polytype $\text{List } (\forall e. e \rightarrow e)$, which is not possible as the inherited information is too coarse.

In other systems such as QuickLook, this program is well-typed. QuickLook directly uses unification variables when specifying the type inference algorithm, which preserves the connection between argument and result types of `head`. We intentionally omit unification variables from the declarative system of Fresco for simplicity and

predictability. In order to permit such examples, a promising extension to explore is named ghosts. We defer this exploration to future work.

Only flowing back and forth once. Fresco only allows information to flow back and forth once, first flowing from the argument to the function, and then flowing back from the function to the argument. It can be helpful for information to flow back and forth multiple times between function and argument. For instance, consider the application $f\ g$ with two variables:

$$f : \forall c\ d. (c \rightarrow c \rightarrow \text{List } d) \rightarrow \text{Int} \quad g : \forall b. (\forall a. a \rightarrow a) \rightarrow b \rightarrow \text{List } b$$

The skeleton of g is $\lambda. (\forall a. a \rightarrow a) \rightarrow \lambda \rightarrow \text{List } \lambda$, which guides f to instantiate c with $\forall a. a \rightarrow a$. This information then flows back to g and guides it to instantiate b with $\forall a. a \rightarrow a$. However, in order to make it well-typed, we should instantiate d with $\forall a. a \rightarrow a$ as well, which requires information flow from g to f again. We intentionally do not allow such recurring information flow in Fresco for efficiency and predictability reasons. We defer exploring such an extension to future work.

8.1.10 Comparison of type systems for FCP

We make no claim that Fresco is the most expressive type inference system for FCP. There are examples that type check in other systems but not in Fresco (and vice versa). We compare Fresco with representative type inference algorithms for FCP including QuickLook (QL) [Serrano et al. 2020], HMF [Leijen 2008], MLF [Le Botlan and Rémy 2003], ATIA [Morris 2026], and FCIF [Kovács 2020] through examples. Figure 8.2 shows the comparison results, where most examples are taken from Serrano et al. [2018, 2020]. We add a few new examples that benefit from mixed directions of information flow. Type signatures of functions appearing in these tables are given in Figure 8.1. For Fresco, QL, and HMF, we show results with both native lists (Nat.) and Church-encoded lists (Ch.), if they are different. For MLF, ATIA, and FCIF, we only show results with Church-encoded lists.

Our comparison shows that Fresco performs well compared to systems that extend surface type syntax (MLF) or algorithms without declarative specification (QL, ATIA, FCIF), even though Fresco stays with plain System F types and has a declarative specification with completeness results. We believe that Fresco achieves a good balance between expressiveness and simplicity in the sense that programs that are obviously well-typed through mixed directions of local type information between functions and arguments can be type checked in Fresco.

$\text{head} : \forall a. \text{List } a \rightarrow a$	$\text{tail} : \forall a. \text{List } a \rightarrow \text{List } a$
$\text{append} : \forall a. \text{List } a \rightarrow \text{List } a \rightarrow \text{List } a$	$\text{nil} : \forall a. \text{List } a$
$\text{cons} : \forall a. a \rightarrow \text{List } a \rightarrow \text{List } a$	$\text{map} : \forall a b. (a \rightarrow b) \rightarrow \text{List } a \rightarrow \text{List } b$
$\text{single} : \forall a. a \rightarrow \text{List } a$	$\text{length} : \forall a. \text{List } a \rightarrow \text{Int}$
$\text{compose} : \forall a b c. (b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c$	$\text{runST} : \forall a. (\forall s. \text{ST } s \ a) \rightarrow a$
$\text{app} : \forall a b. (a \rightarrow b) \rightarrow a \rightarrow b$	$\text{pair} : \forall a b. a \rightarrow b \rightarrow a \times b$
$\text{argST} : \forall s. \text{ST } s \ \text{Int}$	$\text{revapp} : \forall a b. a \rightarrow (a \rightarrow b) \rightarrow b$
$\text{choose} : \forall a. a \rightarrow a \rightarrow a$	$\text{id} : \forall a. a \rightarrow a$
$\text{auto} : (\forall a. a \rightarrow a) \rightarrow (\forall a. a \rightarrow a)$	$\text{auto}' : \forall b. (\forall a. a \rightarrow a) \rightarrow (b \rightarrow b)$
$\text{ids} : \text{List } (\forall a. a \rightarrow a)$	$\text{inc} : \text{Int} \rightarrow \text{Int}$
$\text{poly} : (\forall a. a \rightarrow a) \rightarrow \text{Int} \times \text{Bool}$	$\text{polys} : \text{List } (\forall a. a \rightarrow a) \rightarrow \text{Int} \times \text{Bool}$

Figure 8.1: Type signatures for functions used in the comparison. For Church encodings, $\text{List } A$ denotes $\forall b. b \rightarrow (A \rightarrow b \rightarrow b) \rightarrow b$.

		Fresco		QL		HMF		MLF	ATIA	FCIF
		Nat.	Ch.	Nat.	Ch.	Nat.	Ch.			
A	Polymorphic instantiation									
A1	$\lambda x y. y$	●		●		●		●	●	●
A2	choose id	●		●		●		●	●	●
A3	choose nil ids	●		●	○	●		●	●	●
A4	$\lambda(x : \forall a. a \rightarrow a). x \ x$	●		●		●		●	●	●
A5	id auto	●		●		●		●	●	●
A6	id auto'	●		○		●		●	●	●
A7	choose id auto	●		●		●		●	●	●
A8	choose id auto'	○		○		○		●	○	○
A9	$f \text{ (choose id) ids where } f : \forall a. (a \rightarrow a) \rightarrow \text{List } a \rightarrow a$	●		●	○	○		●	●	●
A10a	poly id	●		●		●		●	●	●
A10b	poly $(\lambda x. x)$	●		●		●		●	●	●
A10c	id poly $(\lambda x. x)$	●		●		●		●	●	●
A11	$k (\lambda f. (f \ 42, f \ \text{true})) \ xs \text{ where } k : \forall a. a \rightarrow \text{List } a \rightarrow \text{Int}, xs : \text{List } ((\forall a. a \rightarrow a) \rightarrow (\text{Int}, \text{Bool}))$	●		●	○	○		○	○	○
A12a	app poly id	●		●		●		●	●	●
A12b	revapp id poly	●		●		●		●	●	●
A13a	app runST argST	●		●		●		●	●	●
A13b	revapp argST runST	●		●		●		●	●	●
B	Functions on polymorphic lists									
B1a	tail ids	●		●	○	●		●	●	●
B1b	head ids	●		●	○	●		●	●	●
B1c	single id	●		●		●		●	●	●
B2	cons id ids	●		●	○	●		●	●	●
B3	cons $(\lambda x. x)$ ids	●		●	○	●		●	○	●
B4	append (single inc) (single id)	●		●		●		●	●	●
B5	append (single id) ids	●		●	○	○		●	●	●
B6	map poly (single id)	●		●		○		●	●	●
B7	map head (single ids)	○		●	○	●	○	●	●	●
B8	head ids true	●		●	○	●		●	●	●
C	Inference of polymorphic lambda binders and generalisation points									
C1a	$\lambda f. (f \ 42, f \ \text{true})$	○		○		○		○	○	○
C1b	$\lambda(f : \forall a. a \rightarrow a). (f \ 42, f \ \text{true})$	●		●		●		●	●	●
C1c	$g (\lambda f. (f \ 42, f \ \text{true})) \text{ where } g : ((\forall a. a \rightarrow a) \rightarrow \text{Int} \times \text{Bool}) \rightarrow 1$	●		●		○		○	●	●
C2	$r (\lambda x y. y) \text{ where } r : (\forall a. a \rightarrow \forall b. b \rightarrow b) \rightarrow \text{Int}$	●		●		○		●	●	●

Figure 8.2: Comparison of type systems for FCP. ● means accepted and ○ means rejected.

		Fresco		QL		HMF		MLF	ATIA	FCIF
		Nat.	Ch.	Nat.	Ch.	Nat.	Ch.			
E	η-expansion									
E1a	$k\ h\ lst$	○		○		○		○	○	○
E1b	$k\ (\lambda x.h\ x)\ lst$ where $lst : \text{List } (\forall a. \text{Int} \rightarrow a \rightarrow a),$ $k : \forall a.a \rightarrow \text{List } a \rightarrow a,$ $h : \text{Int} \rightarrow \forall a.a \rightarrow a$	●		●	○	●		○	○	○
E2a	$\lambda x.\text{poly } x$	○		○		○		●	○	○
E2b	$(\lambda x.\text{poly } x) : (\forall a.a \rightarrow a) \rightarrow \text{Int} \times \text{Bool}$	●		●		○		●	●	●
E3a	app poly id	●		●		●		●	●	●
E3b	$\text{app } (\lambda x.\text{poly } x)\ \text{id}$ Fresco: $\text{app } (\lambda x.\text{poly } x)\ [\text{id}]$	⦿		○		○		●	○	○
E4a	map poly ids	●		●		●		●	●	●
E4b	$\text{map } (\lambda x.\text{poly } x)\ \text{ids}$	●		●	○	○		●	○	○
E5a	compose poly head	○		●		●		●	●	●
E5b	$\lambda xs.\text{poly } (\text{head } xs)$	○		○		○		●	○	○
E	η-expansion (new)									
E6	$(\lambda f.\text{app } f)\ \text{poly id}$	●		○		○		●	●	●
E7	$(\lambda f.\text{app poly } f)\ \text{id}$ Fresco: $(\lambda f.\text{app poly } f)\ [\text{id}]$	⦿		○		○		●	○	○
F	Bidirectional information flow (new)									
F1	$\text{map } (\lambda f.(f\ \text{true}, f\ 42))\ \text{ids}$	●		●	○	○		○	○	○
F2	$\text{app } (\lambda f.(f\ \text{true}, f\ 42))\ \text{id}$ Fresco: $\text{app } (\lambda f.(f\ \text{true}, f\ 42))\ [\text{id}]$	⦿		○		○		○	○	○
F3	map id ids	○		●	○	●		●	●	●
F4	$(\lambda f.(f\ 42, f\ \text{true}))\ \text{id}$ Fresco: $(\lambda f.(f\ 42, f\ \text{true}))\ [\text{id}]$	⦿		○		○		○	○	○
F5	$\text{pair } (\lambda x.42)\ 42 :$ $((\forall a.a \rightarrow a) \rightarrow \text{Int}) \times \text{Int}$	●		●		●		●	●	○
F6	choose nil ids	●		●	○	●		●	●	●
F7	$\text{head } (\text{choose nil ids})$ Fresco: $\text{head } (\text{choose nil } [\text{ids}])$	●	⦿	●	○	●		●	●	●
F8	$f\ (\lambda x.\text{ids})$ where $f : \forall a.(\text{Int} \rightarrow a) \rightarrow a$	●		○		●		●	●	●
F9	$f\ (\lambda x.(x, \text{ids}))$ where $f : \forall a.(\forall b.b \rightarrow b \times a) \rightarrow a$	●		○		●		●	●	●
F10	$f\ (\lambda x.\text{ids})$ where $f : \forall a.((\forall b.b \rightarrow b) \rightarrow a) \rightarrow a$	●		○		●		●	●	●
F11a	$\text{polys } (\text{single id})$	●		●		○		●	○	●
F11b	$\text{let } xs = \text{single id in polys } xs$ Fresco: $\text{let } xs = \text{single } [\text{id}] \text{ in polys } xs$	⦿		○		○		●	○	○

Figure 8.2: Comparison of type systems for FCP (continued). ● means accepted, ○ means rejected, and ⦿ means accepted with freezing.

8.2 Syntax, colours, and modes

For clarity, we first present the syntax of types and terms, typing, and subtyping for Fresco, and then introduce skeletons and skeleton inference.

Types	$A, B ::= 1 \mid \alpha \mid A \rightarrow B \mid \forall \alpha. A$
Synthesised types	$\textcolor{red}{A}, \textcolor{red}{B} ::= 1 \mid \alpha \mid \textcolor{red}{A} \rightarrow \textcolor{red}{B} \mid \forall \alpha. \textcolor{red}{A}$
Inherited types	$\textcolor{blue}{A}, \textcolor{blue}{B} ::= 1 \mid \alpha \mid \textcolor{blue}{A} \rightarrow \textcolor{blue}{B} \mid \forall \alpha. \textcolor{blue}{A}$
Bidirectional types	$A, B ::= 1 \mid \alpha \mid A \rightarrow B \mid \textcolor{blue}{\forall \alpha. A} \mid \textcolor{red}{\forall \alpha. A}$
Modes	$m ::= \text{I} \mid \text{C} \mid \text{F } m$
Terms	$M, N ::= () \mid x \mid \lambda x. M \mid \lambda x^A. M \mid M : A$ $\mid [M] \mid M N$
Values	$V ::= () \mid x \mid \lambda x. M \mid \lambda x^A. M \mid V : A$
Contexts	$\Gamma ::= \cdot \mid \Gamma, \alpha \mid \Gamma, x : A$

Figure 8.3: Syntax of Fresco.

Figure 8.3 shows the syntax of types, terms, contexts, and modes for Fresco. Types include a unit type (1), type variables (α), function types ($A \rightarrow B$), and universal types ($\forall \alpha. A$). As usual, we have standard well-scopedness ($\Gamma \vdash A$) and α -equivalence ($\Gamma \vdash A \equiv B$) on types, including coloured types, as well as terms. We adopt the Barendregt convention that all variables are distinct [Barendregt 1985]. We say that a type is guarded if it has no top-level quantifiers.

Modes, as introduced in Section 8.1.4, track local program structure, indicating whether a term is in an inference position (I), a checking position (C), or a function position ($\text{F } m$) with function body at mode m . Function modes specify whether the argument type comes from an argument term.

Colours, as introduced in Section 8.1.3, track the direction of information flow for polytypes. We write plain types in black, synthesised types in red, inherited types in blue, and bidirectional types in grey. We only care about whether a quantifier is synthesised or inherited; monotypes are always black. A bidirectional type A tracks whether each quantifier is synthesised ($\forall \alpha$) or inherited ($\forall \alpha$). The syntax of bidirectional types subsumes the syntax of synthesised and inherited types. Types in term annotations and contexts remain black. Consequently, programmers never write coloured types in annotations, and never see variable bindings with coloured types in contexts.

The syntax of terms and contexts is mostly standard. We include a freezing operator $[M]$ inspired by FreezeML [Emrich et al. 2020], which allows us to freeze the type synthesised for M in the current context, and thus treat it as inherited. We have a syntactic category of values V for the value restriction [Milner et al. 1990; Wright 1995].

8.3 Typing rules

Figure 8.4 gives the declarative typing rules for Fresco. We highlight judgements we have not yet defined in light grey background. The typing judgement has the form $\Gamma \vdash_m M : A$, which states that under context Γ in mode m the term M has type A . As usual, we require standard well-scopedness of M and A under Γ as well-formedness conditions for the typing judgement. Moreover, we require compatibility between coloured types and modes $A :^{ty} m$ defined as follows.

$$\frac{}{A :^{ty} I} \quad \frac{}{A :^{ty} C} \quad \frac{B :^{ty} m}{A \rightarrow B :^{ty} Fm}$$

This compatibility relation demonstrates how local program structures interact with information flow directions. In inference mode I , the type is always synthesised (i.e., determined by the term itself). In checking mode C , the type is always inherited (i.e., determined by the term context). In function mode Fm , the type is a function type whose argument type is bidirectional and whose result type depends on m recursively.

$\Gamma \vdash_m M : A$			
D-UNIT $\frac{}{\Gamma \vdash_I () : 1}$	D-VAR $\frac{\Gamma \ni x : A}{\Gamma \vdash_I x : A}$	D-ANNO $\frac{\Gamma \vdash_C M : A}{\Gamma \vdash_I (M : A) : A}$	D-SUB $\frac{\Gamma \vdash_I M : A \quad \Gamma \vdash_m A \leq B}{\Gamma \vdash_m M : B}$
D-FREEZE $\frac{\Gamma \vdash_I M : A}{\Gamma \vdash_C [M] : A}$	D-FORALL $\frac{\Gamma, \alpha \vdash_C V : A}{\Gamma \vdash_C V : \forall \alpha. A}$	D-ABS $\frac{\Gamma, x : A \vdash_{m-} M : B}{\Gamma \vdash_m \lambda x. M : A \rightarrow B}$	D-ABSANNO $\frac{\Gamma, x : A \vdash_{m-} M : B}{\Gamma \vdash_m \lambda x^A. M : A \rightarrow B}$
D-APP $\frac{\Gamma \vdash_C N \Rightarrow P \quad \Gamma \vdash P \blacktriangleright A \quad \Gamma \vdash_{Fm} M : A \rightarrow B \quad \Gamma \vdash_C N : A}{\Gamma \vdash_m M N : B}$			

Figure 8.4: Declarative typing rules for Fresco.

Many of the rules are standard bidirectional typing rules. Rule D-UNIT infers the unit type for unit. Rule D-VAR infers the type A for variable x from the context. Rule D-ANNO checks the term M against the type A inherited from the type annotation and infers the type A . As mentioned in Section 8.1.3, the types A , A , and A in this rule are identical modulo colour. Rule D-FORALL introduces an inherited universal quantifier in checking mode. We require the quantifier to be inherited to avoid guessing type abstraction.

Upcasting and abstraction rules allow an arbitrary mode m in the conclusion, generalising standard bidirectional typing, which only allows checking mode. Rule D-SUB

upcasts the synthesised type A of a term M to the bidirectional type B guided by the mode m . We define the subtyping judgement in Section 8.5. Rule D-ABS introduces an unannotated lambda abstraction. We require the argument type to be inherited to avoid guessing quantifiers without restricting it to monotypes. Rule D-ABSANNO introduces an annotated lambda abstraction. The argument type in the conclusion does not need to be inherited since it is fully determined by the known type annotation. In both rules, we use $m-$ as the mode of the premises defined as follows.

$$(F\ m)- = m \quad I- = I \quad C- = C$$

Since monotypes are uncoloured, our rule D-ABS allows monotypes to be guessed for arguments of lambda abstractions. For instance, we have the following typing derivation where we guess the unit type for the parameter x of a lambda abstraction.

$$\frac{x : 1 \vdash_I x : 1}{\cdot \vdash_I \lambda x. x : 1 \rightarrow 1} \text{ (applying D-ABS)}$$

Rule D-APP is key to enabling bidirectional type information flow, as we explained in detail in Section 8.1.5. We will formally define skeletons P , skeleton inference $\Gamma \vdash_c N \Rightarrow P$, and skeleton refinement $\Gamma \vdash P \blacktriangleright A$ in Sections 8.6 and 8.7.

Rule D-FREEZE freezes the synthesised type A of term M , as explained in Section 8.1.7. It disables further upcasting by switching from inference mode I to checking mode C . For instance, we cannot apply rule D-SUB to the term $[id]$, because $[id]$ must be in checking mode, while D-SUB requires inference mode for its premise. Similarly, we cannot apply rule D-SUB to the term $(\lambda x. [id])$ either, which can only be in checking mode or function modes. Moreover, we may not apply D-FORALL to $[id]$, as the syntax of values V does not include frozen terms.

8.4 Syntax-directed typing rules

Our typing rules in Figure 8.4 enjoy a clear connection to conventional bidirectional typing rules. However, they are not syntax-directed because of overlaps introduced by rules D-SUB and D-FORALL. We incorporate both rules into other rules to obtain a syntax-directed type system. The full syntax-directed typing rules are given in Figure 8.5 where the main changes are highlighted.

Concretely, we incorporate D-SUB into the three base rules D-UNIT, D-VAR, and D-ANNO, having the subtyping judgement as a premise. We incorporate D-FORALL into the two lambda abstraction rules D-ABS and D-ABSANNO, which requires separating the

$$\boxed{\Gamma \vdash_m M : A}$$

$$\begin{array}{c}
\text{SD-UNIT} \\
\frac{\Gamma \vdash_m 1 \leq A}{\Gamma \vdash_m () : A}
\end{array}
\quad
\begin{array}{c}
\text{SD-VAR} \\
\frac{\Gamma \ni x : A \quad \Gamma \vdash_m A \leq B}{\Gamma \vdash_m x : B}
\end{array}
\quad
\begin{array}{c}
\text{SD-ANNO} \\
\frac{\Gamma \vdash_c M : A \quad \Gamma \vdash_m A \leq B}{\Gamma \vdash_m (M : A) : B}
\end{array}
\quad
\begin{array}{c}
\text{SD-FREEZE} \\
\frac{\Gamma \vdash_I M : A}{\Gamma \vdash_c [M] : A}
\end{array}$$

$$\begin{array}{c}
\text{SD-ABSCHECK} \\
\frac{\Gamma, \bar{\alpha}, x : A \vdash_c M : B}{\Gamma \vdash_c \lambda x.M : \forall \bar{\alpha}. A \rightarrow B}
\end{array}
\quad
\begin{array}{c}
\text{SD-ABS} \\
\frac{\mathfrak{m} \neq \mathsf{C} \quad \Gamma, x : A \vdash_{m-} M : B}{\Gamma \vdash_m \lambda x.M : A \rightarrow B}
\end{array}$$

$$\begin{array}{c}
\text{SD-ABSANNOCHECK} \\
\frac{\Gamma, \bar{\alpha}, x : A \vdash_c M : B}{\Gamma \vdash_c \lambda x^A.M : \forall \bar{\alpha}. A \rightarrow B}
\end{array}
\quad
\begin{array}{c}
\text{SD-ABSANNO} \\
\frac{\mathfrak{m} \neq \mathsf{C} \quad \Gamma, x : A \vdash_{m-} M : B}{\Gamma \vdash_m \lambda x^A.M : A \rightarrow B}
\end{array}$$

$$\begin{array}{c}
\text{SD-APP} \\
\frac{\Gamma \vdash_c N \Rightarrow P \quad \Gamma \vdash P \blacktriangleright A \quad \Gamma \vdash_{Fm} M : A \rightarrow B \quad \Gamma \vdash_c N : A}{\Gamma \vdash_m M N : B}
\end{array}$$

Figure 8.5: Syntax-directed declarative typing rules for Fresco.

rules for checking mode, SD-ABSCHECK and SD-ABSANNOCHECK, where we introduce inherited quantifiers, from the rules for other modes, SD-ABS and SD-ABSANNO.

Writing $\Gamma \vdash_m^s M : A$ for the syntax-directed typing judgement, we prove the following soundness and completeness lemmas of the syntax-directed type system in our Rocq mechanisation.

Lemma 19 (Soundness of Syntax-Directed Rule). *If $\Gamma \vdash_m^s M : A$, then $\Gamma \vdash_m M : A$.*

Lemma 20 (Completeness of Syntax-Directed Rule). *If $\Gamma \vdash_m M : A$, then $\Gamma \vdash_m^s M : A$.*

The key challenge is to prove the completeness lemma, which requires showing that the general subsumption rule D-SUB is admissible in the syntax-directed type system. The intuition is that if we upcast the type of a term M non-trivially in the declarative type system, we can always push the upcast down, depending on the structure of M , until we reach a base case where we can apply the rules SD-UNIT, SD-VAR, or SD-ANNO.

This syntax-directed type system helps with the design of skeleton inference in Section 8.7 and the design of type inference algorithms in Chapter 9.

8.5 Subtyping and consistency

A fundamental principle of Fresco is to never guess polytypes out of thin air. Instead, we only guess polytypes that can be determined by inherited type information. In order to make sure that every universal quantifier we guess is determined by some

Decorated types $\varphi ::= 1 \mid \alpha \mid \varphi_1 \rightarrow \varphi_2 \mid \forall^\epsilon \alpha. \varphi$ Tag contexts $\Psi ::= \cdot \mid \Psi, \epsilon$ Tags ϵ
 Unresolved types $T ::= 1 \mid \alpha \mid T_1 \rightarrow T_2 \mid \forall^\delta \alpha. T$ Potential tags $\delta ::= \cdot \mid \epsilon$

$$\boxed{\Gamma \vdash \varphi \# \Psi \quad \varphi \# \Psi}$$

$$\frac{\Gamma \vdash \varphi \quad \varphi \# \Psi}{\Gamma \vdash \varphi \# \Psi} \quad \frac{}{1 \# \cdot} \quad \frac{}{\alpha \# \cdot} \quad \frac{\varphi_1 \# \Psi \quad \varphi_2 \# \Psi'}{\varphi_1 \rightarrow \varphi_2 \# \Psi \uplus \Psi'} \quad \frac{\varphi \# \Psi}{\forall^\epsilon \alpha. \varphi \# \Psi \uplus \{\epsilon\}}$$

Figure 8.6: Syntax and well-formedness rules for tagged types.

$$\boxed{\Psi \mid \Gamma \vdash_m T \leq A}$$

$$\begin{array}{c}
 \text{S-INFER} \\
 \frac{\cdot \mid \Gamma \vdash T \sim A}{\cdot \mid \Gamma \vdash_I T \leq A}
 \end{array}
 \quad
 \begin{array}{c}
 \text{S-CHECK} \\
 \frac{T, A \text{ guarded} \quad \Psi \mid \Gamma \vdash T \sim A}{\Psi \mid \Gamma \vdash_c T \leq A}
 \end{array}
 \quad
 \begin{array}{c}
 \text{S-FORALLR} \\
 \frac{\Psi \mid \Gamma, \alpha \vdash_c T \leq A}{\Psi \mid \Gamma \vdash_c T \leq \forall \alpha. A}
 \end{array}$$

$$\begin{array}{c}
 \text{S-ARROW} \\
 \frac{\Psi_1 \mid \Gamma \vdash T_1 \sim A_1 \quad \Psi_2 \mid \Gamma \vdash_m T_2 \leq A_2}{\Psi_1, \Psi_2 \mid \Gamma \vdash_m T_1 \rightarrow T_2 \leq A_1 \rightarrow A_2}
 \end{array}
 \quad
 \begin{array}{c}
 \text{S-FORALLL} \\
 \frac{m \neq I \quad \Psi' \text{ fresh} \quad \Gamma \vdash \varphi \# \Psi' \quad \Psi, \Psi' \mid \Gamma \vdash_m T[\varphi/\alpha] \leq A}{\Psi \mid \Gamma \vdash_m \forall^\delta \alpha. T \leq A}
 \end{array}$$

$$\boxed{\Psi \mid \Gamma \vdash T \sim A}$$

$$\begin{array}{c}
 \text{C-UNIT} \\
 \frac{}{\cdot \mid \Gamma \vdash 1 \sim 1}
 \end{array}
 \quad
 \begin{array}{c}
 \text{C-VAR} \\
 \frac{\Gamma \ni \alpha}{\cdot \mid \Gamma \vdash \alpha \sim \alpha}
 \end{array}
 \quad
 \begin{array}{c}
 \text{C-FORALLINH} \\
 \frac{\Psi \mid \Gamma, \alpha \vdash T \sim A}{\Psi, \delta \mid \Gamma \vdash \forall^\delta \alpha. T \sim \forall \alpha. A}
 \end{array}
 \quad
 \begin{array}{c}
 \text{C-FORALLSYN} \\
 \frac{\Psi \mid \Gamma, \alpha \vdash T \sim A}{\Psi \mid \Gamma \vdash \forall^\delta \alpha. T \sim \forall \alpha. A}
 \end{array}$$

$$\begin{array}{c}
 \text{C-ARROW} \\
 \frac{\Psi_1 \mid \Gamma \vdash T_1 \sim A_1 \quad \Psi_2 \mid \Gamma \vdash T_2 \sim A_2}{\Psi_1, \Psi_2 \mid \Gamma \vdash T_1 \rightarrow T_2 \sim A_1 \rightarrow A_2}
 \end{array}$$

Figure 8.7: Subtyping and consistency rules for Fresco.

inherited quantifier, we decorate universal quantifiers with tags, whose syntax and well-formedness rules are given in Figure 8.6.

Tags track the provenance of quantifiers, as explained in Section 8.1.6. A decorated type φ is always synthesised and includes a tag ϵ on every quantifier. An unresolved type T is always synthesised and may have tags on some quantifiers. A tag context Ψ is a set of tags. We write Ψ, Ψ' for set union and $\Psi \uplus \Psi'$ for disjoint union. A potential tag δ is either a tag or empty. We define Ψ, δ as Ψ if δ is empty, and as Ψ, ϵ if δ is ϵ . We define well-formedness of decorated types $\Gamma \vdash \varphi \# \Psi$ in Figure 8.6. This judgement combines $\Gamma \vdash \varphi$, the standard well-scopedness judgement of types, and $\varphi \# \Psi$, which checks that Ψ contains all tags in φ and each quantifier has a unique tag.

Figure 8.7 defines the subtyping and consistency relations for Fresco. The subtyping judgement has the form $\Psi \mid \Gamma \vdash_m T \leq A$ (abbreviated as $\Gamma \vdash_m T \leq A$ if Ψ is empty), which states that given tags Ψ under context Γ in mode m , unresolved type T is a subtype of bidirectional type A . As with the typing judgement, we require standard well-scopedness and $A :^{ty} m$ as well-formedness conditions for the subtyping judgement. The consistency judgement has the form $\Psi \mid \Gamma \vdash T \sim A$, which states that given tags Ψ under context Γ the unresolved type T is consistent with the bidirectional type A . Intuitively, in addition to their usual roles, subtyping and consistency rules aim to use inherited quantifiers from A to determine tagged quantifiers in T , so that these tagged quantifiers are not guessed out of thin air. The tag context Ψ does not contain all tags in T , but only those determined from A .

Rule S-INFER solves no tags in inference mode. Rule S-CHECK degenerates to consistency when both sides are guarded. Rule S-FORALLR introduces an inherited quantifier in checking mode. Rule S-ARROW upcasts a function type $T_1 \rightarrow T_2$ to a bidirectional function type $A_1 \rightarrow A_2$ in function mode Fm by unifying the argument types and recursively upcasting the result type. This does not break our claim of only allowing shallow subtyping, because the mode Fm indicates that the term of this function type is indeed applied to an argument, so the upcasting of its result type is allowed. The solved tags of the conclusion are the union of those from the two premises. In this sense, the tag context is a relevant context: tags may be duplicated, but each tag it contains must be determined.

Rule S-FORALLL is the most interesting one, since it is where guessing polytypes happens. We have explained its core idea with the motivating example in Section 8.1.6. It requires the mode not to be I to avoid overlapping with S-INFER where no instantiation is allowed. For instance, the type $\forall \alpha. \alpha \rightarrow \alpha$ can only be upcast to itself in inference mode; we cannot instantiate it. For the instantiation, we guess a decorated type φ whose tags in Ψ' are unique and fresh. Adding all tags Ψ' of φ to the tag context of the premise subtyping judgement ensures that every quantifier in φ is determined by inherited information from A . The potential tag δ is not solved in this rule since we cannot guarantee that the quantifier being instantiated must exist.

Most consistency rules are straightforward. Rule C-FORALLINH is the only rule where we add a tag to the tag context, since this tag is determined by the inherited quantifier on the RHS. Rule C-FORALLSYN does not determine any tag since the RHS is synthesised.

8.6 Skeletons and ghosts

Skeletons extend types with ghosts to represent unknown type information. Figure 8.8 gives the syntax, well-formedness, and refinement rules for skeletons.

Skeletons	$P, Q ::= 1 \mid \alpha \mid P \rightarrow Q \mid \forall \alpha. P$	$\mid \mathbb{A} \mid \mathbb{A}. P$
Synthesised skeletons	$\textcolor{red}{P}, \textcolor{red}{Q} ::= 1 \mid \alpha \mid \textcolor{red}{P} \rightarrow \textcolor{red}{Q} \mid \forall \alpha. \textcolor{red}{P}$	$\mid \mathbb{A} \mid \mathbb{A}. \textcolor{red}{P}$
Inherited skeletons	$\textcolor{blue}{P}, \textcolor{blue}{Q} ::= 1 \mid \alpha \mid \textcolor{blue}{P} \rightarrow \textcolor{blue}{Q} \mid \forall \alpha. \textcolor{blue}{P}$	$\mid \mathbb{A} \mid \mathbb{A}. \textcolor{blue}{P}$
Bidirectional skeletons	$P, Q ::= 1 \mid \alpha \mid P \rightarrow Q \mid \textcolor{red}{\forall \alpha. P} \mid \textcolor{blue}{\forall \alpha. P}$	$\mid \mathbb{A} \mid \mathbb{A}. P$

$\Gamma \mid \Delta \vdash P$

$\frac{}{\Gamma \mid \Delta \vdash \mathbb{A}}$	$\frac{}{\Gamma \mid \Delta \vdash 1}$	$\frac{\Gamma \ni \alpha}{\Gamma \mid \Delta \vdash \alpha}$	$\frac{\Gamma \mid \Delta \vdash P \quad \Gamma \mid \Delta \vdash Q}{\Gamma \mid \Delta \vdash P \rightarrow Q}$
$\frac{\Gamma, \alpha \mid \Delta, \alpha \vdash P}{\Gamma \mid \Delta \vdash \forall \alpha. P}$		$\frac{P \text{ guarded} \quad \Gamma \setminus \Delta \mid \cdot \vdash P}{\Gamma \mid \Delta \vdash \mathbb{A}. P}$	

$\Gamma \mid \Delta \vdash \textcolor{blue}{P} \triangleright \textcolor{blue}{Q}$

$\frac{}{\Gamma \mid \Delta \vdash 1 \triangleright 1}$	$\frac{}{\Gamma \mid \Delta \vdash \alpha \triangleright \alpha}$	$\frac{\Gamma \mid \Delta \vdash \textcolor{blue}{P}_1 \triangleright \textcolor{blue}{Q}_1 \quad \Gamma \mid \Delta \vdash \textcolor{blue}{P}_2 \triangleright \textcolor{blue}{Q}_2}{\Gamma \mid \Delta \vdash \textcolor{blue}{P}_1 \rightarrow \textcolor{blue}{P}_2 \triangleright \textcolor{blue}{Q}_1 \rightarrow \textcolor{blue}{Q}_2}$	$\frac{\Gamma, \alpha \mid \Delta, \alpha \vdash \textcolor{blue}{P} \triangleright \textcolor{blue}{Q}}{\Gamma \mid \Delta \vdash \forall \alpha. \textcolor{blue}{P} \triangleright \forall \alpha. \textcolor{blue}{Q}}$
$\frac{\Delta \cap \text{ftv}(\textcolor{red}{Q}) = \emptyset}{\Gamma \mid \Delta \vdash \mathbb{A} \triangleright \textcolor{red}{Q}}$	$\frac{\Gamma \mid \Delta \vdash \textcolor{blue}{P} \triangleright \textcolor{blue}{Q}}{\Gamma \mid \Delta \vdash \mathbb{A}. \textcolor{blue}{P} \triangleright \textcolor{blue}{Q}}$	$\frac{\Gamma, \alpha \mid \Delta \vdash \mathbb{A}. \textcolor{blue}{P} \triangleright \textcolor{blue}{Q}}{\Gamma \mid \Delta \vdash \mathbb{A}. \textcolor{blue}{P} \triangleright \textcolor{red}{\forall \alpha. Q}}$	$\frac{\Gamma \mid \Delta \vdash \textcolor{blue}{P} \triangleright \textcolor{blue}{Q}}{\Gamma \mid \Delta \vdash \mathbb{A}. \textcolor{blue}{P} \triangleright \mathbb{A}. \textcolor{blue}{Q}}$

Figure 8.8: Syntax, well-formedness, and refinement rules for skeletons.

A plain ghost $\mathbb{A}$ represents an unknown type. A universal ghost $\mathbb{A}$ in $\mathbb{A}.P$ represents a sequence of unknown universal quantifiers. Like types, skeletons may be coloured as synthesised skeletons ($\textcolor{red}{P}$), inherited skeletons ($\textcolor{blue}{P}$), and bidirectional skeletons (P). Only concrete quantifiers are coloured; ghosts themselves are always black. The guarded condition is extended to skeletons, with $\mathbb{A}.P$ and $\mathbb{A}$ being unguarded, and similarly extended to coloured skeletons.

Well-formedness of skeletons. We require universal ghosts to only appear at the end of a sequence of quantifiers. This is important for our algorithm to deterministically solve the unification between two quantifier sequences. Moreover, given a skeleton P , for each universal ghost $\mathbb{A}.Q$ in P , a type variable free in Q cannot be bound in P . This maintains the above invariant under instantiation. For instance, under a context containing β , $\forall \alpha. \mathbb{A}. \alpha$ is not well-formed, because the bound variable α appears under a universal ghost, while $\forall \alpha. \mathbb{A}. \beta$ is well-formed, because β is not bound by the skeleton but by the context.

We characterise these requirements on universal ghosts via the well-formedness of skeletons. We write $\Gamma \vdash P$ if the bidirectional skeleton P is well-formed under context Γ . We use an auxiliary set Δ to track the type variables bound by universal quantifiers in the skeleton, since these variables cannot be used under universal ghosts. The full judgement thus has the form $\Gamma \mid \Delta \vdash P$. We write $\Gamma \setminus \Delta$ for the context after removing the variables in Δ from Γ . The well-formedness rules are given in Figure 8.8, where we write $\forall\alpha$ to match both $\forall\alpha$ and $\forall\alpha$. Since colours play no role here, we define well-formedness of uncoloured skeletons ($\Gamma \vdash P$) by simply ignoring colours in these rules.

Skeleton refinement. We define a skeleton refinement relation $\Gamma \vdash P \blacktriangleright Q$ which fills in ghosts on the LHS with synthesised skeletons to obtain the RHS. We require well-formedness of both P and Q under Γ as well-formedness conditions of the refinement judgement. Skeleton refinement characterises what skeletons a skeleton can represent. As discussed in Section 8.1.3, no ghost in a skeleton P can represent skeletons that use type variables bound in P . To enforce this requirement, we augment the full judgement form to $\Gamma \mid \Delta \vdash P \blacktriangleright Q$, where the set Δ tracks type variables bound in P . The refinement rules are given in Figure 8.8. As with well-formedness, we define the refinement of uncoloured skeletons ($\Gamma \vdash P \blacktriangleright Q$) by ignoring colours in these rules.

For refining $\mathbb{A}$, we require Q to contain no variables from Δ . For refining $\mathbb{A}$, we have three rules. The first rule removes a $\mathbb{A}$ on the left. The second rule feeds a concrete quantifier α on the right into a $\mathbb{A}$ on the left. We do not add α to Δ in the premise because it is bound by the synthesised quantifier and can be used when filling ghosts, unlike variables bound by inherited quantifiers in Δ . The third rule removes a $\mathbb{A}$ on the left and the right.

8.7 Skeleton inference

There are different skeleton inference strategies. The most naive approach is to always infer the trivial skeleton $\mathbb{A}$, which provides no useful information. A more refined and principled approach to skeleton inference is to follow the structure of the typing rules, inserting ghosts as needed.

To define skeleton inference, we extend the syntax of variable bindings $x : A$ in contexts Γ to $x : P$. Figure 8.9 gives the skeleton inference rules. Skeleton inference has the form $\Gamma \vdash_m M \Rightarrow P$, which means that the term M has skeleton P under context Γ in mode m . We require well-formedness of M and P under Γ as well-formedness conditions

for the skeleton inference judgement, as well as the compatibility between bidirectional skeletons and modes $P :^{\text{sk}} m$ defined as follows.

$$\frac{}{P :^{\text{sk}} \text{I}} \quad \frac{}{P :^{\text{sk}} \text{C}} \quad \frac{Q :^{\text{sk}} m}{P \rightarrow Q :^{\text{sk}} \text{F} m}$$

The only difference from $A :^{\text{ty}} m$ defined in Section 8.3 is that checking mode also requires a synthesised skeleton, since the goal of skeleton inference in checking mode is to infer a skeleton consistent with any types that the term can be checked against, instead of checking the term against an inherited type.

$$\boxed{\Gamma \vdash_m M \Rightarrow P}$$

$$\begin{array}{c} \text{P-UNIT} \\ \frac{\Gamma \vdash_m^{\text{sk}} 1 \leq P}{\Gamma \vdash_m () \Rightarrow P} \end{array} \quad \begin{array}{c} \text{P-VAR} \\ \frac{\Gamma \ni x : P \quad \Gamma \vdash_m^{\text{sk}} P \leq Q}{\Gamma \vdash_m x \Rightarrow Q} \end{array} \quad \begin{array}{c} \text{P-ANNO} \\ \frac{\Gamma \vdash_m^{\text{sk}} A \leq P}{\Gamma \vdash_m (M : A) \Rightarrow P} \end{array} \quad \begin{array}{c} \text{P-FREEZE} \\ \frac{\Gamma \vdash_{\text{I}} M \Rightarrow P}{\Gamma \vdash_{\text{C}} [M] \Rightarrow P} \end{array}$$

$$\begin{array}{c} \text{P-ABSCHECK} \\ \frac{\Gamma, x : \mathbb{A} \vdash_{\text{C}} M \Rightarrow Q}{\Gamma \vdash_{\text{C}} \lambda x. M \Rightarrow \mathbb{A}. \mathbb{A} \rightarrow Q} \end{array} \quad \begin{array}{c} \text{P-ABS} \\ \frac{m \neq \text{C} \quad \Gamma, x : P \vdash_{m-} M \Rightarrow Q}{\Gamma \vdash_m \lambda x. M \Rightarrow P \rightarrow Q} \end{array}$$

$$\begin{array}{c} \text{P-ABSANNOCHECK} \\ \frac{\Gamma, x : A \vdash_{\text{C}} M \Rightarrow Q}{\Gamma \vdash_{\text{C}} \lambda x^A. M \Rightarrow \mathbb{A}. A \rightarrow Q} \end{array} \quad \begin{array}{c} \text{P-ABSANNO} \\ \frac{m \neq \text{C} \quad \Gamma, x : A \vdash_{m-} M \Rightarrow Q}{\Gamma \vdash_m \lambda x^A. M \Rightarrow A \rightarrow Q} \end{array}$$

$$\begin{array}{c} \text{P-APP} \\ \frac{\Gamma \vdash_{\text{C}} N \Rightarrow P \quad \Gamma \vdash P \blacktriangleright P' \quad \Gamma \vdash_{\text{F} m} M \Rightarrow P' \rightarrow Q}{\Gamma \vdash_m M N \Rightarrow Q} \end{array}$$

Figure 8.9: Skeleton inference rules for Fresco.

Skeleton inference rules correspond one-to-one with the syntax-directed variant of the typing rules in Sections 8.3 and 8.4. There are two main differences. The first is that in rules P-ANNO and P-APP we do not need to do type checking, because skeleton inference only cares about inferring a skeleton for the term, rather than ensuring that the term is well-typed. The second is that in P-ABSCHECK and P-ABSANNOCHECK we insert universal ghosts, while in the corresponding syntax-directed typing rules SD-ABSCHECK and SD-ABSANNOCHECK we have inherited universal quantifiers.

In P-UNIT, P-VAR, and P-ANNO, we use the subtyping judgement between skeletons $\Gamma \vdash_m^{\text{sk}} P \leq Q$ which also mimics the subtyping judgement for types in Section 8.5. Unlike typing, a general subsumption rule like D-SUB does not work for skeleton inference. Otherwise, for example, we can infer a skeleton $\mathbb{A}.1 \rightarrow (\forall \alpha. \alpha \rightarrow \alpha)$ for $\lambda x^1. \text{id}$ under mode C, by first inferring the skeleton of this term under mode I and then applying a

Decorated skeletons	$\psi ::= 1 \mid \alpha \mid \psi_1 \rightarrow \psi_2 \mid \forall^e \alpha. \psi \mid \mathbb{A} \mid \mathbb{A}.\psi$
Unresolved skeletons	$S ::= 1 \mid \alpha \mid S_1 \rightarrow S_2 \mid \forall^\delta \alpha. S \mid \mathbb{A} \mid \mathbb{A} \mid \mathbb{A}.S \mid \mathbb{A}.S$

$\Gamma \vdash^{\text{sk}} \psi \# \Psi$	$\psi \# \Psi$
--	----------------

$\frac{\Gamma \vdash \psi \quad \psi \# \Psi}{\Gamma \vdash^{\text{sk}} \psi \# \Psi}$	$\frac{}{1 \# \cdot}$	$\frac{}{\alpha \# \cdot}$	$\frac{\psi_1 \# \Psi \quad \psi_2 \# \Psi'}{\psi_1 \rightarrow \psi_2 \# \Psi \uplus \Psi'}$	$\frac{\psi \# \Psi}{\forall^e \alpha. \psi \# \Psi \uplus \{\epsilon\}}$
	$\frac{}{\mathbb{A} \# \cdot}$		$\frac{\psi \# \Psi}{\mathbb{A}.\psi \# \Psi}$	

Figure 8.10: Syntax and well-formedness rules for decorated skeletons.

general subsumption rule. However, this skeleton is not compatible with instantiation of id . The expected skeleton is $\mathbb{A}.1 \rightarrow (\mathbb{A}.\mathbb{A} \rightarrow \mathbb{A})$.

8.8 Skeleton subtyping and consistency

As in Section 8.5, we define decorated skeletons and unresolved skeletons in Figure 8.10.

In addition to adding tags to quantifiers, we also use dotted ghosts $\mathbb{A}$ and $\mathbb{A}$ in decorated skeletons. Dotted ghosts are used to avoid trivial guesses for instantiation as explained later. As in Section 8.5, we write $\psi \# \Psi$ if Ψ contains all tags in ψ and each quantifier has a unique tag. We define an erasure operation $[S]$ which gives a synthesised skeleton by removing all tags and transforming dotted ghosts to normal ghosts. Well-formedness and guardedness of decorated skeletons and unresolved skeletons are defined by those of their erasure. The judgement $\Gamma \vdash^{\text{sk}} \psi \# \Psi$ combines $\Gamma \vdash \psi$ with $\psi \# \Psi$, as defined in Figure 8.10. The superscript sk distinguishes this judgement from the well-formedness judgement of decorated types in Section 8.5, which we sometimes also annotate with ty as $\Gamma \vdash^{\text{ty}} \varphi \# \Psi$. We define the sort $n = \text{ty} \mid \text{sk}$ and write $\Gamma \vdash^n \psi \# \Psi$ to range over both.

Figure 8.11 defines skeleton subtyping and skeleton consistency. Most of the subtyping and consistency rules mimic their type counterparts in Figure 8.7. We highlight the new rules for ghosts; other rules coincide with those for types. As with well-formedness judgements, we annotate the subtyping judgement with a sort n as $\Psi \mid \Gamma \vdash_m^n S \leq P$ to range over both types and skeletons. When $n = \text{ty}$, the judgement is for types and metavariables S and P are types; when $n = \text{sk}$, the judgement is for skeletons.

Skeleton subtyping $\Psi \mid \Gamma \vdash_m^{\text{sk}} S \leq P$ requires well-formedness of S and P under Γ as well as $P :^{\text{sk}} m$ as well-formedness conditions. Intuitively, skeleton subtyping upcasts S to P guided by inherited information in P . Rule S-UNIVGHOSTR inserts a universal ghost

$\Psi \mid \Gamma \vdash_m^{\text{sk}} S \leq P$			
$\frac{\text{S-INFER} \quad \cdot \mid \Gamma \vdash S \sim P}{\cdot \mid \Gamma \vdash_I^{\text{sk}} S \leq P}$	$\frac{\text{S-UNIVGHOSTR} \quad S, P \text{ guarded} \quad \cdot \mid \Gamma \vdash S \sim P}{\cdot \mid \Gamma \vdash_c^{\text{sk}} S \leq \mathbb{A}.P}$	$\frac{\text{S-GHOST}}{\cdot \mid \Gamma \vdash_m^{\text{sk}} \mathbb{A} \leq P}$	$\frac{\text{S-GHOSTG}}{\cdot \mid \Gamma \vdash_m^{\text{sk}} \mathbb{A} \leq \mathbb{A}}$
$\frac{\text{S-ARROW} \quad \Psi_1 \mid \Gamma \vdash S_1 \sim P_1 \quad \Psi_2 \mid \Gamma \vdash_m^{\text{sk}} S_2 \leq P_2}{\Psi_1, \Psi_2 \mid \Gamma \vdash_{Fm}^{\text{sk}} S_1 \rightarrow S_2 \leq P_1 \rightarrow P_2}$		$\frac{\text{S-FORALLL} \quad m \neq I \quad \Psi' \text{ fresh} \quad \Gamma \vdash^{\text{sk}} \psi \# \Psi' \quad \Psi, \Psi' \mid \Gamma \vdash_m^{\text{sk}} S[\psi/\alpha] \leq P}{\Psi \mid \Gamma \vdash_m^{\text{sk}} \forall^\delta \alpha. S \leq P}$	
$\frac{\text{S-UNIVGHOSTL} \quad m \neq I \quad \Psi \mid \Gamma \vdash_m^{\text{sk}} S \leq P}{\Psi \mid \Gamma \vdash_m^{\text{sk}} \mathbb{A}.S \leq P}$		$\frac{\text{S-UNIVGHOSTLG} \quad m \neq I \quad \Psi \mid \Gamma \vdash_m^{\text{sk}} S \leq P}{\Psi \mid \Gamma \vdash_m^{\text{sk}} \mathbb{A}.S \leq P}$	
$\Psi \mid \Gamma \vdash S \sim P$			
$\frac{\text{C-UNIT}}{\cdot \mid \Gamma \vdash 1 \sim 1}$	$\frac{\text{C-VAR} \quad \Gamma \ni \alpha}{\cdot \mid \Gamma \vdash \alpha \sim \alpha}$	$\frac{\text{C-FORALLINH} \quad \Psi \mid \Gamma, \alpha \vdash S \sim P}{\Psi, \delta \mid \Gamma \vdash \forall^\delta \alpha. S \sim \forall \alpha. P}$	$\frac{\text{C-FORALLSYN} \quad \Psi \mid \Gamma, \alpha \vdash S \sim P}{\Psi \mid \Gamma \vdash \forall^\delta \alpha. S \sim \forall \alpha. P}$
$\frac{\text{C-ARROW} \quad \Psi_1 \mid \Gamma \vdash S_1 \sim P_1 \quad \Psi_2 \mid \Gamma \vdash S_2 \sim P_2}{\Psi_1, \Psi_2 \mid \Gamma \vdash S_1 \rightarrow S_2 \sim P_1 \rightarrow P_2}$		$\frac{\text{C-GHOST}}{\cdot \mid \Gamma \vdash \mathbb{A} \sim P}$	$\frac{\text{C-GHOSTG}}{\cdot \mid \Gamma \vdash \mathbb{A} \sim \mathbb{A}}$
$\frac{\text{C-UNIVGHOSTG} \quad \Psi \mid \Gamma \vdash S \sim P}{\Psi \mid \Gamma \vdash \mathbb{A}.S \sim \mathbb{A}.P}$	$\frac{\text{C-UNIVGHOST} \quad \Psi \mid \Gamma \vdash S \sim P}{\Psi \mid \Gamma \vdash \mathbb{A}.S \sim \mathbb{A}.P}$	$\frac{\text{C-UNIVGHOST1} \quad \Psi \mid \Gamma, \alpha \vdash \mathbb{A}.S \sim P}{\Psi \mid \Gamma \vdash \mathbb{A}.S \sim \forall \alpha. P}$	$\frac{\text{C-UNIVGHOST2} \quad \Psi \mid \Gamma \vdash S \sim P}{\Psi \mid \Gamma \vdash \mathbb{A}.S \sim P}$

Figure 8.11: Subtyping and consistency rules of skeletons for Fresco.

in checking mode, replacing S-CHECK and S-FORALLR in Figure 8.7. Rule S-UNIVGHOSTL instantiates a universal ghost. Like S-FORALLL, it requires the mode not to be inference mode. Rule S-GHOST upcasts a ghost to any inherited skeleton. There are two new rules for dotted ghosts. S-UNIVGHOSTLG is the same as S-UNIVGHOSTL. S-GHOSTG forces the RHS to be a ghost to avoid trivial guesses.

Skeleton consistency $\Psi \mid \Gamma \vdash S \sim P$ requires well-formedness of S and P under Γ as well as the refinement judgement $\Gamma \vdash [S] \blacktriangleright P$. Intuitively, skeleton consistency refines S to P guided by inherited information in P . The first five rules are the same as their type counterparts in Figure 8.7, except that they are lifted to skeletons. The remaining rules are for ghosts and dotted ghosts. Rule C-GHOST matches a ghost with any inherited skeleton, while rule C-GHOSTG matches a dotted ghost with a ghost. Rule C-UNIVGHOSTG matches a dotted universal ghost with a universal ghost.

Rules C-UNIVGHOST, C-UNIVGHOST1, and C-UNIVGHOST2 match a universal ghost to a universal ghost or an inherited quantifier, or remove a universal ghost.

As shown in rules S-GHOSTG, C-GHOSTG, and C-UNIVGHOSTG, a dotted ghost can only match a ghost. This avoids trivial guesses in S-FORALL. Otherwise, for example, we can always guess the trivial skeleton λ for instantiation, which always succeeds but provides no information.

9 Type inference algorithm

In fact “wrong” has no type.

—Robin Milner (1978)

We presented the declarative specification of Fresco in Chapter 8. It is non-algorithmic because of guesses: we guess monotypes arbitrarily and we guess decorated polytypes for instantiation in subtyping. In this chapter, we give an algorithmic type system for Fresco presented in the style of the type-inference-in-context approach [Gundry 2013; Gundry et al. 2010]. For guessing monotypes, we introduce flexible type variables and solve them with standard first-order unification. For guessing decorated polytypes, we introduce a novel look judgement which first collects local polymorphic constraints for a type variable and then constructs a polytype for instantiating the type variable by solving these constraints. This judgement is the most interesting machinery of the algorithm and is presented in Section 9.6.

We prove the algorithmic type system is sound and complete with respect to the declarative type system in Chapter 8. Our completeness result indicates that instead of understanding and simulating the involved details of the algorithm, programmers can reason about which programs type check by using the more succinct declarative specification. This is in contrast to many existing approaches to type inference for FCP in the literature which focus on developing advanced algorithms without declarative specifications or completeness results [Kovács 2020; Morris 2026; Serrano et al. 2020; Parreaux et al. 2024].

9.1 Algorithmic syntax

Figure 9.1 gives the extended syntax of types, skeletons, and algorithmic contexts for the algorithm. We extend the syntax of types and skeletons with *flexible* type variables $\hat{\alpha}$ which stand for unsolved monomorphic types. These variables are generated and manipulated by the algorithm and can be unified with other monomorphic types. In contrast, *rigid* type variables come from the object language and cannot be unified

with anything. Restricting flexible variables to monomorphic types is crucial to avoid guessing polymorphic types.

Types	$A, B ::= 1 \mid \alpha \mid A \rightarrow B \mid \forall \alpha. A \mid \hat{\alpha}$
Monotypes	$\tau, \pi ::= 1 \mid \alpha \mid \tau \rightarrow \pi \mid \hat{\alpha}$
Skeletons	$P, Q ::= 1 \mid \alpha \mid P \rightarrow Q \mid \forall \alpha. P \mid \hat{\alpha} \mid \lambda \mid \lambda. P$
Entries	$e ::= \alpha \mid x : P \mid \hat{\alpha} \mid \hat{\alpha} := \tau$
Algorithmic contexts	$\Theta ::= \cdot \mid \Theta, e$
Suffixes	$\Xi ::= \cdot \mid \Xi, \hat{\alpha} \mid \Xi, \hat{\alpha} := \tau$

Figure 9.1: Algorithmic syntax.

Algorithmic contexts Θ are ordered. Entries can only depend on those before them. Flexible variables in algorithmic contexts Θ are either declarations ($\hat{\alpha}$) or definitions ($\hat{\alpha} := \tau$). Suffixes Ξ are lists of declarations and definitions of flexible variables that can be moved around by algorithms. We define well-formedness of algorithmic contexts $\Theta \vdash \text{ctx}$ as follows.

$$\begin{array}{c}
 \frac{}{\cdot \vdash \text{ctx}} \quad \frac{\Theta \vdash \text{ctx}}{\Theta, \hat{\alpha} \vdash \text{ctx}} \quad \frac{\Theta \vdash \tau \quad \Theta \vdash \text{ctx}}{\Theta, \hat{\alpha} := \tau \vdash \text{ctx}} \quad \frac{\Theta \vdash P \quad \Theta \vdash \text{ctx}}{\Theta, x : P \vdash \text{ctx}} \quad \frac{\Theta \vdash \text{ctx}}{\Theta, \alpha \vdash \text{ctx}}
 \end{array}$$

9.1.1 Metasubstitutions

Following [Gundry et al. \[2010\]](#), we define metasubstitutions (or information increase) in Figure 9.2. A metasubstitution $\theta :: \Theta \sqsubseteq \Theta'$ is a finite map θ from flexible variables in Θ to well-formed monotypes in Θ' such that the structure and dependency order of Θ are preserved. Metasubstitutions solve and move flexible variables, thus representing an increase in monomorphic information. We write $\iota :: \Theta \sqsubseteq \Theta'$ for an identity metasubstitution and sometimes omit the name ι . For an identity metasubstitution, all entries of flexible variables in Θ are preserved in Θ' but possibly with more monomorphic information (e.g., from $\hat{\alpha}$ to $\hat{\alpha} := 1$). Our algorithms always give identity metasubstitutions. We also define equivalence of metasubstitutions $\theta \equiv \theta' :: \Theta \sqsubseteq \Theta'$ standardly in the same figure.

9.1.2 Declarative judgements in algorithmic contexts

Although all the declarative judgements in Chapter 8, including well-formedness, typing, skeleton inference, skeleton refinement, subtyping, and consistency, are defined on declarative contexts Γ , it is easy to extend them to algorithmic contexts Θ . For most rules we simply replace declarative contexts Γ with algorithmic contexts Θ . In

$$\boxed{\theta :: \Theta \sqsubseteq \Theta'}$$

$$\frac{}{\iota :: \cdot \sqsubseteq \Xi} \quad \frac{\theta :: \Theta \sqsubseteq \Theta' \quad \Theta' \vdash \tau}{(\theta, \tau/\hat{\alpha}) :: \Theta, \hat{\alpha} \sqsubseteq \Theta'} \quad \frac{\theta :: \Theta \sqsubseteq \Theta' \quad \Theta' \vdash \tau' \equiv \theta\tau}{(\theta, \tau'/\hat{\alpha}) :: \Theta, \hat{\alpha} := \tau \sqsubseteq \Theta'}$$

$$\frac{\theta :: \Theta \sqsubseteq \Theta'}{\theta :: \Theta, x : P \sqsubseteq \Theta', x : \theta P, \Xi} \quad \frac{\theta :: \Theta \sqsubseteq \Theta'}{\theta :: \Theta, \alpha \sqsubseteq \Theta', \alpha, \Xi}$$

$$\boxed{\theta \equiv \theta' :: \Theta \sqsubseteq \Theta'}$$

$$\frac{}{\iota \equiv \iota :: \cdot \sqsubseteq \Xi} \quad \frac{\theta_1 \equiv \theta_2 :: \Theta \sqsubseteq \Theta' \quad \Theta' \vdash \tau_1 \equiv \tau_2}{(\theta_1, \tau_1/\hat{\alpha}) \equiv (\theta_2, \tau_2/\hat{\alpha}) :: \Theta, \hat{\alpha} \sqsubseteq \Theta'}$$

$$\frac{\theta_1 \equiv \theta_2 :: \Theta \sqsubseteq \Theta' \quad \Theta' \vdash \tau_1 \equiv \tau_2 \quad \Theta' \vdash \tau_1 \equiv \theta_1\tau}{(\theta_1, \tau_1/\hat{\alpha}) \equiv (\theta_2, \tau_2/\hat{\alpha}) :: \Theta, \hat{\alpha} := \tau \sqsubseteq \Theta'} \quad \frac{\theta_1 \equiv \theta_2 :: \Theta \sqsubseteq \Theta'}{\theta_1 \equiv \theta_2 :: \Theta, x : P \sqsubseteq \Theta', x : \theta_1 P, \Xi}$$

$$\frac{\theta_1 \equiv \theta_2 :: \Theta \sqsubseteq \Theta'}{\theta_1 \equiv \theta_2 :: \Theta, \alpha \sqsubseteq \Theta', \alpha, \Xi}$$

Figure 9.2: Metasubstitutions and equivalence of metasubstitutions.

rules where we implicitly make use of α -equivalence of types or skeletons, we need to explicitly state the equivalence relation because of the existence of flexible variables. We only need to add new rules for the new cases of flexible variables, for which we always retrieve the definition of a flexible variable from the algorithmic context and continue with its definition.

Figure 9.3 gives the new and changed declarative rules in algorithmic contexts. For syntax-directed typing, we only need to extend freezing and annotated lambda rules with explicit type equivalence premises. For subtyping, we add a new rule S-FLEXMONO when the LHS is a flexible variable in function mode. For consistency, we add two new rules C-FLEXL and C-FLEXR when either the LHS or the RHS is a flexible variable. For refinement, we also add two rules dealing with flexible variables. We omit well-formedness rules, where we simply look up the declaration or definition of a flexible variable, and if it is a definition, we recursively check the well-formedness of this definition.

9.1.3 From colours to input skeletons

Our declarative judgements for typing $\Gamma \vdash_m M : Q$, skeleton inference $\Gamma \vdash_m M \Rightarrow Q$, subtyping $\Gamma \vdash_m^{\mathbf{n}} P \leq Q$, and consistency $\Gamma \vdash P \sim Q$ all use a bidirectional skeleton Q on the RHS, whose colours distinguish between inherited and synthesised information.

9 Type inference algorithm

$\Theta \vdash_m M : A$

syntax-directed typing

$$\frac{\text{SD-FREEZE} \quad \begin{array}{c} \Theta \vdash_I M : B \\ \Theta \vdash B \equiv A \end{array}}{\Theta \vdash_c [M] : A}$$

$$\frac{\text{SD-ABSANNOCHECK} \quad \begin{array}{c} \Theta, \bar{\alpha} \vdash A \equiv A' \\ \Theta, \bar{\alpha}, x : A \vdash_c M : B \end{array}}{\Theta \vdash_c \lambda x^A.M : \forall \bar{\alpha}. A' \rightarrow B}$$

$$\frac{\text{SD-ABSANNO} \quad \begin{array}{c} m \neq c \quad \Theta \vdash A \equiv A' \\ \Theta, x : A \vdash_m M : B \end{array}}{\Theta \vdash_m \lambda x^A.M : A' \rightarrow B}$$

$\Psi \mid \Theta \vdash_m^n S \leq Q$

subtyping

$$\frac{\text{S-FLEXMONO} \quad \begin{array}{c} \Theta \ni \hat{\alpha} := \tau \quad \Psi \mid \Theta \vdash_m^n \tau \leq Q \end{array}}{\Psi \mid \Theta \vdash_m^n \hat{\alpha} \leq Q}$$

$\Psi \mid \Theta \vdash S \sim Q$

consistency

$$\frac{\text{C-FLEXL} \quad \begin{array}{c} \Theta \ni \hat{\alpha} := \tau \quad \Psi \mid \Theta \vdash \tau \sim Q \end{array}}{\Psi \mid \Theta \vdash \hat{\alpha} \sim Q}$$

$$\frac{\text{C-FLEXR} \quad \begin{array}{c} \Theta \ni \hat{\alpha} := \tau \quad \Psi \mid \Theta \vdash S \sim \tau \end{array}}{\Psi \mid \Theta \vdash S \sim \hat{\alpha}}$$

$\Theta \mid \Delta \vdash P \blacktriangleright Q$

refinement

$$\frac{\Theta \ni \hat{\alpha} := \tau \quad \Theta \mid \Delta \vdash \tau \blacktriangleright Q}{\Theta \mid \Delta \vdash \hat{\alpha} \blacktriangleright Q}$$

$$\frac{\Theta \ni \hat{\alpha} := \tau \quad \Theta \mid \Delta \vdash P \blacktriangleright \tau}{\Theta \mid \Delta \vdash P \blacktriangleright \hat{\alpha}}$$

Figure 9.3: Declarative judgements in algorithmic contexts (showing only new and changed rules).

(In typing, we know Q is a concrete type. We use Q here for uniformity with the other three judgements.)

We do not have colours in our algorithmic judgements. Instead, for each judgement, we add an input skeleton P_0 to represent the inherited information. As we will explain in detail in the following sections, our algorithmic judgements for typing $\Theta [P_0] \vdash_m M \Rightarrow Q \dashv \Theta'$, skeleton inference $\Theta [P_0] \vdash_m M \Rightarrow Q \dashv \Theta'$, subtyping $\Theta [P_0] \vdash_m^n P \leq Q \dashv \Theta'$, and consistency $\Theta [P_0] \vdash P \sim Q \dashv \Theta'$ all have an input skeleton P_0 and an output skeleton Q . The synthesised information is the part of the output skeleton Q that is not in P_0 . This is formally characterised by the refinement judgement $P \blacktriangleright Q$ in Section 8.6. In the following sections, we use this judgement to establish the connection between the declarative and algorithmic judgements in the metatheory.

9.2 Algorithmic type inference

Figure 9.4 gives the type inference rules together with auxiliary skeleton splitting rules. The type inference rules correspond one-to-one with syntax-directed typing rules in Section 8.4. The type inference judgement has the form $\Theta \vdash_m M \Rightarrow A \dashv \Theta'$. The input includes a context Θ , a mode m , a skeleton P , and a term M ; the output includes a type A and a context Θ' . As usual, we require all inputs to be well-formed. In addition, we require a compatibility relation between modes and input skeletons as a well-formedness condition, playing the same role as the relation $A :^{ty} m$ in Section 8.3. The relation $P :^{ty} m$ is defined as follows.

$$\frac{}{\lambda :^{ty} I} \qquad \frac{}{A :^{ty} C} \qquad \frac{Q :^{ty} m}{P \rightarrow Q :^{ty} F m}$$

Rules I-UNIT, I-VAR, and I-ANNO follow the corresponding syntax-directed rules in Section 8.4. Each rule uses the algorithmic subtyping judgement which will be defined in Section 9.5. Rule I-FREEZE infers the type of M without using any inherited information and then checks the consistency between the inferred type and the inherited type A using the algorithmic consistency judgement which will be defined in Section 9.4.

Rule I-ABS infers the type of a lambda abstraction when not in checking mode, corresponding to the rule SD-ABS in Section 8.4. It uses the skeleton splitting judgement $\Theta \vdash P_0 \triangleright P \rightarrow Q \dashv \Theta'$ to split the input skeleton into a function shape. Here, Θ and P_0 are inputs while $P \rightarrow Q$ and Θ' are outputs. The last case is the most interesting one, where we split a flexible variable $\hat{a}$ into a function type with two fresh flexible variables $\hat{\beta}$ and $\hat{\gamma}$ as its domain and codomain. Rule I-ABS also uses the meta-function $\text{guess}(P) = (\bar{\hat{a}}; A)$ to get a type A by filling in each ghost in P with a fresh flexible variable in $\bar{\hat{a}}$ and removing universal ghosts in P . We formally define $\text{guess}(P)$ as follows.

$$\begin{aligned} \text{guess}(\lambda) &= (\hat{a}; \hat{a}) \quad \text{where } \hat{a} \text{ fresh} & \text{guess}(1) &= (\cdot; 1) \\ \text{guess}(\lambda.P) &= \text{guess}(P) & \text{guess}(\alpha) &= (\cdot; \alpha) \\ \text{guess}(\forall \alpha.P) &= (\bar{\hat{a}}; \forall \alpha.P') \quad \text{where } \text{guess}(P) = (\bar{\hat{a}}; P') & \text{guess}(\hat{a}) &= (\cdot; \hat{a}) \\ \text{guess}(P \rightarrow Q) &= (\bar{\hat{a}}_1, \bar{\hat{a}}_2; P' \rightarrow Q') \quad \text{where } \text{guess}(P) = (\bar{\hat{a}}_1; P') \text{ and } \text{guess}(Q) = (\bar{\hat{a}}_2; Q') \end{aligned}$$

Rule I-ABSCHECK deals with the checking mode, corresponding to the rule SD-ABSCHECK. Rules I-ABSANNO and I-ABSANNOCHECK deal with annotated lambda abstractions, corresponding to the rules SD-ABSANNO and SD-ABSANNOCHECK. Both rules check consistency between the annotated type of x and the argument type or skeleton inherited from the input skeleton.

Rule I-APP infers the type of an application, corresponding to the rule D-APP in Section 8.3. We first infer the skeleton P of the argument, then use it as the argument

$$\boxed{\Theta [P]_m M \Rightarrow A \vdash \Theta'}$$

$$\begin{array}{c}
 \text{I-UNIT} \\
 \frac{\Theta_0 [P]_m^{\text{ty}} 1 \leq A \vdash \Theta_1}{\Theta_0 [P]_m () \Rightarrow A \vdash \Theta_1} \\
 \\
 \text{I-ANNO} \\
 \frac{\Theta_0 [A]_c M \Rightarrow A \vdash \Theta_1 \quad \Theta_1 [P]_m^{\text{ty}} A \leq B \vdash \Theta_2}{\Theta_0 [P]_m (M : A) \Rightarrow B \vdash \Theta_2} \\
 \\
 \text{I-ABSCHECK} \\
 \frac{\Theta_0, \bar{\alpha} \vdash A_0 \triangleright A \rightarrow B \vdash \Theta_1, \bar{\alpha} \quad \Theta_1, \bar{\alpha}, x : A [B]_c M \Rightarrow B \vdash \Theta_2, \bar{\alpha}, x : A, \Xi}{\Theta_0 [\forall \bar{\alpha}. A_0]_c \lambda x. M \Rightarrow \forall \bar{\alpha}. A \rightarrow B \vdash \Theta_2} \\
 \\
 \text{I-ABSANNOCHECK} \\
 \frac{\Theta_0, \bar{\alpha} \vdash A_0 \triangleright A' \rightarrow B \vdash \Theta_1, \bar{\alpha} \quad \Theta_1, \bar{\alpha} \vdash A' \sim A \vdash \Theta_2, \bar{\alpha} \quad \Theta_2, \bar{\alpha}, x : A [B]_c M \Rightarrow B \vdash \Theta_3, \bar{\alpha}, x : A, \Xi}{\Theta_0 [\forall \bar{\alpha}. A_0]_c \lambda x^A. M \Rightarrow \forall \bar{\alpha}. A \rightarrow B \vdash \Theta_3} \\
 \\
 \text{I-APP} \\
 \frac{\Theta_0 [\hat{\lambda}]_c N \Rightarrow P \vdash \Theta_1 \quad \Theta_1 [P \rightarrow Q]_{fm} M \Rightarrow A \rightarrow B \vdash \Theta_2 \quad \Theta_2 [A]_c N \Rightarrow A' \vdash \Theta_3}{\Theta_0 [Q]_m M N \Rightarrow B \vdash \Theta_3} \\
 \\
 \text{I-VAR} \\
 \frac{\Theta_0 \ni x : A \quad \Theta_0 [P]_m^{\text{ty}} A \leq B \vdash \Theta_1}{\Theta_0 [P]_m x \Rightarrow B \vdash \Theta_1} \\
 \\
 \text{I-FREEZE} \\
 \frac{\Theta_0 [\hat{\lambda}]_c M \Rightarrow B \vdash \Theta_1 \quad \Theta_1 \vdash B \sim A \vdash \Theta_2}{\Theta_0 [A]_c [M] \Rightarrow A \vdash \Theta_2} \\
 \\
 \text{I-ABS} \\
 \frac{m \neq c \quad \Theta_0 \vdash P_0 \triangleright P \rightarrow Q \vdash \Theta_1 \quad \text{guess}(P) = (\hat{\alpha}; A) \quad \Theta_1, \hat{\alpha}, x : A [Q]_{m-} M \Rightarrow B \vdash \Theta_2, x : A, \Xi}{\Theta_0 [P_0]_m \lambda x. M \Rightarrow A \rightarrow B \vdash \Theta_2, \Xi} \\
 \\
 \text{I-ABSANNO} \\
 \frac{m \neq c \quad \Theta_0 \vdash P_0 \triangleright P \rightarrow Q \vdash \Theta_1 \quad \Theta_1 \vdash A \sim P \vdash \Theta_2 \quad \Theta_2, x : A [Q]_{m-} M \Rightarrow B \vdash \Theta_3, x : A, \Xi}{\Theta_0 [P_0]_m \lambda x^A. M \Rightarrow A \rightarrow B \vdash \Theta_3, \Xi}
 \end{array}$$

$$\boxed{\Theta \vdash P_0 \triangleright P \rightarrow Q \vdash \Theta'}$$

$$\frac{}{\Theta_0 \vdash P \rightarrow Q \triangleright P \rightarrow Q \vdash \Theta_0} \quad \frac{}{\Theta_0 \vdash \hat{\lambda} \triangleright \hat{\lambda} \rightarrow \hat{\lambda} \vdash \Theta_0} \quad \frac{\Theta_0, \hat{\beta}, \hat{\gamma} \vdash \hat{\alpha} \sim \hat{\beta} \rightarrow \hat{\gamma} \vdash \Theta_1}{\Theta_0 \vdash \hat{\alpha} \triangleright \hat{\beta} \rightarrow \hat{\gamma} \vdash \Theta_1}$$

Figure 9.4: Algorithmic type inference and skeleton splitting.

skeleton to infer the type $A \rightarrow B$ of the function, and finally check the argument against type A .

It is worth noting that although we have two passes for the argument in the rule I-APP, the time complexity of the algorithm is still polynomial in the size of the input term. This is because the first pass is just skeleton inference, in which we only traverse the argument once as we will see in Section 9.3. As a result, even in the extreme case of n nested applications $M (N_1 \cdots (N_{n-1} N_n) \cdots)$, the total number of traversals of the innermost argument N_n is $n + 1$ instead of 2^n .

We now state the metatheory of type inference. Following Gundry et al. [2010], we first define the notion of problems and solutions of type inference, and then state soundness, completeness, and optimality. Our proofs can be found in Appendix D.2.

Definition 9 (Problems and solutions of type inference). *A problem of type inference is a tuple $(\Theta_0; m; P; M)$ where Θ_0 is well-formed, P and M are well-formed in Θ_0 , and $P :^{\text{ty}} m$. A solution to it is a tuple $(\theta_1 :: \Theta_0 \sqsubseteq \Theta_1; A)$ such that $\Theta_1 \vdash_m M : A$ where $\Theta_1 \vdash \theta_1 P \blacktriangleright A$. Such a solution is optimal if for any other solution $(\theta :: \Theta_0 \sqsubseteq \Theta; B)$, there exists a metasubstitution $\zeta :: \Theta_1 \sqsubseteq \Theta$ such that $\theta \equiv \zeta \circ \theta_1 :: \Theta_0 \sqsubseteq \Theta$ (θ factors through θ_1 with cofactor ζ) and $\Theta \vdash \zeta A \equiv B$.*

Theorem 21 (Soundness of type inference). *Given a type inference problem $(\Theta_0; m; P; M)$, if $\Theta_0 [P]_m M \Rightarrow A \dashv \Theta_1$, then $(\Theta_0 \sqsubseteq \Theta_1; A)$ is a solution.*

Theorem 22 (Completeness and optimality of type inference). *Given a type inference problem $(\Theta_0; m; P; M)$, if there exists a solution $(\theta :: \Theta_0 \sqsubseteq \Theta; B)$, then $\Theta_0 [P]_m M \Rightarrow A \dashv \Theta_1$ gives its optimal solution $(\Theta_0 \sqsubseteq \Theta_1; A)$.*

9.3 Algorithmic skeleton inference

Figure 9.5 gives the skeleton inference rules, which correspond one-to-one with the declarative skeleton inference rules in Section 8.7. The judgement has the form $\Theta [P_0]_m M \Rightarrow P \dashv \Theta'$. The input includes a context Θ , a mode m , a skeleton P_0 , and a term M ; the output includes a skeleton P and a context Θ' . As before, we require $P_0 :^{\text{sk}} m$ as a well-formedness condition defined as follows.

$$\frac{}{\lambda :^{\text{sk}} \text{I}} \qquad \frac{}{\lambda :^{\text{sk}} \text{C}} \qquad \frac{Q :^{\text{sk}} m}{P \rightarrow Q :^{\text{sk}} \text{F } m}$$

Algorithmic skeleton inference rules closely mimic algorithmic type inference rules in Section 9.2. Rule **PI-ABS**, unlike rule **I-ABS**, does not guess a monomorphic type from P but directly puts $x : P$ into the context. Introducing flexible variables for ghosts here would be problematic because these ghosts may be matched with inherited polytypes later, as in our motivating example at the beginning of Chapter 8. Rule **PI-ABSCHECK** introduces a universal ghost for the function skeleton, similar to **P-ABSCHECK** in Section 8.7. Rule **PI-APP** does not do type checking for the argument N , similar to **P-APP**.

Problems and solutions of skeleton inference, as well as soundness, completeness, and optimality theorems, are defined similarly to those of type inference in Section 9.2. The main difference is that to characterise the optimality of skeleton inference, we need to formally define when a skeleton is more precise than another skeleton. We write $\Theta \vdash P \leq Q$, read as P is more precise than Q , if P contains exactly all quantifiers in Q , but may have more universal ghosts, and fewer monomorphic types. This is exactly

$$\boxed{\Theta [P_0]_{\mathfrak{m}} M \Rightarrow P \dashv \Theta'}$$

$$\begin{array}{c}
 \text{PI-UNIT} \\
 \frac{\Theta_0 [P_0]_{\mathfrak{m}}^{\text{sk}} 1 \leq P \dashv \Theta_1}{\Theta_0 [P_0]_{\mathfrak{m}} () \Rightarrow P \dashv \Theta_1} \\
 \\
 \text{PI-ANNO} \\
 \frac{\Theta_0 [P]_{\mathfrak{m}}^{\text{sk}} A \leq Q \dashv \Theta_1}{\Theta_0 [P]_{\mathfrak{m}} (M : A) \Rightarrow Q \dashv \Theta_1} \\
 \\
 \text{PI-ABSCHECK} \\
 \frac{\Theta_0, x : \mathbb{A} [\mathbb{A}]_{\mathfrak{c}} M \Rightarrow Q \dashv \Theta_1, x : \mathbb{A}, \Xi}{\Theta_0 [\mathbb{A}]_{\mathfrak{c}} \lambda x.M \Rightarrow \mathbb{A}.\mathbb{A} \rightarrow Q \dashv \Theta_1, \Xi} \\
 \\
 \text{PI-ABSANNOCHECK} \\
 \frac{\Theta_0, x : A [\mathbb{A}]_{\mathfrak{c}} M \Rightarrow Q \dashv \Theta_1, x : A, \Xi}{\Theta_0 [\mathbb{A}]_{\mathfrak{c}} \lambda x^A.M \Rightarrow \mathbb{A}.A \rightarrow Q \dashv \Theta_1, \Xi} \\
 \\
 \text{PI-APP} \\
 \frac{\Theta_0 [\mathbb{A}]_{\mathfrak{c}} N \Rightarrow P_0 \dashv \Theta_1 \quad \Theta_1 [P_0 \rightarrow Q_0]_{\mathfrak{f}\mathfrak{m}} M \Rightarrow P \rightarrow Q \dashv \Theta_2}{\Theta_0 [Q_0]_{\mathfrak{m}} M N \Rightarrow Q \dashv \Theta_2}
 \end{array}$$

$$\begin{array}{c}
 \text{PI-VAR} \\
 \frac{\Theta_0 \ni x : P \quad \Theta_0 [P_0]_{\mathfrak{m}}^{\text{sk}} P \leq Q \dashv \Theta_1}{\Theta_0 [P_0]_{\mathfrak{m}} x \Rightarrow Q \dashv \Theta_1} \\
 \\
 \text{PI-FREEZE} \\
 \frac{\Theta_0 [\mathbb{A}]_{\mathfrak{I}} M \Rightarrow P \dashv \Theta_1}{\Theta_0 [\mathbb{A}]_{\mathfrak{c}} [M] \Rightarrow P \dashv \Theta_1} \\
 \\
 \text{PI-ABS} \\
 \frac{\mathfrak{m} \neq \mathfrak{c} \quad \Theta_0 \vdash P_0 \triangleright P \rightarrow Q_0 \dashv \Theta_1 \quad \Theta_1, x : P [Q_0]_{\mathfrak{m}} M \Rightarrow Q \dashv \Theta_2, x : P, \Xi}{\Theta_0 [P_0]_{\mathfrak{m}} \lambda x.M \Rightarrow P \rightarrow Q \dashv \Theta_2, \Xi} \\
 \\
 \text{PI-ABSANNO} \\
 \frac{\mathfrak{m} \neq \mathfrak{c} \quad \Theta_0 \vdash P_0 \triangleright P \rightarrow Q_0 \dashv \Theta_1 \quad \Theta_1 \vdash A \sim P \dashv \Theta_2 \quad \Theta_2, x : A [Q_0]_{\mathfrak{m}} M \Rightarrow Q \dashv \Theta_3, x : A, \Xi}{\Theta_0 [P_0]_{\mathfrak{m}} \lambda x^A.M \Rightarrow A \rightarrow Q \dashv \Theta_3, \Xi}
 \end{array}$$

Figure 9.5: Algorithmic skeleton inference.

what $\Theta \vdash P \blacktriangleright Q$ means. Since Q is inherited, the refinement relation cannot fill P with more quantifiers, but can only fill its ghosts with monotypes and remove universal ghosts.¹ We therefore directly define $\Theta \vdash P \leq Q$ as $\Theta \vdash P \blacktriangleright Q$. On types, precision coincides with type equivalence. Our optimality theorem guarantees that algorithmic skeleton inference always infers the most precise skeleton. We state the metatheory of skeleton inference below. The proofs can be found in Appendix D.3.

Definition 10 (Problems and solutions of skeleton inference). *A problem of skeleton inference is a tuple $(\Theta_0; \mathfrak{m}; P_0; M)$ where Θ_0 is well-formed, P_0 and M are well-formed in Θ_0 , and $P_0 :^{\text{sk}} \mathfrak{m}$. A solution to it is a tuple $(\theta_1 :: \Theta_0 \sqsubseteq \Theta_1; P_1)$ such that $\Theta_1 \vdash_{\mathfrak{m}} M \Rightarrow P_1$ where $\Theta_1 \vdash \theta_1 P_0 \blacktriangleright P_1$. Such a solution is optimal if for any other solution $(\theta :: \Theta_0 \sqsubseteq \Theta; Q)$, there exists a metasubstitution $\zeta :: \Theta_1 \sqsubseteq \Theta$ such that $\theta \equiv \zeta \circ \theta_1 :: \Theta_0 \sqsubseteq \Theta$ and $\Theta \vdash \zeta P_1 \leq Q$.*

Theorem 23 (Soundness of skeleton inference). *Given a skeleton inference problem $(\Theta_0; \mathfrak{m}; P_0; M)$, if $\Theta_0 [P_0]_{\mathfrak{m}} M \Rightarrow P \dashv \Theta_1$, then $(\Theta_0 \sqsubseteq \Theta_1; P)$ is a solution.*

¹It may appear intriguing that we say P is more precise than Q but actually Q can have more monotypes than P . We use the word “precision” not in terms of the quantity of information, but in terms of how a skeleton characterises terms with fewer unnecessary restrictions.

Theorem 24 (Completeness and optimality of skeleton inference). *Given a skeleton inference problem $(\Theta_0; m; P_0; M)$, if there exists a solution $(\theta :: \Theta_0 \sqsubseteq \Theta; Q)$, then $\Theta_0 [P_0] \vdash_m M \Rightarrow P \dashv \Theta_1$ gives its optimal solution $(\Theta_0 \sqsubseteq \Theta_1; P)$.*

9.4 Algorithmic consistency

We first introduce algorithmic consistency before algorithmic subtyping. Figure 9.6 gives the algorithmic consistency rules, together with the auxiliary variable unification and assignment rules. The judgement has the form $\Theta_0 [P_0] \vdash P \sim Q \dashv \Theta_1$. The input includes a context Θ_0 , a skeleton P_0 , and a skeleton P ; the output includes a skeleton Q and a context Θ_1 . This judgement refines P to Q guided by P_0 , which itself refines to Q . Intuitively, Q can be understood as the join of P and P_0 , viewing the refinement relation as a partial order. When both P and P_0 are concrete types, this judgement degenerates to the standard unification of P and P_0 in type inference.

Refinement disallows uses of bound variables in ghosts. Thus, to enforce that Q is a valid refinement of P and P_0 , we augment the judgement with two input variable sets Δ_1 and Δ_2 to track bound variables in P_0 and P , respectively. The full judgement has the form $\Theta_0 [P_0] \vdash_{\Delta_1|\Delta_2} P \sim Q \dashv \Theta_1$. Rules U-FORALLINH, U-FORALLSYN, and U-UNIVGHOST1 add bound variables to Δ_1 or Δ_2 . The names of rules show their correspondence to declarative consistency rules in Section 8.8. Rules U-GHOST and U-GHOSTR restrict the uses of variables from Δ_1 or Δ_2 in the output skeleton, guaranteeing the output skeleton is a valid refinement of both input skeletons.

Compared to declarative consistency, algorithmic consistency has three new rules dealing with cases where either or both sides are flexible variables: U-FLEX, U-FLEXL, and U-FLEXR. They use auxiliary judgements to look up definitions of flexible variables and move other dependencies as necessary. These auxiliary judgements are standard [Gundry et al. 2010] and are given in the continuation of Figure 9.6. There are also two new rules to deal with ghosts in the input skeletons: U-GHOSTR and U-UNIVGHOSTR. Declarative consistency does not have these two rules because it has no explicit input skeletons.

We write $\Theta_0 \vdash P \sim P_0 \dashv \Theta_1$ for $\Theta_0 [P_0] \vdash P \sim Q \dashv \Theta_1$ when the output Q is not used. When either P or P_0 is a type A , the output Q is a type equivalent to A .

The auxiliary judgement $\Theta \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta'$ deals with the case of unifying two flexible variables by looking up their definitions. The judgement $\Theta \mid \Xi \vdash \hat{\alpha} := P \dashv \Theta'$ deals with the case of assigning a flexible variable $\hat{\alpha}$ to a skeleton P . It uses the context Ξ to contain flexible variables that occur in P and thus need to be moved before $\hat{\alpha}$. Among these rules, rule U-ASSIGN-UNIVGHOST-ID deals with the special corner case where P is

$\hat{\alpha}$, which is not covered by rule U-ASSIGN-SOLVE_M due to its side condition for the occurrence check $\hat{\alpha} \notin \text{ftv}(P)$.

We state the metatheory of consistency below. The proofs of soundness and completeness of consistency are in Appendices D.5.2 and D.5.3.

Definition 11 (Problems and solutions of consistency).

1. A problem of consistency is a tuple $(\Theta_0; P_0; P)$ where P_0 and P are well-formed in Θ_0 . A solution to it is a tuple $(\theta_1 :: \Theta_0 \sqsubseteq \Theta_1; P_1)$ such that $\Theta_1 \vdash \theta_1 P \sim P_1$ where $\Theta_1 \vdash \theta_1 P_0 \triangleright P_1$. The solution is optimal if for any other solution $(\theta :: \Theta_0 \sqsubseteq \Theta; Q)$, there exists a metasubstitution $\zeta :: \Theta_1 \sqsubseteq \Theta$ such that $\theta \equiv \zeta \circ \theta_1 :: \Theta_0 \sqsubseteq \Theta$ and $\Theta \vdash \zeta P_1 \leq Q$.
2. A problem of unification of flexible variables is a tuple $(\Theta_0; \hat{\alpha}; \hat{\beta})$ where $\hat{\alpha} \in \text{dom}(\Theta_0)$ and $\hat{\beta} \in \text{dom}(\Theta_0)$. A solution to it is a metasubstitution $\theta_1 :: \Theta_0 \sqsubseteq \Theta_1$ such that $\Theta_1 \vdash \theta_1 \hat{\alpha} \sim \theta_1 \hat{\beta}$. The solution is optimal if for any other solution $\theta :: \Theta_0 \sqsubseteq \Theta$, there exists a metasubstitution $\zeta :: \Theta_1 \sqsubseteq \Theta$ such that $\theta \equiv \zeta \circ \theta_1 :: \Theta_0 \sqsubseteq \Theta$.
3. A problem of assignment of a flexible variable is a tuple $(\Theta_0; \Xi; \hat{\alpha}; P)$ where $\hat{\alpha} \in \text{dom}(\Theta_0)$ and P is well-formed in Θ_0, Ξ and P is not a flexible variable and Ξ only contains declarations of flexible variables appearing in P . A solution to it is a metasubstitution $\theta_1 :: \Theta_0, \Xi \sqsubseteq \Theta_1$ such that $\Theta_1 \vdash \theta_1 P \sim \theta_1 \hat{\alpha}$. The solution is optimal if for any other solution $\theta :: \Theta_0, \Xi \sqsubseteq \Theta$, there exists a metasubstitution $\zeta :: \Theta_1 \sqsubseteq \Theta$ such that $\theta \equiv \zeta \circ \theta_1 :: \Theta_0, \Xi \sqsubseteq \Theta$.

Lemma 25 (Soundness of consistency).

1. Given a problem of consistency $(\Theta_0; P_0; P)$, if $\Theta_0 \vdash_{P_0} P \sim P_1 \dashv \Theta_1$, then $(\Theta_0 \sqsubseteq \Theta_1; P_1)$ is a solution to it.
2. Given a problem of variable unification $(\Theta_0; \hat{\alpha}; \hat{\beta})$, if $\Theta_0 \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta_1$, then $\Theta_0 \sqsubseteq \Theta_1$ is a solution to it.
3. Given a problem of assignment $(\Theta_0; \Xi; \hat{\alpha}; P)$, if $\Theta_0 \mid \Xi \vdash \hat{\alpha} := P \dashv \Theta_1$, then $\Theta_0, \Xi \sqsubseteq \Theta_1$ is a solution to it.

Lemma 26 (Completeness and optimality of consistency).

1. Given a problem of consistency $(\Theta_0; P_0; P)$, if there exists a solution to it, then $\Theta_0 \vdash_{P_0} P \sim P_1 \dashv \Theta_1$ gives its optimal solution $(\Theta_0 \sqsubseteq \Theta_1; P_1)$.
2. Given a problem of variable unification $(\Theta_0; \hat{\alpha}; \hat{\beta})$, if there exists a solution to it, then $\Theta_0 \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta_1$ gives its optimal solution $\Theta_0 \sqsubseteq \Theta_1$.

3. Given a problem of assignment $(\Theta_0; \Xi; \hat{\alpha}; P)$, if there exists a solution to it, then $\Theta_0 \mid \Xi \vdash \hat{\alpha} := P + \Theta_1$ gives its optimal solution $\Theta_0, \Xi \sqsubseteq \Theta_1$.

$\Theta_0 [P_0]_{\Delta_1 \Delta_2} P \sim Q + \Theta_1$		
$\frac{\text{U-GHOST} \quad \Delta_2 \cap \text{ftv}(P_0) = \emptyset}{\Theta_0 [P_0]_{\Delta_1 \Delta_2} \hat{\alpha} \sim P_0 + \Theta_0}$	$\frac{\text{U-UNIT}}{\Theta_0 [1]_{\Delta_1 \Delta_2} 1 \sim 1 + \Theta_0}$	$\frac{\text{U-RIGID} \quad \Theta_0 \ni \alpha}{\Theta_0 [\alpha]_{\Delta_1 \Delta_2} \alpha \sim \alpha + \Theta_0}$
$\frac{\text{U-ARROW} \quad \Theta_0 [P'_1]_{\Delta_1 \Delta_2} P_1 \sim Q_1 + \Theta_1 \quad \Theta_1 [P'_2]_{\Delta_1 \Delta_2} P_2 \sim Q_2 + \Theta_2}{\Theta_0 [P'_1 \rightarrow P'_2]_{\Delta_1 \Delta_2} P_1 \rightarrow P_2 \sim Q_1 \rightarrow Q_2 + \Theta_2}$		
$\frac{\text{U-FORALLINH} \quad \Theta_0, \alpha [P_0]_{\Delta_1, \alpha \Delta_2, \alpha} P \sim Q + \Theta_1, \alpha}{\Theta_0 [\forall \alpha. P_0]_{\Delta_1 \Delta_2} \forall \alpha. P \sim \forall \alpha. Q + \Theta_1}$	$\frac{\text{U-FORALLSYN} \quad \Theta_0, \alpha [\hat{\alpha}. P_0]_{\Delta_1 \Delta_2, \alpha} P \sim Q + \Theta_1, \alpha}{\Theta_0 [\hat{\alpha}. P_0]_{\Delta_1 \Delta_2} \forall \alpha. P \sim \forall \alpha. Q + \Theta_1}$	
$\frac{\text{U-UNIVGHOST} \quad \Theta_0 [P_0]_{\Delta_1 \Delta_2} P \sim Q + \Theta_1}{\Theta_0 [\hat{\alpha}. P_0]_{\Delta_1 \Delta_2} \hat{\alpha}. P \sim \hat{\alpha}. Q + \Theta_1}$	$\frac{\text{U-UNIVGHOST1} \quad \Theta_0, \alpha [P_0]_{\Delta_1, \alpha \Delta_2} \hat{\alpha}. P \sim Q + \Theta_1, \alpha}{\Theta_0 [\forall \alpha. P_0]_{\Delta_1 \Delta_2} \hat{\alpha}. P \sim \forall \alpha. Q + \Theta_1}$	
$\frac{\text{U-UNIVGHOST2} \quad \Theta_0 \vdash P_0 \text{ guarded} \quad P_0 \text{ non-flex} \quad \Theta_0 [P_0]_{\Delta_1 \Delta_2} P \sim Q + \Theta_1}{\Theta_0 [P_0]_{\Delta_1 \Delta_2} \hat{\alpha}. P \sim Q + \Theta_1}$		
$\frac{\text{U-FLEX} \quad \Theta_0 \vdash \hat{\alpha} \simeq \hat{\beta} + \Theta_1}{\Theta_0 [\hat{\beta}]_{\Delta_1 \Delta_2} \hat{\alpha} \sim \hat{\beta} + \Theta_1}$	$\frac{\text{U-FLEXL} \quad P_0 \text{ non-flex} \quad \Theta_0 \mid \cdot \vdash \hat{\alpha} := P_0 + \Theta_1}{\Theta_0 [P_0]_{\Delta_1 \Delta_2} \hat{\alpha} \sim \hat{\alpha} + \Theta_1}$	$\frac{\text{U-FLEXR} \quad P \text{ non-flex} \quad \Theta_0 \mid \cdot \vdash \hat{\alpha} := P + \Theta_1}{\Theta_0 [\hat{\alpha}]_{\Delta_1 \Delta_2} P \sim \hat{\alpha} + \Theta_1}$
$\frac{\text{U-GHOSTR} \quad \Delta_1 \cap \text{ftv}(P) = \emptyset}{\Theta_0 [\hat{\alpha}]_{\Delta_1 \Delta_2} P \sim P + \Theta_0}$	$\frac{\text{U-UNIVGHOSTR} \quad \Theta_0 \vdash P \text{ guarded} \quad P \text{ non-flex} \quad \Theta_0 [P_0]_{\Delta_1 \Delta_2} P \sim Q + \Theta_1}{\Theta_0 [\hat{\alpha}. P_0]_{\Delta_1 \Delta_2} P \sim Q + \Theta_1}$	

Figure 9.6: Algorithmic consistency.

$$\boxed{\Theta \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta'}$$

$$\begin{array}{c}
 \text{U-FLEX-FLEX-ID} \\
 \hat{\alpha} \in \text{dom}(\Theta_0) \\
 \hline
 \Theta_0 \vdash \hat{\alpha} \simeq \hat{\alpha} \dashv \Theta_0
 \end{array}
 \quad
 \begin{array}{c}
 \text{U-FLEX-FLEX-L} \\
 \hat{\alpha} \neq \hat{\beta} \\
 \hline
 \Theta_0, \hat{\alpha} \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta_0, \hat{\alpha} := \hat{\beta}
 \end{array}
 \quad
 \begin{array}{c}
 \text{U-FLEX-FLEX-R} \\
 \hat{\alpha} \neq \hat{\beta} \\
 \hline
 \Theta_0, \hat{\beta} \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta_0, \hat{\beta} := \hat{\alpha}
 \end{array}$$

$$\begin{array}{c}
 \text{U-FLEX-FLEX-ASSIGN} \\
 \Theta_0 \vdash \hat{\alpha}[\tau/\hat{\gamma}] \sim \hat{\beta}[\tau/\hat{\gamma}] \dashv \Theta_1 \\
 \hline
 \Theta_0, \hat{\gamma} := \tau \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta_1, \hat{\gamma} := \tau
 \end{array}
 \quad
 \begin{array}{c}
 \text{U-FLEX-FLEX-SKIP} \\
 \hat{\alpha} \neq \hat{\gamma} \quad \hat{\beta} \neq \hat{\gamma} \quad \Theta_0 \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta_1 \\
 \hline
 \Theta_0, \hat{\gamma} \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta_1, \hat{\gamma}
 \end{array}$$

$$\begin{array}{c}
 \text{U-FLEX-FLEX-SKIP'} \\
 e = \gamma \text{ or } x : P \quad \Theta_0 \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta_1 \\
 \hline
 \Theta_0, e \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta_1, e
 \end{array}$$

$$\boxed{\Theta \mid \Xi \vdash \hat{\alpha} := P \dashv \Theta'}$$

$$\begin{array}{c}
 \text{U-ASSIGN-UNIVGHOST-ID} \\
 \hline
 \Theta_0, \hat{\alpha} \mid \cdot \vdash \hat{\alpha} := \hat{\alpha} \dashv \Theta_0, \hat{\alpha}
 \end{array}
 \quad
 \begin{array}{c}
 \text{U-ASSIGN-SOLVEM} \\
 \hat{\alpha} \notin \text{ftv}(P) \quad \text{guess}(P) = (\bar{\hat{\beta}}; \tau) \\
 \hline
 \Theta_0, \hat{\alpha} \mid \Xi \vdash \hat{\alpha} := P \dashv \Theta_0, \Xi, \bar{\hat{\beta}}, \hat{\alpha} := \tau
 \end{array}$$

$$\begin{array}{c}
 \text{U-ASSIGN-ASSIGN} \\
 \Theta_0, \Xi \vdash \hat{\alpha}[\tau/\hat{\beta}] \sim P[\tau/\hat{\beta}] \dashv \Theta_1 \\
 \hline
 \Theta_0, \hat{\beta} := \tau \mid \Xi \vdash \hat{\alpha} := P \dashv \Theta_1, \hat{\beta} := \tau
 \end{array}
 \quad
 \begin{array}{c}
 \text{U-ASSIGN-DEPEND} \\
 \hat{\alpha} \neq \hat{\beta} \quad \hat{\beta} \in \text{ftv}(P) \quad \Theta_0 \mid \hat{\beta}, \Xi \vdash \hat{\alpha} := P \dashv \Theta_1 \\
 \hline
 \Theta_0, \hat{\beta} \mid \Xi \vdash \hat{\alpha} := P \dashv \Theta_1
 \end{array}$$

$$\begin{array}{c}
 \text{U-ASSIGN-SKIP} \\
 \hat{\alpha} \neq \hat{\beta} \quad \hat{\beta} \notin \text{ftv}(P) \\
 \Theta_0 \mid \Xi \vdash \hat{\alpha} := P \dashv \Theta_1 \\
 \hline
 \Theta_0, \hat{\beta} \mid \Xi \vdash \hat{\alpha} := P \dashv \Theta_1, \hat{\beta}
 \end{array}
 \quad
 \begin{array}{c}
 \text{U-ASSIGN-SKIPRIGID} \\
 \beta \notin \text{ftv}(P) \quad \Theta_0 \mid \Xi \vdash \hat{\alpha} := P \dashv \Theta_1 \\
 \hline
 \Theta_0, \beta \mid \Xi \vdash \hat{\alpha} := P \dashv \Theta_1, \beta
 \end{array}$$

$$\begin{array}{c}
 \text{U-ASSIGN-SKIPTMVAR} \\
 \Theta_0 \mid \Xi \vdash \hat{\alpha} := P \dashv \Theta_1 \\
 \hline
 \Theta_0, x : Q \mid \Xi \vdash \hat{\alpha} := P \dashv \Theta_1, x : Q
 \end{array}$$

Figure 9.6: Algorithmic consistency (continued).

9.5 Algorithmic subtyping

Figure 9.7 gives the algorithmic subtyping rules. The algorithmic subtyping judgement has the form $\Theta \ [P_0]_{\mathfrak{m}}^n P \leq Q \dashv \Theta'$. The input includes a context Θ , a sort n , a mode $\mathfrak{m}$, a skeleton P_0 , and a skeleton P ; the output includes a skeleton Q and a context Θ' . This judgement upcasts P to Q guided by the input skeleton P_0 , which itself refines to Q . In addition to well-formedness of all inputs, we require $P_0 :^n \mathfrak{m}$ as a well-formedness condition of the judgement, as for algorithmic type inference (Section 9.2) and algorithmic skeleton inference (Section 9.3).

$$\boxed{\Theta [P_0]_{\mathfrak{m}}^n P \leq Q \vdash \Theta'}$$

$$\begin{array}{c}
\text{SI-INFER} \\
\hline
\Theta_0 [\mathbb{A}]_{\mathfrak{I}}^n P \leq P \vdash \Theta_0
\end{array}
\qquad
\begin{array}{c}
\text{SI-CHECK} \\
A, B \text{ guarded} \quad \Theta_0 [B] \vdash A \sim B \vdash \Theta_1 \\
\hline
\Theta_0 [B]_{\mathfrak{c}}^{\text{ty}} A \leq B \vdash \Theta_1
\end{array}$$

$$\begin{array}{c}
\text{SI-FORALLR} \\
\Theta_0, \alpha [B]_{\mathfrak{c}}^{\text{ty}} A \leq B \vdash \Theta_1, \alpha, \Xi \\
\hline
\Theta_0 [\forall \alpha. B]_{\mathfrak{c}}^{\text{ty}} A \leq \forall \alpha. B \vdash \Theta_1
\end{array}
\qquad
\begin{array}{c}
\text{SI-UNIVGHOSTR} \\
P \text{ guarded} \\
\hline
\Theta_0 [\mathbb{A}]_{\mathfrak{c}}^{\text{sk}} P \leq \mathbb{A}.P \vdash \Theta_0
\end{array}
\qquad
\begin{array}{c}
\text{SI-FLEXMONO} \\
\Theta_0 \mid \cdot [P_0]_{\mathfrak{Fm}}^n \hat{\alpha} \leq Q \vdash \Theta_1 \\
\hline
\Theta_0 [P_0]_{\mathfrak{Fm}}^n \hat{\alpha} \leq Q \vdash \Theta_1
\end{array}$$

$$\begin{array}{c}
\text{SI-ARROW} \\
\Theta_0 [P'_1] \vdash P_1 \sim Q_1 \vdash \Theta_1 \quad \Theta_1 [P'_2]_{\mathfrak{m}}^n P_2 \leq Q_2 \vdash \Theta_2 \\
\hline
\Theta_0 [P'_1 \rightarrow P'_2]_{\mathfrak{Fm}}^n P_1 \rightarrow P_2 \leq Q_1 \rightarrow Q_2 \vdash \Theta_2
\end{array}$$

$$\begin{array}{c}
\text{SI-FORALLL} \\
\mathfrak{m} \neq \mathfrak{I} \quad P_0 \text{ guarded or } P_0 = \mathbb{A} \quad P \leq_{\mathfrak{m}}^n P_0 \not\vdash^{\alpha} \Xi; P_1 \quad \Theta_0, \Xi \vdash P_1 \\
\Theta_0, \Xi [P_0]_{\mathfrak{m}}^n P[P_1/\alpha] \leq Q \vdash \Theta_1 \\
\hline
\Theta_0 [P_0]_{\mathfrak{m}}^n \forall \alpha. P \leq Q \vdash \Theta_1
\end{array}$$

$$\begin{array}{c}
\text{SI-GHOST} \\
\hline
\Theta_0 [P_0]_{\mathfrak{m}}^{\text{sk}} \mathbb{A} \leq P_0 \vdash \Theta_0
\end{array}
\qquad
\begin{array}{c}
\text{SI-UNIVGHOSTL} \\
\mathfrak{m} \neq \mathfrak{I} \quad \Theta_0 [P_0]_{\mathfrak{m}}^{\text{sk}} P \leq Q \vdash \Theta_1 \\
\hline
\Theta_0 [P_0]_{\mathfrak{m}}^{\text{sk}} \mathbb{A}.P \leq Q \vdash \Theta_1
\end{array}$$

$$\boxed{\Theta \mid \Xi [P_0]_{\mathfrak{Fm}}^n \hat{\alpha} \leq Q \vdash \Theta'}$$

$$\begin{array}{c}
\text{SI-FLEX-SOLVE} \\
\hat{\alpha}, \hat{\beta} \text{ fresh} \quad \Theta_0, \hat{\alpha}, \hat{\beta}, \hat{\alpha}_0 := \hat{\alpha} \rightarrow \hat{\beta}, \Xi [P_0]_{\mathfrak{Fm}}^n \hat{\alpha} \rightarrow \hat{\beta} \leq Q \vdash \Theta_1 \\
\hline
\Theta_0, \hat{\alpha}_0 \mid \Xi [P_0]_{\mathfrak{Fm}}^n \hat{\alpha}_0 \leq Q \vdash \Theta_1
\end{array}$$

$$\begin{array}{c}
\text{SI-FLEX-ASSIGN} \\
\Theta_0, \Xi [P_0[\tau/\hat{\beta}]]_{\mathfrak{Fm}}^n \hat{\alpha}[\tau/\hat{\beta}] \leq Q \vdash \Theta_1 \\
\hline
\Theta_0, \hat{\beta} := \tau \mid \Xi [P_0]_{\mathfrak{Fm}}^n \hat{\alpha} \leq Q \vdash \Theta_1, \hat{\beta} := \tau
\end{array}
\qquad
\begin{array}{c}
\text{SI-FLEX-DEPEND} \\
\hat{\alpha} \neq \hat{\beta} \quad \hat{\beta} \in \text{ftv}(P_0) \\
\Theta_0 \mid \hat{\beta}, \Xi [P_0]_{\mathfrak{Fm}}^n \hat{\alpha} \leq Q \vdash \Theta_1 \\
\hline
\Theta_0, \hat{\beta} \mid \Xi [P_0]_{\mathfrak{Fm}}^n \hat{\alpha} \leq Q \vdash \Theta_1
\end{array}$$

$$\begin{array}{c}
\text{SI-FLEX-SKIP} \\
\hat{\alpha} \neq \hat{\beta} \quad \hat{\beta} \notin \text{ftv}(P_0) \\
\Theta_0 \mid \Xi [P_0]_{\mathfrak{Fm}}^n \hat{\alpha} \leq Q \vdash \Theta_1 \\
\hline
\Theta_0, \hat{\beta} \mid \Xi [P_0]_{\mathfrak{Fm}}^n \hat{\alpha} \leq Q \vdash \Theta_1, \hat{\beta}
\end{array}
\qquad
\begin{array}{c}
\text{SI-FLEX-SKIPRIGID} \\
\hat{\beta} \notin \text{ftv}(P_0) \quad \Theta_0 \mid \Xi [P_0]_{\mathfrak{m}}^n \hat{\alpha} \leq Q \vdash \Theta_1 \\
\hline
\Theta_0, \beta \mid \Xi [P_0]_{\mathfrak{m}}^n \hat{\alpha} \leq Q \vdash \Theta_1, \beta
\end{array}$$

$$\begin{array}{c}
\text{SI-FLEX-SKIPTMVAR} \\
\Theta_0 \mid \Xi [P_0]_{\mathfrak{Fm}}^n \hat{\alpha} \leq Q \vdash \Theta_1 \\
\hline
\Theta_0, x : P \mid \Xi [P_0]_{\mathfrak{Fm}}^n \hat{\alpha} \leq Q \vdash \Theta_1, x : P
\end{array}$$

Figure 9.7: Algorithmic subtyping.

Rules SI-INFER, SI-CHECK, SI-FORALLR, SI-UNIVGHOSTR, SI-ARROW, SI-GHOST, and SI-UNIVGHOSTL closely follow the declarative rules in Sections 8.5 and 8.8.

Rule SI-FORALLL uses the auxiliary look judgement $P \leq_m^n P' \not\vdash^\alpha \Xi; Q$ defined in Section 9.6 to obtain a skeleton P_1 , which uses fresh flexible variables in Ξ , from the inputs P and P_0 . It checks the well-formedness of P_1 and uses it to instantiate α . Its side conditions ($m \neq \text{I}$ and P_0 guarded or $P_0 = \perp$) avoid overlapping with other rules.

Rule SI-FLEXMONO corresponds to declarative rule S-FLEXMONO in Section 9.1.2 when the flexible variable has a definition. For an unsolved flexible variable, there is no corresponding declarative rule. Rule SI-FLEXMONO handles both cases in function mode by using the auxiliary judgement $\Theta \mid \Xi [P_0]_{Fm}^n \hat{\alpha} \leq Q \dashv \Theta'$, which is similar to the variable assignment judgement in Section 9.4. We only consider the function mode Fm here to avoid overlapping; the other modes are covered by the first four rules.

We define problems and solutions of subtyping, and then state the soundness, completeness, and optimality of subtyping. The proofs can be found in Appendices D.4.2 and D.4.3.

Definition 12 (Problems and solutions of subtyping). *A subtyping problem is a tuple $(\Theta_0; n; m; P_0; P)$ where Θ_0 is well-formed and P_0 and P are well-formed in Θ_0 and $P_0 :^n m$. A solution to it is a tuple $(\theta_1 :: \Theta_0 \sqsubseteq \Theta_1; P_1)$ such that $\Theta_1 \vdash_m^n \theta_1 P \leq P_1$ where $\Theta_1 \vdash \theta_1 P_0 \blacktriangleright P_1$. The solution is optimal if for any other solution $(\theta :: \Theta_0 \sqsubseteq \Theta; Q)$, there exists a metasubstitution $\zeta :: \Theta_1 \sqsubseteq \Theta$ such that $\theta \equiv \zeta \circ \theta_1 :: \Theta_0 \sqsubseteq \Theta$ and $\Theta \vdash \zeta P_1 \leq Q$.*

Theorem 27 (Soundness of subtyping). *Given a subtyping problem $(\Theta_0; n; m; P_0; P)$, if $\Theta_0 [P_0]_{Fm}^n P \leq Q \dashv \Theta_1$, then $(\Theta_0 \sqsubseteq \Theta_1; Q)$ is a solution.*

Theorem 28 (Completeness and optimality of subtyping). *Given a subtyping problem $(\Theta_0; n; m; P_0; P)$, if there exists a solution, $\Theta_0 [P_0]_{Fm}^n P \leq P_1 \dashv \Theta_1$ gives its optimal solution $(\Theta_0 \sqsubseteq \Theta_1; P_1)$.*

9.6 Constraint collection and solving for subtyping

Figures 9.8 and 9.9 give the look, constraint collection, and constraint solving rules. The look judgement has the form $P \leq_m^n P_0 \not\vdash^\alpha \Xi; P_1$. The input includes a sort n , a mode m , and two skeletons P and P_0 . The goal of this judgement is to find a proper substitution for α in P such that the substituted P can be upcast to some Q compatible with P_0 . The type variable α may appear only in P . We require this as a well-formedness condition of the judgement. We do not need the context Θ_0 since we care about extracting polymorphic information, while flexible variables in Θ_0 only carry monomorphic information. The

output includes a skeleton P_1 that carries all polymorphic information we extract for α , and a context Ξ of fresh flexible variables used in P_1 .

The look judgement replaces the guessing of decorated skeletons in S-FORALL in Sections 8.5 and 8.8. A look judgement first collects all constraints C and fresh flexible variables Ξ for α using the constraint-collection judgement $P \leq_m P' \not\vdash^\alpha \Xi; C$. A constraint list C contains subtyping constraints $\text{sub}_m(P)$ and consistency constraints $\text{eq}(P)$. The look judgement then solves these constraints using the constraint-solving judgement $C; \mathbb{A} \rightsquigarrow \Xi'; Q$, where $\mathbb{A}$ is the empty input skeleton, Q is the result, and Ξ' is the suffix of fresh flexible variables used in Q . Finally, when $n = \text{sk}$, we use the solved Q as the final output; when $n = \text{ty}$, since we want a type for instantiation, we further transform Q to a type A by guessing fresh flexible variables with the meta-function $\text{guess}(Q) = (\Xi''; A)$ as defined in Section 9.2.

Constraint collection is separated into two judgement forms: one for subtyping $P \leq_m P' \not\vdash^\alpha \Xi; C$ and the other for consistency $P \sim P' \not\vdash^\alpha \Xi; C$. We omit the suffix Ξ when it is empty. Both judgements try to find the skeletons in P' that correspond to appearances of α in P , and collect these constraints. Their rules follow the algorithmic subtyping and consistency rules, respectively. We discuss representative rules here. Rules LS-VAR and LE-VAR deal with the base cases when the LHS is the type variable we look for. We add the constraint $\text{sub}_m(P)$ or $\text{eq}(P)$ to the output constraints. Rules LS-ARROW and LE-ARROW deal with arrows. We recursively look at the argument and result components, and append their constraints and flexible variables together. Rule LS-FORALL removes a top-level universal quantifier on the LHS. We do not need to instantiate β , since our goal is to extract polymorphic information for α , which the instantiation of β contributes nothing to. Rule LE-FLEXR deals with a flexible variable $\hat{\beta}$ on the RHS. Since $\alpha \in \text{ftv}(P)$, we know that α corresponds to some monotype which carries no polymorphic information. Thus, we allocate a fresh flexible variable $\hat{\gamma}$ and add the constraint $\text{eq}(\hat{\gamma})$. Rules LS-ABSENT and LE-ABSENT add no constraint when α does not appear on the left-hand side. Rules LS-VACUOUS and LE-VACUOUS add no constraint when the RHS is a ghost.

Constraint solving $C; P \rightsquigarrow \Xi; Q$ sequentially solves each constraint C , starting from the input skeleton P and producing the output skeleton Q and a suffix Ξ of fresh flexible variables. It calls two auxiliary judgements: $P \leq_m Q \rightsquigarrow \Xi; P'$ for solving a subtyping constraint and $P \sim Q \rightsquigarrow \Xi; P'$ for solving a consistency constraint. We omit the suffix Ξ when it is empty. Their rules again follow the algorithmic subtyping and consistency rules, respectively. We discuss representative rules here. Rules SOLS-VAR and SOLE-VAR deal with the base cases when the LHS is a rigid variable. We directly keep this rigid type variable instead of forcing both sides to coincide, since our goal is to extract polymorphic information while a rigid variable is monomorphic. Similarly, rules SOLE-FLEXL and

SOLS-FLEX also ignore the RHS and keep the left-hand side variable. Rules SOLS-ARROW and SOLE-ARROW deal with arrow shapes. We recursively solve the argument and result components, and build the output arrow skeleton from the results. Rule SOLS-FORALLL deals with a universal quantifier on the LHS, which we simply keep in the output. Rule SOLE-GHOSTL deals with the base case where the LHS is a ghost and we simply keep the RHS in the output.

As an example, consider the algorithmic subtyping judgement

$$\Theta \left[(\forall b. \mathbb{A} \rightarrow b) \rightarrow \text{Int} \rightarrow \text{Int} \right]_{\text{FFC}}^{\text{ty}} \forall a. a \rightarrow a \leq Q \vdash \Theta'$$

for some Q , Θ , and Θ' . Its look judgement has the form:

$$a \rightarrow a \leq_{\text{FFC}}^{\text{ty}} (\forall b. \mathbb{A} \rightarrow b) \rightarrow \text{Int} \rightarrow \text{Int} \not\leq^a \cdot; \forall b. \text{Int} \rightarrow b$$

The constraint collection process collects two constraints $\text{eq}(\forall b. \mathbb{A} \rightarrow b)$ and $\text{sub}_{\text{FC}}(\text{Int} \rightarrow \text{Int})$ by first applying LS-ARROW and then LE-VAR and LS-VAR for the argument and result components, respectively. The constraint solving judgement is derived as follows, with two individual constraint solving judgements highlighted and numbered.

$$\frac{\frac{\mathbb{A} \sim \forall b. \mathbb{A} \rightarrow b \rightsquigarrow \forall b. \mathbb{A} \rightarrow b \text{ ①} \quad \frac{\forall b. \mathbb{A} \rightarrow b \leq_{\text{FC}} \text{Int} \rightarrow \text{Int} \rightsquigarrow \forall b. \text{Int} \rightarrow b \text{ ②}}{\text{sub}_{\text{FC}}(\text{Int} \rightarrow \text{Int}); \forall b. \mathbb{A} \rightarrow b \rightsquigarrow \forall b. \text{Int} \rightarrow b}}{\text{eq}(\forall b. \mathbb{A} \rightarrow b), \text{sub}_{\text{FC}}(\text{Int} \rightarrow \text{Int}); \mathbb{A} \rightsquigarrow \forall b. \text{Int} \rightarrow b}$$

We solve ① by SOLE-GHOSTL. We solve ② by SOLS-FORALLL, SOLS-ARROW, and then SOLE-GHOSTL and SOLS-VAR for the argument and result, respectively.

$$\begin{array}{c}
 \text{Constraints } C ::= \cdot \mid \text{sub}_m(P) \mid \text{eq}(P), C \\
 \\
 \boxed{P \leq_m^n P' \not\leq^\alpha \Xi; Q} \text{ look} \\
 \\
 \frac{P \leq_m P' \not\leq^\alpha \Xi; C \quad C; \mathbb{A} \rightsquigarrow \Xi'; Q \quad (\Xi''; A) = \text{guess}(Q)}{P \leq_m^{\text{ty}} P' \not\leq^\alpha \Xi, \Xi', \Xi''; A} \qquad \frac{P \leq_m P' \not\leq^\alpha \Xi; C \quad C; \mathbb{A} \rightsquigarrow \Xi'; Q}{P \leq_m^{\text{sk}} P' \not\leq^\alpha \Xi, \Xi'; Q} \\
 \\
 \boxed{P \leq_m P' \not\leq^\alpha \Xi; C} \quad \boxed{P \sim P' \not\leq^\alpha \Xi; C} \text{ constraint collection} \\
 \\
 \begin{array}{ccc}
 \text{LS-VAR} & \text{LS-ABSENT} & \text{LS-VACUOUS} \\
 \frac{}{\alpha \leq_m P \not\leq^\alpha \text{sub}_m(P)} & \frac{\alpha \notin \text{ftv}(P)}{P \leq_m P' \not\leq^\alpha \cdot} & \frac{}{P \leq_m \mathbb{A} \not\leq^\alpha \cdot} \\
 \\
 \text{LS-CHECK} & & \text{LS-FORALLR} \\
 \frac{A \neq \alpha \quad A, B \text{ guarded} \quad A \sim B \not\leq^\alpha \Xi; C}{A \leq_c B \not\leq^\alpha \Xi; C} & & \frac{A \leq_c B \not\leq^\alpha \Xi; C}{A \leq_c \forall \beta. B \not\leq^\alpha \Xi; C} \\
 \\
 \text{LS-ARROW} & \text{LS-FORALLL} & \text{LS-UNIVGHOSTL} \\
 \frac{P_1 \sim P'_1 \not\leq^\alpha \Xi_1; C_1 \quad P_2 \leq_m P'_2 \not\leq^\alpha \Xi_2; C_2}{P_1 \rightarrow P_2 \leq_{\text{fm}} P'_1 \rightarrow P'_2 \not\leq^\alpha \Xi_1, \Xi_2; C_1, C_2} & \frac{P \leq_m P' \not\leq^\alpha \Xi; C}{\forall \beta. P \leq_m P' \not\leq^\alpha \Xi; C} & \frac{P \leq_m P' \not\leq^\alpha \Xi; C}{\mathbb{A}. P \leq_m P' \not\leq^\alpha \Xi; C} \\
 \\
 \text{LE-VAR} & \text{LE-ABSENT} & \text{LE-VACUOUS} & \text{LE-FLEXR} \\
 \frac{}{\alpha \sim P \not\leq^\alpha \text{eq}(P)} & \frac{\alpha \notin \text{ftv}(P)}{P \sim P' \not\leq^\alpha \cdot} & \frac{}{P \sim \mathbb{A} \not\leq^\alpha \cdot} & \frac{P \neq \alpha \quad \alpha \in \text{ftv}(P) \quad \hat{\gamma} \text{ fresh}}{P \sim \hat{\beta} \not\leq^\alpha \hat{\gamma}; \text{eq}(\hat{\gamma})} \\
 \\
 \text{LE-ARROW} & \text{LE-FORALLINH} & \text{LE-FORALLSYN} \\
 \frac{P_1 \sim P'_1 \not\leq^\alpha \Xi_1; C_1 \quad P_2 \sim P'_2 \not\leq^\alpha \Xi_2; C_2}{P_1 \rightarrow P_2 \sim P'_1 \rightarrow P'_2 \not\leq^\alpha \Xi_1, \Xi_2; C_1, C_2} & \frac{P \sim P' \not\leq^\alpha \Xi; C}{\forall \beta. P \sim \forall \beta. P' \not\leq^\alpha \Xi; C} & \frac{P \sim \mathbb{A}. P' \not\leq^\alpha \Xi; C}{\forall \beta. P \sim \mathbb{A}. P' \not\leq^\alpha \Xi; C} \\
 \\
 \text{LE-UNIVGHOST} & \text{LE-UNIVGHOST1} & \text{LE-UNIVGHOST2} \\
 \frac{P \sim P' \not\leq^\alpha \Xi; C}{\mathbb{A}. P \sim \mathbb{A}. P' \not\leq^\alpha \Xi; C} & \frac{\mathbb{A}. P \sim P' \not\leq^\alpha \Xi; C}{\mathbb{A}. P \sim \forall \beta. P' \not\leq^\alpha \Xi; C} & \frac{P' \text{ guarded} \quad P \sim P' \not\leq^\alpha \Xi; C}{\mathbb{A}. P \sim P' \not\leq^\alpha \Xi; C} \\
 \\
 \text{LE-UNIVGHOSTR} \\
 \frac{P \neq \alpha \quad P \text{ guarded} \quad P \sim P' \not\leq^\alpha \Xi; C}{P \sim \mathbb{A}. P' \not\leq^\alpha \Xi; C}
 \end{array}
 \end{array}$$

Figure 9.8: Look and constraint collection.

$C; P \rightsquigarrow \Xi; Q$ constraint solving			
$\frac{}{.; P \rightsquigarrow P}$	$\frac{P \leq_m Q \rightsquigarrow \Xi; P'}{\text{sub}_m(Q); P \rightsquigarrow \Xi; P'}$	$\frac{P \sim Q \rightsquigarrow \Xi_1; P_1 \quad C; P_1 \rightsquigarrow \Xi_2; P'}{\text{eq}(Q), C; P \rightsquigarrow \Xi_1, \Xi_2; P'}$	
$P \leq_m Q \rightsquigarrow \Xi; P'$			
$\frac{\text{SOLS-VAR}}{\alpha \leq_m Q \rightsquigarrow \alpha}$	$\frac{\text{SOLS-FORALLL} \quad P \leq_m Q \rightsquigarrow \Xi; P'}{\forall \alpha. P \leq_m Q \rightsquigarrow \Xi; \forall \alpha. P'}$	$\frac{\text{SOLS-UNIVGHOSTL} \quad P \leq_m Q \rightsquigarrow \Xi; P'}{\mathbb{A}.P \leq_m Q \rightsquigarrow \Xi; \mathbb{A}.P'}$	$\frac{\text{SOLS-VACUOUS}}{P \leq_m \mathbb{A} \rightsquigarrow P}$
$\frac{\text{SOLS-CHECK} \quad P, B \text{ guarded} \quad P \sim B \rightsquigarrow \Xi; P'}{P \leq_c B \rightsquigarrow \Xi; P'}$		$\frac{\text{SOLS-FORALLR} \quad A \leq_c B \rightsquigarrow \Xi; P'}{A \leq_c \forall \beta. B \rightsquigarrow \Xi; P'}$	
$\frac{\text{SOLS-ARROW} \quad P_1 \sim Q_1 \rightsquigarrow \Xi_1; P'_1 \quad P_2 \leq_m Q_2 \rightsquigarrow \Xi_2; P'_2}{P_1 \rightarrow P_2 \leq_{\text{Fm}} Q_1 \rightarrow Q_2 \rightsquigarrow \Xi_1, \Xi_2; P'_1 \rightarrow P'_2}$		$\frac{\text{SOLS-GHOSTLF} \quad \mathbb{A} \sim Q_1 \rightsquigarrow \Xi_1; P'_1 \quad \mathbb{A} \leq_m Q_2 \rightsquigarrow \Xi_2; P'_2}{\mathbb{A} \leq_{\text{Fm}} Q_1 \rightarrow Q_2 \rightsquigarrow \Xi_1, \Xi_2; \mathbb{A}.P'_1 \rightarrow P'_2}$	
$\frac{\text{SOLS-GHOSTL} \quad B \text{ guarded}}{\mathbb{A} \leq_c B \rightsquigarrow \mathbb{A}.B}$		$\frac{\text{SOLS-FLEX}}{\hat{\alpha} \leq_m Q \rightsquigarrow \hat{\alpha}}$	
$P \sim Q \rightsquigarrow \Xi; P'$			
$\frac{\text{SOLE-UNIT}}{1 \sim 1 \rightsquigarrow 1}$	$\frac{\text{SOLE-VAR}}{\alpha \sim Q \rightsquigarrow \alpha}$	$\frac{\text{SOLE-GHOSTL}}{\mathbb{A} \sim Q \rightsquigarrow Q}$	$\frac{\text{SOLE-VACUOUSR}}{P \sim \mathbb{A} \rightsquigarrow P}$
$\frac{\text{SOLE-ARROW} \quad P_1 \sim Q_1 \rightsquigarrow \Xi_1; P'_1 \quad P_2 \sim Q_2 \rightsquigarrow \Xi_2; P'_2}{P_1 \rightarrow P_2 \sim Q_1 \rightarrow Q_2 \rightsquigarrow \Xi_1, \Xi_2; P'_1 \rightarrow P'_2}$		$\frac{\text{SOLE-FORALL} \quad P \sim Q \rightsquigarrow \Xi; P'}{\forall \alpha. P \sim \forall \alpha. Q \rightsquigarrow \Xi; \forall \alpha. P'}$	
$\frac{\text{SOLE-UNIVGHOST} \quad P \sim Q \rightsquigarrow \Xi; P'}{\mathbb{A}.P \sim \mathbb{A}.Q \rightsquigarrow \Xi; \mathbb{A}.P'}$	$\frac{\text{SOLE-GHOSTFORALL} \quad \mathbb{A}.P \sim Q \rightsquigarrow \Xi; P'}{\mathbb{A}.P \sim \forall \alpha. Q \rightsquigarrow \Xi; \forall \alpha. P'}$	$\frac{\text{SOLE-GHOSTGUARDED} \quad Q \text{ guarded} \quad P \sim Q \rightsquigarrow \Xi; P'}{\mathbb{A}.P \sim Q \rightsquigarrow \Xi; P'}$	
$\frac{\text{SOLE-FORALLGHOST} \quad P \sim \mathbb{A}.Q \rightsquigarrow \Xi; P'}{\forall \alpha. P \sim \mathbb{A}.Q \rightsquigarrow \Xi; \forall \alpha. P'}$	$\frac{\text{SOLE-GUARDEDGHOST} \quad P \text{ guarded} \quad P \sim Q \rightsquigarrow \Xi; P'}{P \sim \mathbb{A}.Q \rightsquigarrow \Xi; P'}$		$\frac{\text{SOLE-FLEXL}}{\hat{\alpha} \sim P \rightsquigarrow \hat{\alpha}}$
$\frac{\text{SOLE-FLEXR} \quad P \text{ non-flex} \quad \text{guess}(P) = (\Xi; P')}{P \sim \hat{\alpha} \rightsquigarrow \Xi; P'}$			

Figure 9.9: Constraint solving.

10 Inferring modal effect types

Farewell, Maetel. Farewell, Galaxy Express 999. Farewell, days of youth.

—Closing line, *Galaxy Express 999* (1979)

We outline the design of *Maetel*, a surface language with modal effect types and type inference based on *Fresco*. As introduced in Part I, *MET* is an explicit core calculus which requires explicit annotations for lambda abstractions as well as explicit introduction and elimination of modal types. The extension with polymorphism in Section 5.4 also requires explicit type abstraction and instantiation. Even though *MET* reduces the verbosity of conventional effect types by not requiring effect variables, the remaining annotation burdens of explicit modalities and polymorphism in *MET* are still not satisfactory for practical programming.

In Chapters 8 and 9, we have introduced the *Fresco* type inference approach which infers first-class polymorphism, allowing polymorphic types to be implicitly introduced, eliminated, and used for instantiation. In this chapter, we adapt the *Fresco* type inference algorithm to further infer *first-class modal types*, allowing modal types to be implicitly introduced, eliminated, and used for instantiation. We only consider modal effect types in this chapter, but we believe the same approach can be applied to other modal type systems.

We first discuss how the problem of inferring first-class modal types is related to the problem of inferring first-class polymorphism (Section 10.1). Then we formally present the declarative type system of a core fragment of *Maetel*, which extends the declarative type system of *Fresco* with modal effect types (Section 10.2), and sketch how the algorithm is extended as well (Section 10.3). Finally we discuss other features and design choices of the language *Maetel* (Section 10.4).

10.1 First-class modal types

As discussed in Chapter 8, first-class polymorphism consists of higher-rank polymorphism, which allows quantifiers to appear in arbitrary positions in types, and polymorphic instantiation, which allows instantiation with polymorphic types. Similarly, first-class

modal types allow modalities to appear in arbitrary positions in types and allow modal instantiation, i.e., instantiation with modal types. Our goal of inferring first-class modal types is to infer introduction and elimination of modalities, as well as modal instantiation.

As observed by [Tang et al. \[2025\]](#), inferring implicit introduction and elimination of modalities is closely related to inferring implicit type abstraction and instantiation for FCP. For instance, in **MET**, a type $\forall a.[\log](a \rightarrow a)$ denotes a polymorphic function that may use the $\log$ effect, as specified by the modality $[\log]$. Consequently, for the type checking judgement $\Gamma \vdash_c \lambda x.x : \forall a.[\log](a \rightarrow a)$, we infer not only the type abstraction of a , but also the modality introduction of $[\log]$. As another example, for the application `run idlog` where `run` : $(\text{Int} \rightarrow \text{Int}) \rightarrow \text{Int}$ and `idlog` : $\forall a.[\log](a \rightarrow a)$, we not only infer the instantiation of a to Int , but also the elimination of the modality $[\log]$. Usual bidirectional type systems, such as the one proposed in [Tang et al. \[2025\]](#), can neatly infer such implicit introduction and elimination of modalities, but only in a simply-typed setting.

Incorporating modal instantiation is challenging, for the same reason that incorporating polymorphic instantiation in FCP is. We must avoid guessing arbitrary modal types for instantiation. Otherwise, for instance, `single ilog` where `ilog` : $[\log](\text{Int} \rightarrow \text{Int})$ would have two possible types `List ([log](Int → Int))` and `[log](List (Int → Int))` with no best choice.

Mixing different directions of information flow plays an important role in inferring modal instantiation, as it does in inferring polymorphic instantiation. It is particularly desirable for modal effect types. A typical pattern of effectful programming is that given a polymorphic effect handler $h : \forall a.\langle E \rangle(1 \rightarrow A) \rightarrow B$ for some effect E and types A and B (the modality $\langle E \rangle$ attaches to the argument type), we want to apply this handler to a suspended computation $\lambda x.M$ to handle the effects used in M . In order to type $h(\lambda x.M)$, the information should first flow from the argument $\lambda x.M$ to the function h for instantiation of a , then flow back to the argument to introduce the modality $\langle E \rangle$.

10.2 A haunted bidirectional type system with modal effects

We present a declarative type system for **Maetel** based on the declarative type system of **Fresco** in Chapter 8. For brevity, we only consider the core features of **MET** from Chapter 4 plus polymorphism for value types from Section 5.4. Other extensions are more routine; we will discuss some of them in Section 10.4.

10.2.1 Syntax

Figure 10.1 gives the syntax of Maetel. It extends Fresco syntax with modal effect types, including extensions, effect contexts, modalities, kinds, locks, and term constructs for invoking and handling effects. We may omit the kind of a type variable when it is Any. The mode theory remains unchanged from that of MET in Section 4.3.

Types	$A, B ::= 1 \mid \alpha \mid A \rightarrow B \mid \forall \alpha^K. A \mid \mu A$		
Synthesised types	$\textcolor{red}{A}, \textcolor{red}{B} ::= 1 \mid \alpha \mid \textcolor{red}{A} \rightarrow \textcolor{red}{B} \mid \forall \alpha^K. \textcolor{red}{A} \mid \mu \textcolor{red}{A}$		
Inherited types	$\textcolor{blue}{A}, \textcolor{blue}{B} ::= 1 \mid \alpha \mid \textcolor{blue}{A} \rightarrow \textcolor{blue}{B} \mid \forall \alpha^K. \textcolor{blue}{A} \mid \mu \textcolor{blue}{A}$		
Bidirectional types	$A, B ::= 1 \mid \alpha \mid A \rightarrow B \mid \forall \alpha^K. A \mid \forall \alpha^K. A \mid \mu A \mid \mu A$		
Skeletons	$P, Q ::= 1 \mid \alpha \mid P \rightarrow Q \mid \forall \alpha^K. P \mid \mu P \mid \mathbb{A} \mid \mathbb{A}.P$		
Synthesised skeletons	$\textcolor{red}{P}, \textcolor{red}{Q} ::= 1 \mid \alpha \mid \textcolor{red}{P} \rightarrow \textcolor{red}{Q} \mid \forall \alpha^K. \textcolor{red}{P} \mid \mu \textcolor{red}{P} \mid \mathbb{A} \mid \mathbb{A}.P$		
Inherited skeletons	$\textcolor{blue}{P}, \textcolor{blue}{Q} ::= 1 \mid \alpha \mid \textcolor{blue}{P} \rightarrow \textcolor{blue}{Q} \mid \forall \alpha^K. \textcolor{blue}{P} \mid \mu \textcolor{blue}{P} \mid \mathbb{A} \mid \mathbb{A}.P$		
Bidirectional skeletons	$P, Q ::= 1 \mid \alpha \mid P \rightarrow Q \mid \forall \alpha^K. P \mid \forall \alpha^K. P \mid \mu P \mid \mu P \mid \mathbb{A} \mid \mathbb{A}.P$		
Extensions	$D ::= \cdot \mid \ell, D$	Terms	$M, N ::= () \mid x \mid \lambda x. M \mid \lambda x^A. M \mid M N$
Effect contexts	$E, F ::= \cdot \mid \ell, E$		$\mid M : A \mid \lceil M \rceil$
Modalities	$\mu, \nu ::= [E] \mid \langle D \rangle$		$\mid \text{do } \ell M \mid \text{handle } M \text{ with } H$
Kinds	$K ::= \text{Abs} \mid \text{Any}$	Handlers	$H ::= \{\text{return } x \mapsto N\} \uplus \{\ell_i p_i r_i \mapsto N_i\}_i$
Telescopes	$\rho ::= \cdot \mid \forall \alpha^K. \rho \mid \mu \rho$	Values	$V ::= () \mid x \mid \lambda x. M \mid \lambda x^A. M \mid V : A$
Modes	$\mathfrak{m} ::= \mathbb{I} \mid \mathbb{C} \mid \mathbb{F} \mathfrak{m}$	Contexts	$\Gamma ::= \cdot \mid \Gamma, \alpha : K \mid \Gamma, x : A \mid \Gamma, \mathbb{A}_\mu$

Figure 10.1: Syntax of Maetel.

We colour modalities in the same way as quantifiers. Thus, a synthesised or inherited type has all quantifiers and modalities in the corresponding colour. A bidirectional type may mix the two colours. Skeletons are also extended with modalities μP . A universal ghost $\mathbb{A}$ now represents both unknown quantifiers and unknown modalities.

The kinding relation for types is unchanged from MET. For coloured types, we simply ignore colours. Figure 10.2 gives kinding and refinement for skeletons. The kinding relation combines kinding of types in MET with well-formedness of skeletons in Fresco. We highlight rules relevant to modalities and ghosts. The rules for modalities follow the kinding rules of MET. The most interesting rules are for ghosts. An ordinary ghost $\mathbb{A}$ has kind Any. A universal ghost preserves the kind of its body, which must be guarded and independent of the variables bound by the skeleton as required in Fresco. As a result, we can preserve the kind of a skeleton regardless of how it is refined later. Refinement may refine a skeleton of kind Any to one of kind Abs, which is also of kind Any.

One main difference from MET is that since modality eliminations are implicit in Maetel, variable bindings in contexts do not need to carry modality indices any more. Instead, for a variable binding $x : A$, all top-level modalities of A can be implicitly

$$\boxed{\Gamma \mid \Delta \vdash P : K}$$

$$\begin{array}{c}
\boxed{\Gamma \mid \Delta \vdash \mathbb{A} : \text{Any}} \quad \overline{\Gamma \mid \Delta \vdash 1 : \text{Abs}} \quad \frac{\Gamma \mid \Delta \vdash P : \text{Abs}}{\Gamma \mid \Delta \vdash P : \text{Any}} \quad \frac{\Gamma \ni \alpha : K}{\Gamma \mid \Delta \vdash \alpha : K} \\
\\
\frac{\Gamma \mid \Delta \vdash P : \text{Any} \quad \Gamma \mid \Delta \vdash Q : \text{Any}}{\Gamma \mid \Delta \vdash P \rightarrow Q : \text{Any}} \quad \frac{\Gamma, \alpha : K \mid \Delta, \alpha \vdash P : K'}{\Gamma \mid \Delta \vdash \forall \alpha^K. P : K'} \quad \frac{\Gamma \vdash [E] \quad \Gamma \mid \Delta \vdash P : \text{Any}}{\Gamma \mid \Delta \vdash [E] P : \text{Abs}} \\
\\
\boxed{\frac{\Gamma \vdash \langle D \rangle \quad \Gamma \mid \Delta \vdash P : K}{\Gamma \mid \Delta \vdash \langle D \rangle P : K}} \quad \boxed{\frac{P \text{ guarded} \quad \Gamma \setminus \Delta \mid \cdot \vdash P : K}{\Gamma \mid \Delta \vdash \mathbb{A}.P : K}}
\end{array}$$

$$\boxed{\Gamma \mid \Delta \vdash P \blacktriangleright Q}$$

$$\begin{array}{c}
\overline{\Gamma \mid \Delta \vdash 1 \blacktriangleright 1} \quad \overline{\Gamma \mid \Delta \vdash \alpha \blacktriangleright \alpha} \quad \frac{\Gamma \mid \Delta \vdash P_1 \blacktriangleright Q_1 \quad \Gamma \mid \Delta \vdash P_2 \blacktriangleright Q_2}{\Gamma \mid \Delta \vdash P_1 \rightarrow P_2 \blacktriangleright Q_1 \rightarrow Q_2} \\
\\
\frac{\Gamma, \alpha : K \mid \Delta, \alpha \vdash P \blacktriangleright Q}{\Gamma \mid \Delta \vdash \forall \alpha^K. P \blacktriangleright \forall \alpha^K. Q} \quad \boxed{\frac{\Gamma \mid \Delta \vdash P \blacktriangleright Q}{\Gamma \mid \Delta \vdash \mu P \blacktriangleright \mu Q}} \quad \frac{\Delta \cap \text{ftv}(Q) = \emptyset}{\Gamma \mid \Delta \vdash \mathbb{A} \blacktriangleright Q} \quad \frac{\Gamma \mid \Delta \vdash P \blacktriangleright Q}{\Gamma \mid \Delta \vdash \mathbb{A}.P \blacktriangleright Q} \\
\\
\frac{\Gamma, \alpha : K \mid \Delta \vdash \mathbb{A}.P \blacktriangleright Q}{\Gamma \mid \Delta \vdash \mathbb{A}.P \blacktriangleright \forall \alpha^K. Q} \quad \boxed{\frac{\Gamma \mid \Delta \vdash \mathbb{A}.P \blacktriangleright Q}{\Gamma \mid \Delta \vdash \mathbb{A}.P \blacktriangleright \mu Q}} \quad \frac{\Gamma \mid \Delta \vdash P \blacktriangleright Q}{\Gamma \mid \Delta \vdash \mathbb{A}.P \blacktriangleright \mathbb{A}.Q}
\end{array}$$

Figure 10.2: Kinding and refinement rules for skeletons in Maetel.

eliminated as necessary when using x . An elaboration process, which we will discuss in Section 10.5.2, can reintroduce `let mod` terms for explicit elimination.

A telescope ρ records a sequence of quantifiers and modalities. For P ranging over types and skeletons, telescope application $\rho\{P\}$ is defined as follows.

$$\cdot\{P\} = P \quad (\forall \alpha^K. \rho)\{P\} = \forall \alpha^K. \rho\{P\} \quad (\mu \rho)\{P\} = \mu \rho\{P\}$$

We introduce some conventions for writing telescopes. As usual, we write $\overline{\forall \alpha^K}$ for a telescope consisting of a sequence of quantifiers, and write $\overline{\forall \alpha^K}. A$ for the type $\overline{\forall \alpha^K}\{A\}$. Similarly, we write $\overline{\mu}$ for a telescope consisting of a sequence of modalities, and write $\overline{\mu}A$ for the type $\overline{\mu}\{A\}$. For brevity, we also write $\overline{\mu}$ in places where a modality is expected, by which we mean the modality derived by composing the sequence of modalities in $\overline{\mu}$ starting with the identity modality $\langle \rangle$.

10.2.2 Typing and skeleton inference

Figure 10.3 gives the declarative typing rules for Maetel. The typing judgement has the form $\Gamma \vdash_m M : A @ E$. It extends the judgement of Fresco with an ambient effect context

E . We require the usual well-formedness conditions from **Fresco** and **MET**. Most typing rules are inherited from **Fresco** by adding the same effect context E to the premises and conclusion. We highlight rules that are new or changed.

$\Gamma \vdash A \Rightarrow (\nu, B)$		
$\frac{\Gamma \vdash A : \text{Abs}}{\Gamma \vdash A \Rightarrow (\nu, A)}$	$\frac{\Gamma \not\vdash A : \text{Abs} \quad A = \bar{\mu}B \quad B \text{ non-modal} \quad \nu \setminus \bar{\mu} = \zeta}{\Gamma \vdash A \Rightarrow (\nu, \zeta B)}$	
$\Gamma \vdash_m M : A @ E$		
D-UNIT $\frac{}{\Gamma \vdash_I () : 1 @ E}$	D-VAR $\frac{\Gamma_0 \vdash A \Rightarrow (\text{locks}(\Gamma_1), B)}{\Gamma_0, x : A, \Gamma_1 \vdash_I x : \textcolor{red}{B} @ E}$	D-ANNO $\frac{\Gamma \vdash_c M : \textcolor{blue}{A} @ E}{\Gamma \vdash_I (M : A) : \textcolor{red}{A} @ E}$
D-SUB $\frac{\Gamma \vdash_I M : \textcolor{red}{A} @ E \quad \Gamma \vdash_m \textcolor{red}{A} \leq B @ E}{\Gamma \vdash_m M : B @ E}$	D-FREEZE $\frac{\Gamma \vdash_I M : \textcolor{red}{A} @ E}{\Gamma \vdash_c [M] : \textcolor{blue}{A} @ E}$	D-TELESCOPE $\frac{\rho\{\Gamma\} \vdash_c V : \textcolor{blue}{A} @ \rho\{E\}}{\Gamma \vdash_c V : \textcolor{blue}{\rho}\{\textcolor{red}{A}\} @ E}$
D-ABS $\frac{\Gamma, x : A \vdash_m M : B @ E}{\Gamma \vdash_m \lambda x. M : \textcolor{blue}{A} \rightarrow B @ E}$	D-ABSanNO $\frac{\Gamma, x : A \vdash_m M : B @ E}{\Gamma \vdash_m \lambda x^A. M : A \rightarrow B @ E}$	
D-APP $\frac{\Gamma \vdash_c N \Rightarrow \textcolor{red}{P} \quad \Gamma \vdash \textcolor{blue}{P} \blacktriangleright A \quad \Gamma \vdash_m M : \textcolor{red}{A} \rightarrow B @ E \quad \Gamma \vdash_c N : \textcolor{blue}{A} @ E}{\Gamma \vdash_m M N : B @ E}$	D-DO $\frac{\Sigma \ni \ell : A \twoheadrightarrow B \quad \Gamma \vdash_c M : \textcolor{blue}{A} @ \ell, E}{\Gamma \vdash_I \text{do } \ell M : \textcolor{red}{B} @ \ell, E}$	
D-HANDLE $\frac{D = \ell_i \quad \Gamma, \blacksquare_{(D)} \vdash_I M : \textcolor{red}{A} @ D, E \quad \Gamma, x : \langle D \rangle A \vdash_m N : B @ E \quad [\Sigma \ni \ell_i : A_i \twoheadrightarrow B_i \quad \Gamma, p_i : A_i, r_i : B_i \rightarrow B \vdash_m N_i : B @ E]_i}{\Gamma \vdash_m \text{handle } M \text{ with } \{\text{return } x \mapsto N\} \uplus \{\ell_i \ p_i \ r_i \mapsto N_i\}_i : B @ E}$		

Figure 10.3: Declarative typing rules for Maetel.

Rule **D-TELESCOPE** generalises rule **D-FORALL** of **Fresco**. It introduces any sequence of quantifiers and modalities described by a telescope ρ . To state this rule, we define the applications of a telescope to a typing context and an effect context as follows.

$$\begin{aligned}
 \cdot\{\Gamma\} &= \Gamma & \cdot\{E\} &= E \\
 (\forall \alpha^K. \rho)\{\Gamma\} &= \rho\{\Gamma, \alpha : K\} & (\forall \alpha^K. \rho)\{E\} &= \rho\{E\} \\
 (\mu \rho)\{\Gamma\} &= \rho\{\Gamma, \blacksquare_\mu\} & (\mu \rho)\{E\} &= \rho\{\mu(E)\}
 \end{aligned}$$

Both applications are defined only if the inputs are well-formed. A quantifier extends the typing context but does not change the effect context. A modality adds a lock and

transforms the effect context. As with rule D-FORALL of *Fresco* and T-MOD of *MET*, rule D-TELESCOPE is restricted to values.

Rule D-VAR is the most interesting rule. It differs from that of *Fresco* because locks control variable access. For a context $\Gamma_0, x : A, \Gamma_1$, in order to use x , we need to properly eliminate the top-level modalities of A so that x can go across the locks in Γ_1 . We define an across judgement $\Gamma \vdash A \Rightarrow (\nu, B)$ which gives a type B for x after crossing the locks in Γ_1 with $\nu = \text{locks}(\Gamma_1)$. If A has kind *Abs*, we can cross any modality ν without changing the type. Otherwise, we take all top-level modalities $\bar{\mu}$ of A , and calculate the right residual $\zeta = \nu \backslash \bar{\mu}$ for the output type following [Tang et al. \[2025\]](#). The side condition B non-modal indicates that B has no top-level modalities. The right residual operator is defined as follows.

$$\begin{aligned} \nu \backslash [E] &= [E] \\ \langle D' \rangle \backslash \langle D \rangle &= \langle D'' \rangle \quad \text{if } D \equiv D'', D' \end{aligned}$$

Intuitively, the right residual $\zeta = \nu \backslash \mu$ gives the best modality ζ after crossing ν from μ . First, we have $\Gamma \vdash \mu \Rightarrow \nu \circ \zeta @ F$, which shows that ζ is sufficient to cross ν from μ . Second, for any $\Gamma \vdash \mu \Rightarrow \nu \circ \xi @ F$, we have $\Gamma \vdash \zeta \Rightarrow \xi @ \nu(F)$, which shows that ζ is as good as any other remaining modality since it can be further transformed to it. For example, suppose $f : \langle \text{log}, \text{yield} \rangle (1 \rightarrow 1)$ is bound outside a handler for *yield*. Inside the handler, accessing f crosses the lock $\blacktriangleleft_{\langle \text{yield} \rangle}$ introduced by the handler. Since $\langle \text{yield} \rangle \backslash \langle \text{log}, \text{yield} \rangle = \langle \text{log} \rangle$, rule D-VAR gives f the type $\langle \text{log} \rangle (1 \rightarrow 1)$.

Rules D-DO and D-HANDLE extend the corresponding typing rules of *MET* with modes and colours. Rule D-DO requires the inference mode; we can apply D-SUB for other modes. Rule D-HANDLE infers the type of the handled computation M , and checks all handler clauses against the same result type B in the same mode m .

As in Section 8.4, the typing rules here are not syntax-directed because of the overlaps introduced by D-SUB and D-TELESCOPE. We incorporate both rules into other rules to obtain a syntax-directed type system. The full syntax-directed typing rules are given in Figure 10.4, where the rules that differ from *Fresco* are highlighted.

Concretely, we incorporate D-SUB into SD-UNIT, SD-VAR, SD-ANNO, and SD-DO. We incorporate D-TELESCOPE into the two abstraction rules in checking mode, giving SD-ABSCHECK and SD-ABSANNOCHECK. These two rules generalise their *Fresco* counterparts by allowing introduction of both quantifiers and modalities.

Figure 10.5 gives the skeleton inference rules for *Maetel*, which correspond one-to-one with the syntax-directed typing rules in Figure 10.4. The skeleton inference judgement has the same form $\Gamma \vdash_m M \Rightarrow P$ as in *Fresco* with the same well-formedness conditions. As in *Fresco*, we extend variable bindings $x : A$ in contexts to $x : P$ for skeleton inference.

Unlike typing, skeleton inference does not take an effect context and ignores locks in Γ when accessing variables. This is because rules P-ABSCHECK and P-ABSANNOCHECK

$\boxed{\Gamma \vdash_m M : A @ E}$		
$\frac{\text{SD-UNIT}}{\Gamma \vdash_m 1 \leq A @ E}$	$\frac{\text{SD-VAR} \quad \Gamma_0 \vdash A \Rightarrow (\text{locks}(\Gamma_1), B) \quad \Gamma_0, x : A, \Gamma_1 \vdash_m \textcolor{red}{B} \leq B' @ E}{\Gamma_0, x : A, \Gamma_1 \vdash_m x : B' @ E}$	$\frac{\text{SD-ANNO} \quad \Gamma \vdash_c M : \textcolor{blue}{A} @ E \quad \Gamma \vdash_m \textcolor{red}{A} \leq B @ E}{\Gamma \vdash_m (M : A) : B @ E}$
$\frac{\text{SD-FREEZE} \quad \Gamma \vdash_I M : \textcolor{red}{A} @ E}{\Gamma \vdash_c [M] : A @ E}$	$\frac{\text{SD-ABSCHECK} \quad \rho\{\Gamma\}, x : A \vdash_c M : \textcolor{blue}{B} @ \rho\{E\}}{\Gamma \vdash_c \lambda x.M : \rho\{\textcolor{blue}{A} \rightarrow \textcolor{blue}{B}\} @ E}$	$\frac{\text{SD-ABS} \quad m \neq C \quad \Gamma, x : A \vdash_{m-} M : B @ E}{\Gamma \vdash_m \lambda x.M : \textcolor{blue}{A} \rightarrow B @ E}$
$\frac{\text{SD-ABSANNOCHECK} \quad \rho\{\Gamma\}, x : A \vdash_c M : \textcolor{blue}{B} @ \rho\{E\}}{\Gamma \vdash_c \lambda x^A.M : \rho\{\textcolor{blue}{A} \rightarrow \textcolor{blue}{B}\} @ E}$		
$\frac{\text{SD-ABSANNO} \quad m \neq C \quad \Gamma, x : A \vdash_{m-} M : B @ E}{\Gamma \vdash_m \lambda x^A.M : A \rightarrow B @ E}$		
$\frac{\text{SD-APP} \quad \Gamma \vdash_c N \Rightarrow \textcolor{red}{P} \quad \Gamma \vdash \textcolor{blue}{P} \blacktriangleright A \quad \Gamma \vdash_{Fm} M : A \rightarrow B @ E \quad \Gamma \vdash_c N : \textcolor{blue}{A} @ E}{\Gamma \vdash_m M N : B @ E}$	$\frac{\text{SD-DO} \quad \Sigma \ni \ell : A \rightarrow B \quad \Gamma \vdash_c M : \textcolor{blue}{A} @ \ell, E \quad \Gamma \vdash_m \textcolor{red}{B} \leq B' @ \ell, E}{\Gamma \vdash_m \text{do } \ell M : B' @ \ell, E}$	
$\frac{\text{SD-HANDLE} \quad D = \bar{\ell}_i \quad \Gamma, \mathbf{A}_{(D)} \vdash_I M : \textcolor{red}{A} @ D, E \quad \Gamma, x : \langle D \rangle A \vdash_m N : B @ E \quad [\Sigma \ni \ell_i : A_i \rightarrow B_i \quad \Gamma, p_i : A_i, r_i : B_i \rightarrow B \vdash_m N_i : B @ E]_i}{\Gamma \vdash_m \text{handle } M \text{ with } \{\text{return } x \mapsto N\} \uplus \{\ell_i p_i r_i \mapsto N_i\}_i : B @ E}$		

Figure 10.4: Syntax-directed typing rules for Maetel.

$\Gamma \vdash_m M \Rightarrow P$			
$\frac{\text{P-UNIT} \quad \Gamma \vdash_m^{\text{sk}} 1 \leq P}{\Gamma \vdash_m () \Rightarrow P}$	$\frac{\text{P-VAR} \quad \Gamma \ni x : P \quad \Gamma \vdash_m^{\text{sk}} P \leq Q}{\Gamma \vdash_m x \Rightarrow Q}$	$\frac{\text{P-ANNO} \quad \Gamma \vdash_m^{\text{sk}} A \leq P}{\Gamma \vdash_m (M : A) \Rightarrow P}$	$\frac{\text{P-FREEZE} \quad \Gamma \vdash_I M \Rightarrow P}{\Gamma \vdash_c [M] \Rightarrow P}$
$\frac{\text{P-ABSCHECK} \quad \Gamma, x : \mathbb{A} \vdash_c M \Rightarrow Q}{\Gamma \vdash_c \lambda x.M \Rightarrow \mathbb{A}. \mathbb{A} \rightarrow Q}$	$\frac{\text{P-ABS} \quad m \neq c \quad \Gamma, x : P \vdash_{m-} M \Rightarrow Q}{\Gamma \vdash_m \lambda x.M \Rightarrow P \rightarrow Q}$	$\frac{\text{P-ABSANNOCHECK} \quad \Gamma, x : A \vdash_c M \Rightarrow Q}{\Gamma \vdash_c \lambda x^A.M \Rightarrow \mathbb{A}. A \rightarrow Q}$	
$\frac{\text{P-ABSANNO} \quad m \neq c \quad \Gamma, x : A \vdash_{m-} M \Rightarrow Q}{\Gamma \vdash_m \lambda x^A.M \Rightarrow A \rightarrow Q}$	$\frac{\text{P-APP} \quad \Gamma \vdash_c N \Rightarrow P \quad \Gamma \vdash P \blacktriangleright P' \quad \Gamma \vdash_{Fm} M \Rightarrow P' \rightarrow Q}{\Gamma \vdash_m M N \Rightarrow Q}$		
$\frac{\text{P-DO} \quad \Sigma \ni \ell : A \rightarrow B \quad \Gamma \vdash_m^{\text{sk}} B \leq P}{\Gamma \vdash_m \text{do } \ell M \Rightarrow P}$	$\frac{\text{P-HANDLE} \quad D = \ell_i \quad \Gamma \vdash_I M \Rightarrow P \quad \Gamma, x : \langle D \rangle P \vdash_m N \Rightarrow Q \quad [\Sigma \ni \ell_i : A_i \rightarrow B_i \quad \Gamma, p_i : A_i, r_i : B_i \rightarrow Q \vdash_m N_i \Rightarrow Q]_i}{\Gamma \vdash_m \text{handle } M \text{ with } \{\text{return } x \mapsto N\} \uplus \{\ell_i p_i r_i \mapsto N_i\}_i \Rightarrow Q}$		

Figure 10.5: Skeleton inference rules for Maetel.

introduce universal ghosts which may represent any modalities. As a result, even if we tracked effect contexts, we would not know what the effect context for the premise should be and which lock to introduce. Since we do not track effect contexts or introduce locks in rules P-ABSCHECK and P-ABSANNOCHECK, rule P-VAR reads the skeleton stored in the context without using the across judgement. Another option is to introduce ghosts into locks and effect contexts, so that for rules P-ABSCHECK and P-ABSANNOCHECK, we can introduce $\mathfrak{L}_{\mathfrak{A}}$ and set the effect context to $\mathfrak{A}$ for the premises. We leave this as future work.

Freezing interacts with modal types in an interesting way during skeleton inference. In short, for $x : A$, the frozen term $\llbracket x \rrbracket$ makes sure that the skeleton must be the type A as if the variable were not influenced by any locks in the context and any input type. This behaviour is consistent with the fact that during skeleton inference, we do not know which potential modalities may be introduced.

10.2.3 Subtyping and consistency

As in Fresco, we introduce decorated types which attach tags to quantifiers and modalities. Figure 10.6 gives the syntax and well-formedness rules for types with tags. These provenance tags ensure that subtyping does not guess quantifiers or modalities out of thin air.

Decorated types	$\varphi ::= 1 \mid \alpha \mid \varphi_1 \rightarrow \varphi_2 \mid \forall^\epsilon \alpha^K. \varphi \mid \mu^\epsilon \varphi$	Tag contexts	$\Psi ::= \cdot \mid \Psi, \epsilon$
Unresolved types	$T ::= 1 \mid \alpha \mid T_1 \rightarrow T_2 \mid \forall^\delta \alpha^K. T \mid \mu^\delta T$	Potential tags	$\delta ::= \cdot \mid \epsilon$
		Tags	ϵ

$\Gamma \vdash \varphi \# \Psi : K$	$\varphi \# \Psi$
-------------------------------------	-------------------

$\frac{\Gamma \vdash \varphi : K \quad \varphi \# \Psi}{\Gamma \vdash \varphi \# \Psi : K}$	$\frac{}{1 \# \cdot}$	$\frac{}{\alpha \# \cdot}$	$\frac{\varphi_1 \# \Psi \quad \varphi_2 \# \Psi'}{\varphi_1 \rightarrow \varphi_2 \# \Psi \uplus \Psi'}$
$\frac{\varphi \# \Psi}{\forall^\epsilon \alpha^K. \varphi \# \Psi \uplus \{\epsilon\}}$	$\frac{\varphi \# \Psi}{\mu^\epsilon \varphi \# \Psi \uplus \{\epsilon\}}$		

Figure 10.6: Syntax and well-formedness rules for tagged types in Maetel.

A decorated type φ has a tag on every quantifier and modality. An unresolved type T may have these tags on some of its quantifiers and modalities. As in Fresco, a tag context Ψ is a set of tags, and a potential tag δ is either a tag or empty. The judgement $\Gamma \vdash \varphi \# \Psi : K$ combines the kinding judgement $\Gamma \vdash \varphi : K$ with $\varphi \# \Psi$, which requires Ψ

to contain every tag in φ exactly once. We say that a type is guarded if it has no top-level quantifier or modality.

Figure 10.7 gives the full subtyping and consistency rules. We highlight the rules about modalities.

$\Psi \mid \Gamma \vdash_m T \leq A @ E$			
$\frac{\text{S-INFER} \quad \cdot \mid \Gamma \vdash T \sim A}{\cdot \mid \Gamma \vdash_I T \leq A @ E}$	$\frac{\text{S-CHECK} \quad T, A \text{ guarded} \quad \Psi \mid \Gamma \vdash T \sim A}{\Psi \mid \Gamma \vdash_c T \leq A @ E}$	$\frac{\text{S-FORALLR} \quad \Psi \mid \Gamma, \alpha : K \vdash_c T \leq A @ E}{\Psi \mid \Gamma \vdash_c T \leq \forall \alpha^K. A @ E}$	
$\frac{\text{S-ARROW} \quad \Psi_1 \mid \Gamma \vdash T_1 \sim A_1 \quad \Psi_2 \mid \Gamma \vdash_m T_2 \leq A_2 @ E}{\Psi_1, \Psi_2 \mid \Gamma \vdash_{Fm} T_1 \rightarrow T_2 \leq A_1 \rightarrow A_2 @ E}$	$\frac{\text{S-FORALLL} \quad m \neq I \quad \Psi' \text{ fresh} \quad \Gamma \vdash \varphi \# \Psi' : K \quad \Psi, \Psi' \mid \Gamma \vdash_m T[\varphi/\alpha] \leq A @ E}{\Psi \mid \Gamma \vdash_m \forall^\delta \alpha^K. T \leq A @ E}$		
$\frac{\text{S-MODCHECK} \quad \Gamma \vdash (\bar{v}, T) \Rightarrow \bar{\mu} @ E \quad \Psi \mid \bar{\mu}\{\Gamma\} \vdash_c T \leq A @ \bar{\mu}\{E\}}{\Psi \mid \Gamma \vdash_c \bar{v}^\delta T \leq \bar{\mu} A @ E}$		$\frac{\text{S-MODFUN} \quad \Gamma \vdash (\bar{v}, T) \Rightarrow \langle \rangle @ E \quad \Psi \mid \Gamma \vdash_{Fm} T \leq A @ E}{\Psi \mid \Gamma \vdash_{Fm} \bar{v}^\delta T \leq A @ E}$	
$\Psi \mid \Gamma \vdash T \sim A$			
$\frac{\text{C-UNIT}}{\cdot \mid \Gamma \vdash 1 \sim 1}$	$\frac{\text{C-VAR} \quad \Gamma \ni \alpha : K}{\cdot \mid \Gamma \vdash \alpha \sim \alpha}$	$\frac{\text{C-FORALLINH} \quad \Psi \mid \Gamma, \alpha : K \vdash T \sim A}{\Psi, \delta \mid \Gamma \vdash \forall^\delta \alpha^K. T \sim \forall \alpha^K. A}$	$\frac{\text{C-FORALLSYN} \quad \Psi \mid \Gamma, \alpha : K \vdash T \sim A}{\Psi \mid \Gamma \vdash \forall^\delta \alpha^K. T \sim \forall \alpha^K. A}$
$\frac{\text{C-MODINH} \quad \Psi \mid \Gamma \vdash T \sim A}{\Psi, \delta \mid \Gamma \vdash \mu^\delta T \sim \mu A}$	$\frac{\text{C-MODSYN} \quad \Psi \mid \Gamma \vdash T \sim A}{\Psi \mid \Gamma \vdash \mu^\delta T \sim \mu A}$	$\frac{\text{C-ARROW} \quad \Psi_1 \mid \Gamma \vdash T_1 \sim A_1 \quad \Psi_2 \mid \Gamma \vdash T_2 \sim A_2}{\Psi_1, \Psi_2 \mid \Gamma \vdash T_1 \rightarrow T_2 \sim A_1 \rightarrow A_2}$	

Figure 10.7: Subtyping and consistency rules for Maetel.

The subtyping judgement has the form $\Psi \mid \Gamma \vdash_m T \leq A @ E$. As with typing, it extends the subtyping judgement of Fresco with an ambient effect context. The subtyping judgement states that, given tags Ψ under Γ in mode m and effect context E , the unresolved type T is a subtype of A . We require standard well-scopedness and compatibility between A and m . The consistency judgement $\Psi \mid \Gamma \vdash T \sim A$ states that, given tags Ψ under Γ , the unresolved type T is consistent with A . It does not depend on an effect context. As in Fresco, Ψ contains only the tags determined by A , rather than all tags in T .

The rules for quantifiers and functions follow Fresco, with kind annotations added to quantifiers. Rule S-MODCHECK introduces, eliminates, or replaces a modality sequence when the transformation judgement holds. Its premise checks the bodies under $\bar{\mu}\{\Gamma\}$ at $\bar{\mu}\{E\}$. Rule S-MODFUN removes a modality sequence from the left-hand side in function

mode. Neither rule determines modality tags, since modality transformation does not determine a unique modality on the left-hand side. The two highlighted consistency rules are the modal counterparts of C-FORALLINH and C-FORALLSYN. Rule C-MODINH solves the tag of an inherited modality, whereas C-MODSYN does not solve the tag of a synthesised modality.

As in *Fresco*, decorated and unresolved skeletons extend their type counterparts with ghosts. Figure 10.8 gives their full syntax and well-formedness rules. The tag judgements for skeletons are defined in the same way as those for types. The additional rules account for dotted ghosts. They have the same role as in *Fresco*.

Decorated skeletons	$\psi ::= 1 \mid \alpha \mid \psi_1 \rightarrow \psi_2 \mid \forall^\epsilon \alpha^K. \psi \mid \mu^\epsilon \psi \mid \mathbb{A} \mid \mathbb{A}.\psi$
Unresolved skeletons	$S ::= 1 \mid \alpha \mid S_1 \rightarrow S_2 \mid \forall^\delta \alpha^K. S \mid \mu^\delta S \mid \mathbb{A} \mid \mathbb{A} \mid \mathbb{A}.S \mid \mathbb{A}.S$
Guarded skeletons	$R ::= 1 \mid \alpha \mid P \rightarrow Q$
Guarded unresolved skeletons	$V ::= 1 \mid \alpha \mid S_1 \rightarrow S_2$

$\Gamma \vdash \psi \# \Psi : K$	$\psi \# \Psi$
----------------------------------	----------------

$\frac{\Gamma \vdash \psi : K \quad \psi \# \Psi}{\Gamma \vdash \psi \# \Psi : K}$	$\frac{}{1 \# \cdot}$	$\frac{}{\alpha \# \cdot}$	$\frac{\psi_1 \# \Psi \quad \psi_2 \# \Psi'}{\psi_1 \rightarrow \psi_2 \# \Psi \uplus \Psi'}$	$\frac{\psi \# \Psi}{\forall^\epsilon \alpha^K. \psi \# \Psi \uplus \{\epsilon\}}$
$\frac{\psi \# \Psi}{\mu^\epsilon \psi \# \Psi \uplus \{\epsilon\}}$	$\frac{}{\mathbb{A} \# \cdot}$	$\frac{\psi \# \Psi}{\mathbb{A}.\psi \# \Psi}$		

Figure 10.8: Syntax and well-formedness rules for tagged skeletons in Maetel.

Figure 10.9 gives the full skeleton subtyping and consistency rules. Most rules follow their counterparts in *Fresco*. The highlighted rules extend them with modalities.

The skeleton subtyping judgement states that, given tags Ψ under Γ in mode m , the unresolved skeleton S is a subtype of P . The skeleton consistency judgement states that, given tags Ψ under Γ , the unresolved skeleton S is consistent with P .

Rule S-MOD removes arbitrary modality sequences from both sides outside inference mode, without determining their tags. It has no modality-transformation premise because skeleton subtyping does not care about whether the transformation holds. Rule C-UNIVGHOSTMOD additionally matches a universal ghost with an inherited modality while retaining the ghost in its premise.

$\Psi \mid \Gamma \vdash_m^{\text{sk}} S \leq P$			
$\frac{\text{S-INFER} \quad \cdot \mid \Gamma \vdash S \sim P}{\cdot \mid \Gamma \vdash_I^{\text{sk}} S \leq P}$	$\frac{\text{S-UNIVGHOSTR} \quad S, P \text{ guarded} \quad \cdot \mid \Gamma \vdash S \sim P}{\cdot \mid \Gamma \vdash_c^{\text{sk}} S \leq \mathbb{A}.P}$	$\frac{\text{S-GHOST}}{\cdot \mid \Gamma \vdash_m^{\text{sk}} \mathbb{A} \leq P}$	$\frac{\text{S-GHOSTG}}{\cdot \mid \Gamma \vdash_m^{\text{sk}} \mathbb{A} \leq \mathbb{A}}$
$\frac{\text{S-MOD} \quad m \neq I \quad \Psi \mid \Gamma \vdash_m^{\text{sk}} S \leq P}{\Psi \mid \Gamma \vdash_m^{\text{sk}} \bar{\mu}^\delta S \leq \bar{\nu} P}$		$\frac{\text{S-ARROW} \quad \Psi_1 \mid \Gamma \vdash S_1 \sim P_1 \quad \Psi_2 \mid \Gamma \vdash_m^{\text{sk}} S_2 \leq P_2}{\Psi_1, \Psi_2 \mid \Gamma \vdash_{Fm}^{\text{sk}} S_1 \rightarrow S_2 \leq P_1 \rightarrow P_2}$	
$\frac{\text{S-FORALLL} \quad m \neq I \quad K \in \{\text{Abs}, \text{Any}\} \quad \Psi' \text{ fresh} \quad \Gamma \vdash \psi \# \Psi' : K \quad \Psi, \Psi' \mid \Gamma \vdash_m^{\text{sk}} S[\psi/\alpha] \leq P}{\Psi \mid \Gamma \vdash_m^{\text{sk}} \forall^\delta \alpha^K. S \leq P}$			
$\frac{\text{S-UNIVGHOSTL} \quad m \neq I \quad \Psi \mid \Gamma \vdash_m^{\text{sk}} S \leq P}{\Psi \mid \Gamma \vdash_m^{\text{sk}} \mathbb{A}.S \leq P}$		$\frac{\text{S-UNIVGHOSTLG} \quad m \neq I \quad \Psi \mid \Gamma \vdash_m^{\text{sk}} S \leq P}{\Psi \mid \Gamma \vdash_m^{\text{sk}} \mathbb{A}.S \leq P}$	
$\Psi \mid \Gamma \vdash S \sim P$			
$\frac{\text{C-UNIT}}{\cdot \mid \Gamma \vdash 1 \sim 1}$	$\frac{\text{C-VAR} \quad \Gamma \ni \alpha : K}{\cdot \mid \Gamma \vdash \alpha \sim \alpha}$	$\frac{\text{C-FORALLINH} \quad \Psi \mid \Gamma, \alpha : K \vdash S \sim P}{\Psi, \delta \mid \Gamma \vdash \forall^\delta \alpha^K. S \sim \forall \alpha^K. P}$	$\frac{\text{C-FORALLSYN} \quad \Psi \mid \Gamma, \alpha : K \vdash S \sim P}{\Psi \mid \Gamma \vdash \forall^\delta \alpha^K. S \sim \forall \alpha^K. P}$
$\frac{\text{C-MODINH} \quad \Psi \mid \Gamma \vdash S \sim P}{\Psi, \delta \mid \Gamma \vdash \mu^\delta S \sim \mu P}$		$\frac{\text{C-MODSYN} \quad \Psi \mid \Gamma \vdash S \sim P}{\Psi \mid \Gamma \vdash \mu^\delta S \sim \mu P}$	
$\frac{\text{C-ARROW} \quad \Psi_1 \mid \Gamma \vdash S_1 \sim P_1 \quad \Psi_2 \mid \Gamma \vdash S_2 \sim P_2}{\Psi_1, \Psi_2 \mid \Gamma \vdash S_1 \rightarrow S_2 \sim P_1 \rightarrow P_2}$			
$\frac{\text{C-GHOST}}{\cdot \mid \Gamma \vdash \mathbb{A} \sim P}$	$\frac{\text{C-GHOSTG}}{\cdot \mid \Gamma \vdash \mathbb{A} \sim \mathbb{A}}$	$\frac{\text{C-UNIVGHOSTG} \quad \Psi \mid \Gamma \vdash S \sim P}{\Psi \mid \Gamma \vdash \mathbb{A}.S \sim \mathbb{A}.P}$	$\frac{\text{C-UNIVGHOST} \quad \Psi \mid \Gamma \vdash S \sim P}{\Psi \mid \Gamma \vdash \mathbb{A}.S \sim \mathbb{A}.P}$
$\frac{\text{C-UNIVGHOST1} \quad \Psi \mid \Gamma, \alpha : K \vdash \mathbb{A}.S \sim P}{\Psi \mid \Gamma \vdash \mathbb{A}.S \sim \forall \alpha^K. P}$		$\frac{\text{C-UNIVGHOSTMOD} \quad \Psi \mid \Gamma \vdash \mathbb{A}.S \sim P}{\Psi \mid \Gamma \vdash \mathbb{A}.S \sim \mu P}$	
$\frac{\text{C-UNIVGHOST2} \quad \Psi \mid \Gamma \vdash S \sim P}{\Psi \mid \Gamma \vdash \mathbb{A}.S \sim P}$			

Figure 10.9: Skeleton subtyping and consistency rules for Maetel.

10.3 Type inference algorithm

We sketch how the type inference algorithm can be extended for Maetel. We leave a full formalisation of the type inference algorithm and its metatheory for future work.

10.3.1 Algorithmic syntax and kinding

Algorithmic syntax. Figure 10.10 gives the algorithmic syntax of Maetel. It extends the syntax of Fresco in Figure 9.1 with the highlighted forms. In particular, types include

modalities, rigid and flexible type variables carry kinds, and algorithmic contexts contain locks. Flexible type variables still range over monotypes, so the algorithm does not guess modalities.

Types	$A, B ::= 1 \mid \alpha \mid A \rightarrow B \mid \forall \alpha^K. A \mid \mu A \mid \hat{a}$
Monotypes	$\tau, \pi ::= 1 \mid \alpha \mid \tau \rightarrow \pi \mid \hat{a}$
Skeletons	$P, Q ::= 1 \mid \alpha \mid P \rightarrow Q \mid \forall \alpha^K. P \mid \mu P \mid \hat{a} \mid \mathbb{Q} \mid \mathbb{Q}.P$
Kinds	$K ::= \text{Abs} \mid \text{Any}$
Extensions	$D ::= \cdot \mid \ell, D$
Effect contexts	$E, F ::= \cdot \mid \ell, E$
Modalities	$\mu, \nu ::= [E] \mid \langle D \rangle$
Entries	$e ::= \alpha : K \mid x : P \mid \hat{a} : K \mid \hat{a} := \tau \mid \mathbb{Q}_\mu$
Algorithmic contexts	$\Theta ::= \cdot \mid \Theta, e$
Suffixes	$\Xi ::= \cdot \mid \Xi, \hat{a} : K \mid \Xi, \hat{a} := \tau$

Figure 10.10: Algorithmic syntax of Maetel.

The well-formedness and metasubstitution rules of *Fresco* extend to the new syntax in the standard way. Before assigning a solution to $\hat{a} : K$, the algorithm checks that the solution has kind K . The solved entry $\hat{a} := \tau$ does not retain K , since any required kind of $\hat{a}$ can then be checked against τ . Locks are preserved by metasubstitution. Since every type of kind *Abs* also has kind *Any*, an information increase may change the kind of an unsolved flexible type variable from *Any* to *Abs*.

Algorithmic kinding. Algorithmic kinding has the form $\Theta_0 \vdash P : K \dashv \Theta_1$, where Θ_1 is an output context and the other components are inputs. Most of its rules follow the declarative kinding rules in Figure 10.2 and thread the context through their premises. The only new rules concern flexible type variables, as shown in Figure 10.11.

$\Theta_0 \vdash P : K \dashv \Theta_1$		
K-FLEXANY $\Theta \ni \hat{a} : K$	K-FLEXABS	K-FLEXDEF $\Theta_0 \vdash \tau : K \dashv \Theta_1$
$\Theta \vdash \hat{a} : \text{Any} \dashv \Theta$	$\Theta_0, \hat{a} : K, \Xi \vdash \hat{a} : \text{Abs} \dashv \Theta_0, \hat{a} : \text{Abs}, \Xi$	$\Theta_0, \hat{a} := \tau, \Xi \vdash \hat{a} : K \dashv \Theta_1, \hat{a} := \tau, \Xi$

Figure 10.11: Algorithmic kinding rules for flexible variables in Maetel.

Algorithmic kinding may change the kind of a flexible type variable from *Any* to *Abs* as shown by rule K-FLEXABS. If the type variable has been solved, rule K-FLEXDEF checks its solution τ at the required kind, which may further change the kind of any flexible type variables in τ .

10.3.2 Algorithmic type and skeleton inference

Type inference. The type inference judgement now includes an ambient effect context and has the form $\Theta_0 [P]_{\mathfrak{m}} M \Rightarrow A \vdash \Theta_1 @E$. The effect context is an input and is unchanged by the judgement. Figure 10.12 shows the central changed rules. The remaining rules follow Fresco.

$$\boxed{\Theta_0 \vdash A \Rightarrow (\nu, B) \vdash \Theta_1}$$

$$\frac{A = \bar{\mu}B \quad B \text{ non-modal} \quad \nu \backslash \bar{\mu} = \zeta}{\Theta \vdash A \Rightarrow (\nu, \zeta B) \vdash \Theta} \quad \frac{\Theta_0 \vdash A : \text{Abs} \vdash \Theta_1}{\Theta_0 \vdash A \Rightarrow (\nu, A) \vdash \Theta_1}$$

$$\boxed{\Theta_0 [P]_{\mathfrak{m}} M \Rightarrow A \vdash \Theta_1 @E}$$

$$\begin{array}{c}
\text{I-VAR} \\
\frac{\Theta_0 \vdash A \Rightarrow (\text{locks}(\Theta_1), B) \vdash \Theta'_0 \quad \Theta'_0, x : A, \Theta_1 [P]_{\mathfrak{m}}^{\text{ty}} B \leq C \vdash \Theta_2 @E}{\Theta_0, x : A, \Theta_1 [P]_{\mathfrak{m}} x \Rightarrow C \vdash \Theta_2 @E} \\
\\
\text{I-ABS} \\
\frac{\text{guess}(P) = (\hat{\alpha} : \text{Any}; A) \quad \Theta_1, \hat{\alpha} : \text{Any}, x : A [Q]_{\mathfrak{m}} M \Rightarrow B \vdash \Theta_2, x : A, \Xi @E}{\Theta_0 [P_0]_{\mathfrak{m}} \lambda x. M \Rightarrow A \rightarrow B \vdash \Theta_2, \Xi @E} \\
\\
\text{I-ABSCHECK} \\
\frac{\rho\{\Theta_0\} \vdash A_0 \triangleright A \rightarrow B \vdash \rho\{\Theta_1\} \quad \rho\{\Theta_1\}, x : A [B]_{\mathfrak{c}} M \Rightarrow B \vdash \rho\{\Theta_2\}, x : A, \Xi @\rho\{E\}}{\Theta_0 [\rho\{A_0\}]_{\mathfrak{c}} \lambda x. M \Rightarrow \rho\{A \rightarrow B\} \vdash \Theta_2 @E} \\
\\
\text{I-ABSANNOCHECK} \\
\frac{\rho\{\Theta_0\} \vdash A_0 \triangleright A' \rightarrow B \vdash \rho\{\Theta_1\} \quad \rho\{\Theta_1\} [A]_{\mathfrak{c}} A' \sim A \vdash \rho\{\Theta_2\} \quad \rho\{\Theta_2\}, x : A [B]_{\mathfrak{c}} M \Rightarrow B \vdash \rho\{\Theta_3\}, x : A, \Xi @\rho\{E\}}{\Theta_0 [\rho\{A_0\}]_{\mathfrak{c}} \lambda x^A. M \Rightarrow \rho\{A \rightarrow B\} \vdash \Theta_3 @E}
\end{array}$$

Figure 10.12: Changed type inference rules for Maetel.

To state rule I-VAR, we use an algorithmic version of the across judgement: $\Theta_0 \vdash A \Rightarrow (\nu, B) \vdash \Theta_1$. The two rules are ordered. The algorithm first attempts to compute the right residual. Only if this fails does it check A against the kind Abs.

Rule I-ABS gives every fresh flexible type variable introduced by $\text{guess}(P)$ kind Any. Typing the body may later require some of these variables to have kind Abs, in which case algorithmic kinding changes their kinds, as in the I-VAR rule. Rules I-ABSCHECK and I-ABSANNOCHECK use a telescope ρ , which is the maximal top-level telescope of the input type. We use ρ to extend the typing context with rigid type variables and locks, and to transform the ambient effect context. The rules for operations and handlers are obtained from the syntax-directed rules in Figure 10.4 by replacing each declarative

premise with its algorithmic counterpart and threading the output context through the premises.

We give an example of how kinds of flexible type variables may change during type inference. Suppose $f : \forall a. []a \rightarrow a$ and consider $\lambda x. f\ x$. Rule I-ABS initially guesses a fresh flexible type variable $\hat{a}$ for x with kind Any. Typing the application instantiates a with $\hat{a}$ and checks x against $[]\hat{a}$. The typing derivation of checking x is:

$$\begin{array}{c}
 \frac{}{\Theta, \hat{a} : \text{Any} \vdash \hat{a} : \text{Abs} \dashv \Theta, \hat{a} : \text{Abs}} \text{K-FLEXABS} \\
 \frac{\Theta, \hat{a} : \text{Any} \vdash \hat{a} \Rightarrow ([] , \hat{a}) \dashv \Theta, \hat{a} : \text{Abs} \quad \dots}{\Theta, \hat{a} : \text{Any}, x : \hat{a}, \blacksquare_{[]} [\hat{a}] \vdash_c x \Rightarrow \hat{a} \dashv \Theta, \hat{a} : \text{Abs}, x : \hat{a}, \blacksquare_{[]} @ \cdot} \text{I-VAR} \\
 \frac{\Theta, \hat{a} : \text{Any}, x : \hat{a}, \blacksquare_{[]} [\hat{a}] \vdash_c x \Rightarrow \hat{a} \dashv \Theta, \hat{a} : \text{Abs}, x : \hat{a}, \blacksquare_{[]} @ \cdot}{\Theta, \hat{a} : \text{Any}, x : \hat{a} \ [[]\hat{a}] \vdash_c x \Rightarrow []\hat{a} \dashv \Theta, \hat{a} : \text{Abs}, x : \hat{a} @ E} \text{I-TELESCOPE}
 \end{array}$$

As the derivation shows, the algorithmic kinding changes $\hat{a}$ from kind Any to Abs. We do not lose completeness with respect to the declarative type system. For the term $\lambda x. f\ x$, any monotype τ guessed for x must be usable under the modality $[]$, which forces the type τ to have kind Abs.

Skeleton inference. Algorithmic skeleton inference $\Theta_0 \ [p_0] \vdash_m M \Rightarrow P \dashv \Theta_1$ keeps all rules from Section 9.3. It has no ambient effect context. The new rules for operations and handlers follow Figure 10.5, using algorithmic subtyping and threading the output context through handler clauses.

10.3.3 Algorithmic subtyping, consistency, and look

Subtyping. The subtyping judgement has the form $\Theta_0 \ [p_0] \vdash_m^n P \leq Q \dashv \Theta_1 @ E$. The effect context is only needed when $n = \text{ty}$. The new rules are shown in Figure 10.13. The modality sequences in these rules are maximal, and at least one sequence in each conclusion is non-empty. Rule SI-MODCHECK checks the same auxiliary modality transformation judgement as declarative S-MODCHECK and checks the bodies under the target locks and transformed effect context. This auxiliary check uses algorithmic kinding for its Abs case, so it may change kinds in the output context. Rule SI-FORALLL also uses algorithmic kinding to check that the instantiation returned by the look judgement has the kind required by the quantified variable.

Rule SI-MODFUN removes source modalities in function mode. For skeletons, rule SI-MOD removes both modality sequences without comparing them. The full type pass performs the omitted transformation check.

$$\boxed{\Theta_0 [P_0]_m^n P \leq Q \vdash \Theta_1 @ E}$$

$$\text{SI-MODCHECK} \quad \frac{\Theta_0 \vdash (\bar{\nu}, P) \Rightarrow \bar{\mu} @ E \vdash \Theta_1 \quad \bar{\mu}\{\Theta_1\} [P_0]_c^{\text{ty}} P \leq Q \vdash \bar{\mu}\{\Theta_2\} @ \bar{\mu}\{E\}}{\Theta_0 [\bar{\mu}P_0]_c^{\text{ty}} \bar{\nu}P \leq \bar{\mu}Q \vdash \Theta_2 @ E}$$

$$\text{SI-MODFUN} \quad \frac{\Theta_0 \vdash (\bar{\nu}, P) \Rightarrow \langle \rangle @ E \vdash \Theta_1 \quad \Theta_1 [P_0]_{\text{fm}}^{\text{ty}} P \leq Q \vdash \Theta_2 @ E}{\Theta_0 [P_0]_{\text{fm}}^{\text{ty}} \bar{\nu}P \leq Q \vdash \Theta_2 @ E}$$

$$\text{SI-MOD} \quad \frac{m \neq \text{I} \quad \Theta_0 [P_0]_m^{\text{sk}} P \leq Q \vdash \Theta_1}{\Theta_0 [\bar{\mu}P_0]_m^{\text{sk}} \bar{\nu}P \leq \bar{\mu}Q \vdash \Theta_1}$$

$$\text{SI-FORALL} \quad \frac{m \neq \text{I} \quad P_0 \text{ guarded or } P_0 = \mathbb{A} \quad P \leq_m^n P_0 \not\prec^\alpha \Xi; P_1 \quad \Theta_0, \Xi \vdash P_1 : K \vdash \Theta_1 \quad \Theta_1 [P_0]_m^n P[P_1/\alpha] \leq Q \vdash \Theta_2 @ E}{\Theta_0 [P_0]_m^n \forall \alpha^K. P \leq Q \vdash \Theta_2 @ E}$$

$$\boxed{\Theta_0 [P_0] \vdash P \sim Q \vdash \Theta_1}$$

$$\text{U-MODINH} \quad \frac{\Theta_0 [P_0] \vdash P \sim Q \vdash \Theta_1}{\Theta_0 [\mu P_0] \vdash \mu P \sim \mu Q \vdash \Theta_1}$$

$$\text{U-MODSYN} \quad \frac{\Theta_0 [\mathbb{A}.P_0] \vdash P \sim Q \vdash \Theta_1}{\Theta_0 [\mathbb{A}.P_0] \vdash \mu P \sim \mu Q \vdash \Theta_1}$$

$$\text{U-UNIVGHOSTMOD} \quad \frac{\Theta_0 [P_0] \vdash \mathbb{A}.P \sim Q \vdash \Theta_1}{\Theta_0 [\mu P_0] \vdash \mathbb{A}.P \sim \mu Q \vdash \Theta_1}$$

$$\boxed{\Theta \mid \Xi \vdash \hat{\alpha} := P \vdash \Theta'}$$

$$\text{U-ASSIGN-SOLVEM} \quad \frac{\hat{\alpha} \notin \text{ftv}(P) \quad \text{guess}(P) = (\hat{\beta} : \text{Any}; \tau) \quad \Theta_0, \Xi, \hat{\beta} : \text{Any} \vdash \tau : K \vdash \Theta_1}{\Theta_0, \hat{\alpha} : K \mid \Xi \vdash \hat{\alpha} := P \vdash \Theta_1, \hat{\alpha} := \tau}$$

Figure 10.13: Changed algorithmic subtyping and consistency rules for Maetel.

Consistency. Algorithmic consistency keeps the judgement form $\Theta_0 [P_0] \vdash P \sim Q \vdash \Theta_1$. We show changed and new rules in Figure 10.13. As in Section 9.4, the full judgement also carries two sets of bound variables, which we omit here since they are unchanged in the shown rules. Rules U-MODINH and U-MODSYN are the modality counterparts of the corresponding quantifier rules in Fresco. Rule U-UNIVGHOSTMOD allows a universal ghost to match an inherited modality while retaining the ghost in its premise. Rule U-MODINH compares its two modalities up to modality equivalence, which we do not write out here since we have no flexible variables in modalities. The changed assignment rule U-ASSIGN-SOLVEM checks the guessed monotype at the kind of the flexible type variable before assigning it.

The look judgement. The look judgement keeps the form $P \leq_m^n P_0 \not\vdash^\alpha \Xi; P_1$ from Section 9.6. It first collects constraints for α and then solves them to obtain P_1 . Constraint collection keeps the forms $P \leq_m P_0 \not\vdash^\alpha \Xi; C$ and $P \sim P_0 \not\vdash^\alpha \Xi; C$. Constraint solving keeps the form $C; P \rightsquigarrow Q$ with two individual solving judgements of the forms $P \leq_m Q \rightsquigarrow P'$ and $P \sim Q \rightsquigarrow P'$.

$$\boxed{P \leq_m P' \not\vdash^\alpha \Xi; C}$$

$$\text{LS-TELESCOPE} \quad \frac{\rho_1 \neq \cdot \text{ or } \rho_2 \neq \cdot \quad P \leq_m P' \not\vdash^\alpha \Xi; C}{\rho_1\{P\} \leq_m \rho_2\{P'\} \not\vdash^\alpha \Xi; C}$$

$$\boxed{P \sim P' \not\vdash^\alpha \Xi; C}$$

$$\begin{array}{c}
\text{LE-TELESCOPEINH} \quad \frac{\rho \neq \cdot \quad P \sim P' \not\vdash^\alpha \Xi; C}{\rho\{P\} \sim \rho\{P'\} \not\vdash^\alpha \Xi; C} \\
\text{LE-TELESCOPESYN} \quad \frac{\rho \neq \cdot \quad P \sim \mathbb{A}.P' \not\vdash^\alpha \Xi; C}{\rho\{P\} \sim \mathbb{A}.P' \not\vdash^\alpha \Xi; C} \\
\text{LE-UNIVGHOSTTELESCOPE} \quad \frac{\rho \neq \cdot \quad \mathbb{A}.P \sim P' \not\vdash^\alpha \Xi; C}{\mathbb{A}.P \sim \rho\{P'\} \not\vdash^\alpha \Xi; C} \\
\text{LE-FLEXR} \quad \frac{P \neq \alpha \quad \alpha \in \text{ftv}(P) \quad \hat{\gamma} \text{ fresh}}{P \sim \hat{\beta} \not\vdash^\alpha \hat{\gamma} : \text{Any}; \text{eq}(\hat{\gamma})}
\end{array}$$

Figure 10.14: Changed constraint-collection rules for Maetel.

Figure 10.14 shows the changed constraint-collection rules. Rule LS-TELESCOPE replaces rules LS-FORALL and LS-FORALLR. It collects constraints from the bodies of the two telescopes. Rule LE-TELESCOPEINH replaces rule LE-FORALLINH. Rule LE-TELESCOPESYN replaces rule LE-FORALLSYN. Rule LE-UNIVGHOSTTELESCOPE replaces rule LE-UNIVGHOST1. Rule LE-FLEXR declares its fresh flexible variable at kind Any.

$$\boxed{P \leq_m Q \rightsquigarrow \Xi; P'}$$

$$\text{SOLS-TELESCOPE} \quad \frac{\rho_1 \neq \cdot \text{ or } \rho_2 \neq \cdot \quad P \leq_m Q \rightsquigarrow \Xi; P'}{\rho_1\{P\} \leq_m \rho_2\{Q\} \rightsquigarrow \Xi; \rho_1\{P'\}}$$

$$\boxed{P \sim Q \rightsquigarrow \Xi; P'}$$

$$\begin{array}{c}
\text{SOLE-TELESCOPEINH} \quad \frac{\rho \neq \cdot \quad P \sim Q \rightsquigarrow \Xi; P'}{\rho\{P\} \sim \rho\{Q\} \rightsquigarrow \Xi; \rho\{P'\}} \\
\text{SOLE-TELESCOPESYN} \quad \frac{\rho \neq \cdot \quad P \sim \mathbb{A}.Q \rightsquigarrow \Xi; P'}{\rho\{P\} \sim \mathbb{A}.Q \rightsquigarrow \Xi; \rho\{P'\}} \\
\text{SOLE-UNIVGHOSTTELESCOPE} \quad \frac{\rho \neq \cdot \quad \mathbb{A}.P \sim Q \rightsquigarrow \Xi; P'}{\mathbb{A}.P \sim \rho\{Q\} \rightsquigarrow \Xi; \rho\{P'\}}
\end{array}$$

Figure 10.15: Changed constraint-solving rules for Maetel.

Figure 10.15 shows the changed constraint-solving rules. Rule SOLS-TELESCOPE replaces rules SOLS-FORALL and SOLS-FORALLR. It retains the first telescope in the result and recursively solves the two bodies. Rule SOLE-TELESCOPEINH replaces rule SOLE-FORALL. Rule SOLE-TELESCOPESYN replaces rule SOLE-FORALLGHOST. Rule SOLE-UNIVGHOSTTELESCOPE replaces rule SOLE-GHOSTFORALL.

10.4 Extensions and design choices

We discuss some extensions and design choices of Maetel.

10.4.1 Algebraic data types

Maetel supports algebraic data types and pattern matching. This extension poses no particular challenge. Taking pairs as an example, the declarative skeleton inference rules for pair introduction and elimination in Maetel are as follows.

$$\begin{array}{c}
 \text{P-PAIR} \\
 \frac{\Gamma \vdash_I M \Rightarrow P \quad \Gamma \vdash_I N \Rightarrow Q}{\Gamma \vdash_I (M, N) \Rightarrow P \times Q} \\
 \\
 \text{P-PAIRCHECK} \\
 \frac{\Gamma \vdash_c M \Rightarrow P \quad \Gamma \vdash_c N \Rightarrow Q}{\Gamma \vdash_c (M, N) \Rightarrow \mathbb{A}.P \times Q} \\
 \\
 \text{P-CASE} \\
 \frac{\Gamma \vdash_c M \Rightarrow \rho\{\mathbb{A}[(P_1 \times P_2)]\} \quad \Gamma, x : \rho\{\mathbb{A}[(P_1)]\}, y : \rho\{\mathbb{A}[(P_2)]\} \vdash_m N \Rightarrow Q}{\Gamma \vdash_m \text{case } M \text{ of } \{(x, y) \mapsto N\} \Rightarrow Q}
 \end{array}$$

We have two rules for pair introduction, one for inference mode and the other for checking mode. In P-PAIRCHECK, we introduce a universal ghost to cover potential type abstraction, similar to P-ABSCHECK in Section 8.7. We use a case construct to eliminate pairs. We allow the term to be inferred with a pair skeleton under a telescope, followed by an optional universal ghost. The same telescope and universal ghost prefix the skeletons assigned to the two pattern variables. The branch term N has the same mode m as the whole case term.

10.4.2 Global functions are absolute

As with bidirectional typing, we require type annotations for global functions. Moreover, we assume that global functions are always absolute; they are implicitly wrapped into a $[]$ modality if no absolute modality is given. This enables us to give higher-order functions such as `map` their usual types.

10.4.3 Modality elimination within telescopes

As shown by the D-VAR rule in Section 10.2.2, when we use a variable, we only allow implicit elimination of its top-level modalities. We can further generalise this to allow implicit elimination of modalities within the top-level telescope. For example, suppose we have a variable binding $x : \forall a. \langle \text{yield} \rangle (a \rightarrow a)$ in the context with an operation $\text{yield} : \text{Int} \rightarrow 1$. We can write the following program to run x and handle its effects:

$$\text{handle } x \text{ () with } \{\text{return } y \mapsto y, \text{yield } p \ r \mapsto r \text{ ()}\}$$

However, under the current typing rules, x cannot cross the lock $\mathbf{\text{lock}}_{\langle \text{yield} \rangle}$ introduced by the handler, because the modality in its type occurs beneath the quantifier $\forall a$. Extending implicit modality elimination from top-level modalities to top-level telescopes makes this program type check. The top-level telescope of x is $\forall a. \langle \text{yield} \rangle$, whose modality $\langle \text{yield} \rangle$ can be consumed to cross the lock $\mathbf{\text{lock}}_{\langle \text{yield} \rangle}$.

Considering a top-level telescope as a whole would also enable a more powerful subtyping relation. For instance, the subtyping relation

$$\Gamma \vdash_{\text{fm}} \langle \text{yield} \rangle (\forall a. \langle \text{yield} \rangle (1 \rightarrow 1)) \leq 1 \rightarrow 1 @ \text{yield}$$

fails because S-MODFUN first tries to transform the modality $\langle \text{yield} \rangle$ to the identity modality, which is not allowed. If we consider the top-level telescope $\langle \text{yield} \rangle \forall a. \langle \text{yield} \rangle$ as a whole, we can transform it to the identity modality.

We leave a formal treatment of this extension to future work.

10.4.4 Masks

Adding masks is routine. The only challenge is to update the right residual operator $\nu \backslash \mu$ to account for masks, which is already covered in Tang et al. [2025].

10.4.5 Parameterised effects

We ignored parameterised effects in the formal presentation of Maetel. Adding parameterised effects is mostly routine, except for introducing a new challenge to kind inference. In Section 10.3, we always give kind Any to flexible type variables and make them Abs when necessary during type inference. This approach does not break completeness of type inference because when we make a flexible variable Abs, we know this is the only choice. However, with parameterised effects, it may not be the only choice any more. For instance, in order to make the transformation $\langle \text{Gen } \hat{a} \rangle \hat{c} \Rightarrow \langle \text{Gen } \hat{b} \rangle \hat{c}$ hold, we can either make $\hat{c}$ Abs or unify $\hat{a}$ and $\hat{b}$. It would be interesting future work to find a simple solution to this problem without breaking completeness.

10.4.6 Effect variables

We ignored effect variables in the formal presentation of Maetel. There are two ways to add effect variables to Maetel. One approach is to introduce flexible effect variables $\hat{e}$ when instantiating an effect abstraction and solve them during type inference, similar to how we solve flexible type variables. The challenge with this approach is that we may encounter inequality constraints such as $\hat{e} \leq E$ since Maetel has subeffecting, in which case we may not be able to find an optimal solution. Another approach is to never guess effect types out of thin air. Instead, when instantiating an effect abstraction, we decide which effect type to use from known information following what we do for quantifiers and modalities. In the declarative type system, we would also attach tags to effect types. We leave a formal treatment of effect variables to future work.

10.5 Implementation and elaboration

We discuss the implementation and elaboration of Maetel.

10.5.1 Implementation

I do not yet have a full implementation of Maetel. I am working on a prototype implementation of Maetel on top of my implementation of METL [Tang 2025], a surface language with a much simpler bidirectional type inference algorithm introduced in Tang et al. [2025]. Aghilas Y. Boussaa implemented a surface language for modal effect types with ML-like syntax and the type inference algorithm described in this chapter [Boussaa 2026].

10.5.2 Elaboration

Maetel has neither explicit introduction nor elimination for quantifiers and modalities. We can elaborate a Maetel term into a MET term by inserting explicit introduction and elimination forms at the appropriate positions guided by the typing derivation.

In Figure 10.16, we define the elaboration from the declarative type system of Maetel in Section 10.2 to MET. The judgement $\Gamma \vdash_m M : A @ E \dashrightarrow M'$ elaborates the Maetel term M to the MET term M' . The key intuition is that we insert modality elimination forms when a variable is bound, and insert modality introduction forms when a variable is used. We also insert type abstraction and modality introduction forms in the D-TELESCOPE rule, and insert instantiation, together with modality introduction and elimination forms, in the D-SUB rule, which uses a separate elaboration on subtyping relations defined later.

For simplicity, we follow $\text{MET}(\mathcal{X})$ in treating $\text{let}_v \text{ mod}_\mu x = V$ in W as a value when both V and W are values in order to make sure the elaboration preserves values. The operational semantics is extended correspondingly to use value normal forms as in $\text{MET}(\mathcal{X})$. We also extend telescope application to values to insert type abstractions and modality introductions:

$$\begin{aligned} \cdot\{V\} &= V \\ (\forall \alpha^K. \rho)\{V\} &= \Lambda \alpha^K. \rho\{V\} \\ (\mu \rho)\{V\} &= \text{mod}_\mu \rho\{V\} \end{aligned}$$

The auxiliary function $\text{unvar}(x; A; M)$ eliminates all top-level modalities of a variable x of type A . The auxiliary function $\text{unvar}_\mu(x; A; M)$ additionally assumes the variable x is annotated with the modality μ in the context.

$$\begin{aligned} \text{unvar}(x; A; M) &= \text{unvar}_{\langle \rangle}(x; A; M) \\ \text{unvar}_v(x; \mu A; M) &= \text{let}_v \text{ mod}_\mu x = x \text{ in } \text{unvar}_{v \circ \mu}(x; A; M) \\ \text{unvar}_v(x; A; M) &= M \quad \text{if } A \text{ non-modal} \end{aligned}$$

Rule D-VAR reintroduces the modalities in the type B returned by the across judgement. Rules D-ABS and D-ABSANNO use unvar to immediately eliminate the modalities of the bound variable x . In D-HANDLE, MET already binds the return variable as $x : \langle D \rangle A$, so we use $\text{unvar}_{\langle D \rangle}$. The continuation has no top-level modality and needs no modality elimination.

Figure 10.17 elaborates a subtyping relation in Figure 10.7 to a meta-level function between MET values. The judgement has the form $\Gamma \vdash_m A \leq B @ E \dashrightarrow f$, where f is a meta-function that transforms a MET value of type A into one of type B . We write $\lambda V.W$ for defining such a meta-level function. It binds the metavariable V in W . We write $f(V')$ for applying a meta-function. For simplicity, when we write T or φ in black, we implicitly erase their tags. We write $\text{let mod}_{\bar{\mu}} x = V$ in M as a shorthand for a sequence of modality eliminations.

Rules S-INFER and S-CHECK leave the value unchanged. Rule S-FORALLL inserts type application, and S-FORALLR inserts type abstraction. Rule S-ARROW upcasts the function result and leaves the argument unchanged. Rule S-MODCHECK eliminates the original modalities and introduces the new modalities for the recursively elaborated term. We insert an extra modality elimination for the identity modality to ensure V is fully evaluated before passing to f even when $\bar{v}$ is empty. Otherwise, we cannot guarantee that any value V can be accessed under the $\bar{\mu}$ modalities since our extended definition

$$\boxed{\Gamma \vdash_m M : A @ E \dashrightarrow M'}$$

$$\begin{array}{c}
\text{D-UNIT} \\
\hline
\Gamma \vdash_I () : 1 @ E \dashrightarrow ()
\end{array}
\quad
\begin{array}{c}
\text{D-VAR} \\
\hline
\Gamma_0 \vdash A \Rightarrow (\text{locks}(\Gamma_1), B) \quad B = \bar{\zeta} B_0 \quad B_0 \text{ non-modal} \\
\hline
\Gamma_0, x : A, \Gamma_1 \vdash_I x : B @ E \dashrightarrow \bar{\zeta}\{x\}
\end{array}$$

$$\begin{array}{c}
\text{D-ANNO} \\
\hline
\Gamma \vdash_c M : A @ E \dashrightarrow M' \\
\hline
\Gamma \vdash_I (M : A) : A @ E \dashrightarrow M'
\end{array}
\quad
\begin{array}{c}
\text{D-SUB} \\
\hline
\Gamma \vdash_I M : A @ E \dashrightarrow M' \quad \Gamma \vdash_m A \leq B @ E \dashrightarrow f \\
\hline
\Gamma \vdash_m M : B @ E \dashrightarrow \begin{cases} f(M') & \text{if } M' \text{ is a value} \\ \text{let } x^A = M' \text{ in } f(x) & \text{otherwise} \end{cases}
\end{array}$$

$$\begin{array}{c}
\text{D-FREEZE} \\
\hline
\Gamma \vdash_I M : A @ E \dashrightarrow M' \\
\hline
\Gamma \vdash_c [M] : A @ E \dashrightarrow M'
\end{array}
\quad
\begin{array}{c}
\text{D-TELESCOPE} \\
\hline
\rho\{\Gamma\} \vdash_c V : A @ \rho\{E\} \dashrightarrow V' \\
\hline
\Gamma \vdash_c V : \rho\{A\} @ E \dashrightarrow \rho\{V'\}
\end{array}$$

$$\begin{array}{c}
\text{D-ABS} \\
\hline
\Gamma, x : A \vdash_{m-} M : B @ E \dashrightarrow M' \\
\hline
\Gamma \vdash_m \lambda x. M : A \rightarrow B @ E \dashrightarrow \lambda x^A. \text{unvar}(x; A; M')
\end{array}$$

$$\begin{array}{c}
\text{D-ABSANNO} \\
\hline
\Gamma, x : A \vdash_{m-} M : B @ E \dashrightarrow M' \\
\hline
\Gamma \vdash_m \lambda x^A. M : A \rightarrow B @ E \dashrightarrow \lambda x^A. \text{unvar}(x; A; M')
\end{array}$$

$$\begin{array}{c}
\text{D-APP} \\
\hline
\Gamma \vdash_c N \Rightarrow P \quad \Gamma \vdash P \blacktriangleright A \\
\Gamma \vdash_{Fm} M : A \rightarrow B @ E \dashrightarrow M' \quad \Gamma \vdash_c N : A @ E \dashrightarrow N' \\
\hline
\Gamma \vdash_m M N : B @ E \dashrightarrow M' N'
\end{array}$$

$$\begin{array}{c}
\text{D-DO} \\
\hline
\Sigma \ni \ell : A \rightarrow B \quad \Gamma \vdash_c M : A @ \ell, E \dashrightarrow M' \\
\hline
\Gamma \vdash_I \text{do } \ell M : B @ \ell, E \dashrightarrow \text{do } \ell M'
\end{array}$$

$$\begin{array}{c}
\text{D-HANDLE} \\
\hline
D = \bar{\ell}_i \quad \Gamma, \blacksquare_{\langle D \rangle} \vdash_I M : A @ D, E \dashrightarrow M' \quad \Gamma, x : \langle D \rangle A \vdash_m N : B @ E \dashrightarrow N' \\
[\Sigma \ni \ell_i : A_i \rightarrow B_i \quad \Gamma, p_i : A_i, r_i : B_i \rightarrow B \vdash_m N_i : B @ E \dashrightarrow N'_i]_i \\
\hline
\Gamma \vdash_m \text{handle } M \text{ with } \{\text{return } x \mapsto N\} \uplus \{\ell_i \ p_i \ r_i \mapsto N_i\}_i : B @ E \\
\hline
\dashrightarrow \text{handle } M' \text{ with } \{\text{return } x \mapsto \text{unvar}_{\langle D \rangle}(x; A; N')\} \uplus \{\ell_i \ p_i \ r_i \mapsto \text{unvar}(p_i; A_i; N'_i)\}_i
\end{array}$$

Figure 10.16: Elaboration from Maetel to MET.

of values contains $\text{let}_v, \text{mod}_\mu x = V_1 \text{ in } V_2$. Rule S-ModFun only eliminates the original modalities.

$$\boxed{\Psi \mid \Gamma \vdash_m T \leq A @ E \dashrightarrow f}$$

$$\begin{array}{c}
\text{S-INFER} \quad \frac{\cdot \mid \Gamma \vdash T \sim A}{\cdot \mid \Gamma \vdash_I T \leq A @ E \dashrightarrow \lambda V.V} \quad \text{S-CHECK} \quad \frac{T, A \text{ guarded} \quad \Psi \mid \Gamma \vdash T \sim A}{\Psi \mid \Gamma \vdash_c T \leq A @ E \dashrightarrow \lambda V.V} \\
\\
\text{S-FORALLR} \quad \frac{\Psi \mid \Gamma, \alpha : K \vdash_c T \leq A @ E \dashrightarrow f}{\Psi \mid \Gamma \vdash_c T \leq \forall \alpha^K. A @ E \dashrightarrow \lambda V. \Lambda \alpha^K. f(V)} \\
\\
\text{S-ARROW} \quad \frac{\Psi_1 \mid \Gamma \vdash T_1 \sim A_1 \quad \Psi_2 \mid \Gamma \vdash_m T_2 \leq A_2 @ E \dashrightarrow f}{\Psi_1, \Psi_2 \mid \Gamma \vdash_{Fm} T_1 \rightarrow T_2 \leq A_1 \rightarrow A_2 @ E \dashrightarrow \lambda V. \lambda y^{A_1}. \text{let } x^{T_2} = V y \text{ in } f(x)} \\
\\
\text{S-FORALLL} \quad \frac{\begin{array}{c} m \neq I \quad \Psi' \text{ fresh} \quad \Gamma \vdash \varphi \# \Psi' : K \\ \Psi, \Psi' \mid \Gamma \vdash_m T[\varphi/\alpha] \leq A @ E \dashrightarrow f \end{array}}{\Psi \mid \Gamma \vdash_m \forall^\delta \alpha^K. T \leq A @ E \dashrightarrow \lambda V. f(V \varphi)} \\
\\
\text{S-MODCHECK} \quad \frac{\Gamma \vdash (\bar{v}, T) \Rightarrow \bar{\mu} @ E \quad \Psi \mid \bar{\mu}\{\Gamma\} \vdash_c T \leq A @ \bar{\mu}\{E\} \dashrightarrow f}{\Psi \mid \Gamma \vdash_c \bar{v}^{\bar{\delta}} T \leq \bar{\mu} A @ E \dashrightarrow \lambda V. \text{let mod}_{\langle \rangle} x = \text{mod}_{\langle \rangle} V \text{ in let mod}_{\bar{v}} x = x \text{ in } \bar{\mu}\{f(x)\}} \\
\\
\text{S-MODFUN} \quad \frac{\Gamma \vdash (\bar{v}, T) \Rightarrow \langle \rangle @ E \quad \Psi \mid \Gamma \vdash_{Fm} T \leq A @ E \dashrightarrow f}{\Psi \mid \Gamma \vdash_{Fm} \bar{v}^{\bar{\delta}} T \leq A @ E \dashrightarrow \lambda V. \text{let mod}_{\bar{v}} x = V \text{ in } f(x)}
\end{array}$$

Figure 10.17: Elaboration of Maetel subtyping to MET.

Epilogue

11 Related work

Thus we have achieved a thousand things in each genre, and one of the principal occupations of science and philosophy, in order to understand the reasons for them, is to discover their origin and primitive cause. This is what must be called “type” in architecture, as in every other field of inventions and human institutions.

—Quatremère de Quincy (1825)

We start this chapter with a direct comparison of the types of the same programs in different programming languages, including our surface language Maetel and two other research languages: Koka [Leijen 2026], which has a row-based effect system, and Effekt [Brachthäuser et al. 2026], which has a capability-based effect system. Then we discuss related work on effect systems and type inference.

11.1 Programming with Maetel, Koka, and Effekt

The encodings in Chapter 7 encode the core calculi of Koka and Effekt into $\text{MET}(\mathcal{X})$, a framework based on the core calculus of Maetel. However, these encodings focus on the expressive power and make significant use of effect variables in $\text{MET}(\mathcal{X})$. In practice, as we see in Chapter 3, effect variables are rarely used in Maetel. This section provides a more direct comparison of how the same programs have different types in Maetel, Koka, and Effekt. These examples reflect common programming patterns and reveal the essential differences between the three languages for programmers.

11.1.1 Invoking and handling effects

An effect system tracks which effects are invoked. Consider the function `genInc x = do yield (x + 1)` from Section 3.3, which uses the `yield` operation. Its types in Maetel, Koka, and Effekt are as follows.

```
genInc : [Gen Int](Int → 1)           # Maetel
genInc : forall<e>. (x : int) → <gen<int>|e> () # Koka
def genInc(x : Int) : 1 / { Gen[Int] }  # Effekt
```

Both Maetel and Effekt support effect subtyping, which enables us to apply `genInc` with other effects. Koka does not support effect subtyping and uses an effect variable `e` for modularity. Koka supports a special typing rule which implicitly inserts effect variables on function types in covariant position. Consequently, we can in fact simply write the following type in Koka.

```
genInc : (x : int) → (gen<int>) () # Koka
```

An effect system tracks which effects are handled. Recall the `asList` handler from Section 3.4. Its types in Maetel, Koka, and Effekt are as follows.

```
asList : forall [a] . [](<Gen a>(1 → 1) → List a) # Maetel
asList : forall<a,e>. (action : () → <gen<a>|e> ()) → e list<a> # Koka
def asList[A]{ f: 1 ⇒ List[A] / { Gen[A] } }: List[A] / {} # Effekt
```

Both Maetel and Effekt allow the argument of `asList` to use the ambient effects in addition to `Gen a`. In Koka, we must make the argument polymorphic over other effects.

11.1.2 Data types and higher-order functions

An effect system should be compatible with algebraic data types. Consider a pair of `genInc` functions. In Maetel, we can choose to either share a single absolute modality between both components, or have a separate absolute modality for each component.

```
pa : [Gen Int]((Int → 1) * (Int → 1))
pa : ([Gen Int](Int → 1)) * ([Gen Int](Int → 1))
```

In Koka, we can choose to use one or two effect variables:

```
pa : forall<e>. ((x:int) →<gen<int>|e> (), (y:int) →<gen<int>|e> ())
pa : forall<e,e1>. ((x:int) →<gen<int>|e> (), (y:int) →<gen<int>|e1> ())
```

In Effekt, we cannot express this pair of functions directly, as functions are second-class. We must box the functions to make them first-class.

```
pa : Tuple2[Int ⇒ 1 / {Gen[Int]} at {}, Int ⇒ 1 / {Gen[Int]} at {}]
```

The syntax `at {}` means that the preceding function type `Int ⇒ 1 / {Gen[Int]}` is boxed with no captured capability.

An effect system should be compatible with higher-order functions. Consider the standard sequential composition function `compose f g x = g (f x)`. Its type in Maetel states that the function itself is pure:

```
compose : forall a b c . []((a → b) → (b → c) → (a → c)) # Maetel
```

Its type in Koka uses an effect variable:

```
compose : forall<a,b,c,e>. (f: (a) →e b) → (g: (b) →e c) → ((x:a) →e c)
```

In Effekt, since functions are second-class, we must either switch to an uncurried version or box the result with capability variables:

```
def composeUncurried[A, B, C](x: A){ f: A ⇒ B }{ g: B ⇒ C }: C / {}
def compose[A, B, C]{ f: A ⇒ B }{ g: B ⇒ C }: A ⇒ C at {f, g} / {}
```

11.2 Effect systems

There is a large amount of work on effect systems in the literature. Here we focus more on the work targeted at algebraic effects and handlers.

Row-based effect systems. We use the term *row-based effect systems* to refer to the class of effect systems that associate effects with function types. The structure of effect types is usually based on row types. Links [Hillerström and Lindley 2016] uses Rémy-style row polymorphism with presence types [Rémy 1994], whereas Koka [Leijen 2017] and Frank [Lindley et al. 2017] use scoped rows [Leijen 2005], which allow duplicated labels. Many row-based effect systems support subeffecting. Eff [Bauer and Pretnar 2014; Pretnar 2014] is equipped with an effect system with both effect variables and subeffecting based on the type inference and elaboration described in Karachalias et al. [2020]. The effect system of Helium [Biernacki et al. 2020] is based on finite sets, offering a natural subeffecting relation corresponding to set inclusion. Tang et al. [2024] propose a calculus for effect handlers with effect polymorphism and subeffecting via qualified types [Jones 1994; Morris and McKinnon 2019]. MET adopts Leijen-style scoped rows, but also allows operation labels to be absent in the spirit of Rémy-style presence types. We expect that other row-based effect systems mentioned here can also be encoded in MET($\mathcal{X}$) by instantiating the effect structure appropriately.

The Frank language. Frank [Lindley et al. 2017; Convent et al. 2020] also adopts a row-based effect system and introduces several novel notions for getting rid of effect variables. The design of modal effect types is inspired by Frank. Our ambient effect contexts are inspired by the notion of *ambient abilities* in Frank, and our relative modalities are inspired by the notion of *adjustments* in Frank. Frank solves the eee problem in Section 1.1 via syntactic sugar that implicitly inserts effect variables into higher-order programs. This approach works in practice, but it is a fragile syntactic abstraction that can be leaked to programmers in places such as error messages. Syntactic sugar is

not satisfactory from a theoretical perspective either. In contrast, **MET** exploits modal types to robustly capture the essence of modular effect programming without effect polymorphism. Our calculus **System $F^{1\epsilon}$** in Section 7.4 characterises the core idea of Frank. We encode **System $F^{1\epsilon}$** into **MET** without effect variables. Another difference is that Frank’s adjustments are second-class and only appear beside argument types of functions, while relative modalities of **MET** are first-class and can appear anywhere. Frank also supports *adaptors*, which provide more ways to manipulate the ambient effect context than the relative modalities of **MET**, which only add or remove effects from the ambient effect context. We expect relative modalities to extend readily to encompass the full power of adaptors.

Capability-based effect systems. Some recent effect systems use capabilities to model effects and offer a form of effect polymorphism through capability passing. Different variations diverge on when capability sets appear in types. **Effekt** [Brachthäuser et al. 2020, 2022] uses second-class functions and only attaches capability sets to types when boxing functions to make them first-class. **CC_{<:□}** [Boruch-Gruszecki et al. 2023] and **Capless** [Xu et al. 2025], the foundations for capture tracking in Scala 3, always annotate every type with its capability set and use subtyping and syntactic sugar to simplify capability sets. Our notion of ambient effect contexts is also inspired by the view of effects as capabilities provided by the context. Capability-based effect systems also solve the *eee* problem. The argument of a higher-order function may use any capabilities provided by the context without explicitly mentioning them in its type. One main difference is that capability-based effect systems either make functions second-class or attach capability sets to every type to prevent capabilities from escaping. The latter requires some form of lightweight dependent types to mention capability variables in types. In contrast, **MET** works more smoothly with first-class functions. Another difference is that capability-based effect systems target named handlers, in which operations are dispatched to a specific named handler, whereas **MET** uses Plotkin and Pretnar [2013]-style handlers, in which operations are dispatched to the first matching handler in the dynamic context.

The purity modality. Choudhury and Krishnaswami [2020] propose to use the necessity modality to recover purity from an effectful calculus. This modality is similar to our empty absolute modality.

A modal type system for benign effects. Nanevski [2004] proposes a modal type system for benign effects in Section 4.3. We refer to this system as **MTBE**. **MTBE** supports local effects and indexes the standard necessity modality $\Box$ with effects for effect tracking.

In MTBE, a type $\Box_E A$ means a computation which returns a value of type A and may perform effects in E . This indexed necessity modality is similar to the absolute modality of MET . The key difference between MTBE and MET is that MTBE has no notion of ambient effect context. MTBE requires functions to be pure: every function type must specify all the effects it may perform via a box. In contrast, $\text{MET}(\mathcal{X})$ allows a function to perform any effects from the ambient effect context. In order to apply a function to effectful arguments in MTBE, we must specify what effects may be performed in the type. This requires parametric effect polymorphism in order to support arbitrary effectful arguments. Moreover, MTBE does not support relative modalities as relative modalities are intimately tied to the notion of ambient effect contexts.

Effectful contextual modal type theory. Zyuzin and Nanevski [2021] propose effectful contextual modal type theory (ECMTT), which extends the *contextual necessity modality* [Nanevski et al. 2008] to track contexts of effectful operations. Similar to MTBE, ECMTT also lacks the notion of ambient effect contexts and thus cannot solve the eee problem.

Abstracting effect systems. Morris and McKinnon [2019] propose a general framework that abstracts different styles of row types. Following this idea, Yoshioka et al. [2024] study different treatments of effect collections in row-based effect systems. They propose a parameterised core calculus, λ_{EA} , whose effect types can be instantiated to various kinds of sets and rows. λ_{EA} focuses on row-based effect systems and thus cannot encode capability-based effect systems. We follow λ_{EA} in parameterising our framework $\text{MET}(\mathcal{X})$ over different treatments of effect collections. We make use of modalities to decouple effect tracking from function types, enabling the encodings of both row-based and capability-based effect systems.

Local effects. Local labels in $\text{MET}(\mathcal{X})$ allow us to introduce fresh effects locally. They are useful for solving the effect pollution and accidental handling problems [Biernacki et al. 2019; Convent et al. 2020; Zhang and Myers 2019]. There are various local effect formalisms in the literature [Biernacki et al. 2019; de Vilhena and Pottier 2023; Jourde et al. 2022]; most are based on dynamic generation of fresh effect names, whereas the calculus of Biernacki et al. [2019] is based on effect coercions. We conjecture that local labels of $\text{MET}(\mathcal{X})$ are as expressive as these formalisms.

11.3 Type inference

Type inference for first-class polymorphism (FCP) spans a wide design space. We first discuss type inference approaches to semi-explicit FCP. Then we organise the discussion of type inference for implicit FCP around the two central questions we care about when designing type inference for FCP: how to mix different directions of information flow, and how to represent unknown type information beyond unification variables.

Semi-explicit polymorphism. PolyML [Garrigue and Rémy 1999] first introduces the notion of semi-explicit polymorphism. PolyML separates polymorphic types from first-class monomorphic types, and introduces a type-level box operator to make polymorphic types first-class. Boxed polymorphic types require explicit introduction, which needs a polymorphic type annotation, and explicit elimination, which needs no type annotation. OmniML [O’Brien et al. 2025] extends PolyML to enable more flexible information flow of boxed polymorphic types, allowing the type annotation on the introduction of a boxed polymorphic type to be omitted when it can be determined from the context. Unlike PolyML and OmniML, Fresco targets implicit FCP with no boxes for polymorphic types, and does not require explicit introduction and elimination of polymorphic types. FreezeML [Emrich et al. 2020] also has no type-level boxes. Instead, FreezeML introduces a term-level freezing operator that prevents a term variable from being implicitly instantiated. Our freezing operator is directly inspired by FreezeML. Compared to Fresco, FreezeML makes no use of bidirectional information flow to infer polymorphic types.

Mixed directions of information flow. The line of work closest to Fresco does type inference for FCP by considering information flow between functions and arguments. We discuss the limitations of these approaches in terms of mixed information flow and explain why they fail to type check our motivating example in Chapter 8. Boxy types [Vytiniotis et al. 2006] merge the checking and inference judgements of bidirectional typing into a single judgement in which “boxes” mark the parts of a type that must be inferred, and share with Fresco the principle that polymorphism should never be guessed. In the core system, information only flows from functions to arguments. They proposed several ad hoc rules to allow mixing of information flow, but none of these rules allow information to flow from one argument to a sibling argument, so challenge (2) still fails. GI [Serrano et al. 2018] works in the opposite direction: it treats an n -ary application as a whole and instantiates a variable with polytypes only when the variable is *guarded* under a type constructor in some argument type. GI does not allow information to flow from functions to arguments, so challenge (2) fails. Similarly,

HMF [Leijen 2008] does not allow information to flow from functions to arguments as robustly as Fresco does, so challenge (2) fails. QuickLook [Serrano et al. 2020] combines the two directions by first having a quick look at the arguments of an application spine to guide the polymorphic instantiation of the function. Then the type information of the function is propagated back into the arguments. As discussed in Chapter 8, although QuickLook addresses challenge (2), it fails to address challenge (1) because it only learns from an argument when the argument type is guarded, following GI. However, in our example, the argument `churchIds` has the unguarded type `CList` ($\forall a. a \rightarrow a$). Moreover, the quick look process itself is a deliberately shallow algorithmic pass aimed at easy implementation in GHC. Skeleton inference of Fresco provides a more systematic and precise way to collect information from arguments.

Local type inference. The line of work on local type inference also cares about information flow. As its name suggests, local type inference (LTI) [Pierce and Turner 2000] combines bidirectional checking with *local* synthesis of type arguments. However, LTI only allows information flow from arguments to functions, and propagates information in an all-or-nothing fashion. Coloured local type inference (CLTI) [Odersky et al. 2001] lifts the second limitation by propagating partial types, with a colour on every type component recording whether it is inherited or synthesised. Since everything is coloured, CLTI is not compatible with unification of monotypes where the direction of information flow of monotypes is left open until a solution is found. Fresco works well with monotype unification by only ascribing colours to universal quantifiers. As discussed in Section 8.1.6, both CLTI and Boxy types also restrict polymorphic instantiation more than Fresco does, requiring every appearance of a guessed instantiation to be determined. Local Contextual Type Inference (LCTI) [Xue et al. 2026] redesigns LTI on top of contextual typing [Xue and Oliveira 2024], using modes that reflect program structure to guide instantiation. LCTI lacks the colours used in Fresco and thus cannot represent fine-grained information flow, where only part of the type information flows from an argument to a function while the rest flows back. Moreover, LCTI never instantiates the type of an argument and is sensitive to argument order. In our example, `churchIds` can guide the instantiation of `churchMap`, but the polytype learned this way is not used when checking the sibling lambda, which fails to solve challenge (2).

Beyond unification variables. Every inference algorithm must handle type information that is not yet known. For monotypes, unification variables together with a simple solving strategy suffice: equality constraints may be solved eagerly, in any order, without losing generality. This is not enough for general FCP, or any other scenario where the

solution does not form an equivalence class. Committing to an instantiation too early can exclude valid typings, so the order of solving starts to matter. To address this, people have proposed different *placeholders* to represent what is known about an unknown type, and/or designed smart *solving orders* to delay work on an unknown type until it becomes determined. ATIA [Morris 2026] interleaves an Algorithm-M-style traversal that pushes expected types towards the leaves with constraint solving, and extends the syntax of types with *pseudotypes* $[S]P$ that stand for the result of an unknown sequence of instantiations. Pottier and Régis-Gianas [2006] stratify type inference for GADTs into a constraint-based core preceded by a *shape* inference phase, where shapes are types with holes that propagate the information contained in explicit annotations. In a dependently typed setting with implicit arguments, FCIF [Kovács 2020] elaborates into a core theory with strictly curried telescope function types, so that types themselves can defer decisions about inserting implicit abstractions and applications. Instead of enriching the placeholders, its successor DOE [Kovács 2026] postpones any checking problem whose expected type is an undetermined metavariable and resumes it once the metavariable is solved. OmniML [O’Brien et al. 2025] similarly suspends constraints until the *shape* of a type is uniquely determined by the context, making the order of information flow dynamic rather than fixed. Parreaux [2026] proposes to go further and decouple such resolution steps from type inference altogether. None of the approaches described above leak placeholders into the surface syntax, in contrast to approaches that enrich the type language itself [Le Botlan and Rémy 2003; Leijen 2009; Parreaux et al. 2024]. Most of the above approaches also emphasise describing their algorithms with their own placeholders and solving strategies, with no declarative specification. Leijen and Ye [2025] embrace this trend and argue that specifications should move *closer* to algorithms, rephrasing Hindley-Milner as inference under a prefix. Like the placeholders in the above approaches, skeletons and ghosts of Fresco also represent unknown type information for polymorphism. In contrast to these approaches, Fresco insists on a declarative specification of the type system with soundness and completeness results. Fresco gives a declarative specification of how skeletons are inferred.

12 Conclusion

One must imagine Sisyphus happy.

—Albert Camus (1942)

The main goal of this thesis is to design an effect system for practical languages to use. Much previous work has been done on effect systems, but a main challenge these systems face, as introduced in Section 1.1, is the *eee* problem of having to thread effect variables through types for modularity. Some recent approaches aim to solve the *eee* problem, but they have one or more of the following caveats: they rely on fragile syntactic sugar, sacrifice first-class functions, require type-level capabilities, or are restricted to named effect handlers. This thesis avoids all these caveats, supporting first-class higher-order functions while preserving their usual types, supporting Plotkin and Pretnar [2013]-style dynamic effect handlers, and only requiring the uses and handling of effects to be annotated in most cases.

This thesis does not achieve the above by adopting ad hoc syntactic sugar or introducing mechanisms for binding and tracking capabilities in types. Instead, this thesis develops a new theoretical foundation for effect systems, modal effect types, inspired by multimodal type theory. We observe that, when viewing effects as provided by the surrounding context, the notion of ambient effect contexts gives a natural solution to the *eee* problem. However, the entanglement of effects with function types in traditional effect systems prevents us from benefiting from this observation. We decouple effects from function types via modalities, with absolute modalities denoting uses of effects and relative modalities denoting changes (handling) of effects. Part I presents the core calculus MET that formalises modal effect types with many extensions.

Decoupling effects from function types enables us to unify different styles of effect systems by encoding them into a single target. Part II develops such a uniform framework $\text{MET}(\mathcal{X})$ and presents compositional encodings of different effect systems into $\text{MET}(\mathcal{X})$.

Type inference for modal types can be challenging when we want to keep their introduction and elimination implicit, while using them as first-class types for implicit instantiation, similar to the notion of first-class polymorphism. Part III presents a novel

type inference approach for first-class polymorphism, and extends it to infer modal effect types in a language *Maetel*, whose tutorial is given in Chapter 3.

A general takeaway from this thesis is the idea of decoupling programming language features. Modal effect types decouple effects from function types. This idea has been applied in many other contexts. For instance, call-by-push-value (CBPV) [Levy 2004] decouples thunking and forcing from function abstraction and application.

12.1 Future work

This thesis introduces a novel effect system with a new theoretical foundation, opening up many directions for future work. Some of them have been discussed in previous chapters. We summarise a few promising directions here, grouped by topic.

12.1.1 Modal types

Fitch-style modality elimination. MET follows multimodal type theory to use let-style modality elimination. We must bind a value to a variable in order to eliminate its modality. In contrast, with Fitch-style modality elimination [Clouston 2018], we can eliminate the modality of a value in place. Fitch-style modality elimination is more convenient to use as shown in the related work section of Gratzer et al. [2022], but has extra requirements on the mode theory as discussed in Section 7.5.2. The elaboration procedure of *Maetel* automatically inserts let-style modality elimination, which gives us similar convenience to Fitch-style modality elimination. It remains an open question whether we can translate between let-style modality elimination and Fitch-style modality elimination, and if so, whether such translations are local or global. It is also interesting future work to extend MET with Fitch-style modality elimination.

Value restriction of modal type systems. We adopt the value restriction in MET to ensure effect safety as discussed in Section 4.5. Modality introduction and elimination only apply to values. In a recent extended abstract [Lorenzen et al. 2026], we show that the value restriction is a common trick to ensure type soundness of modal type systems with call-by-value semantics. However, this extended abstract only discusses a collection of examples of modal type systems that adopt the value restriction; it does not give a general characterisation of the conditions under which the value restriction is necessary and sufficient. Exploring such conditions is interesting future work.

Relaxing the value restriction. An observation related to the above is that we do not have to adopt the value restriction for all modalities in MET. In particular, for

absolute modalities $[E]$, the term $\text{mod}_{[E]} M$ is safe for any term M as long as the current ambient effect context F is larger than E , i.e., $E \leq F$. We have no risk of leaking effects from M , since the ambient effect context provides every effect that M requires. It is straightforward to extend **MET** with a relaxed form of value restriction that allows such cases. It is more interesting to explore if similar situations exist for other modal type systems, and if we can give a formal general characterisation of such a relaxed value restriction.

Denotational semantics of **MET.** We present only an operational semantics of **MET** in this thesis. A promising approach to developing denotational semantics of **MET** is to follow the categorical semantics of multimodal type theory [Gratzer et al. 2021; Gratzer 2023].

Submoding. **MET** extends the mode theory of MTT from a 2-category to a double category with a partial order between modes for subeffecting. However, in addition to being a double category, **MET** also requires some extra ad hoc properties of the double category. For instance, as shown by Lemma 29, horizontal morphisms act functorially on vertical morphisms. Future work could explore the algebraic structure that more precisely describes the mode theory with submoding as in **MET**. A possible direction is to consider fibrations. With the right algebraic structure for mode theories with submoding, a next step is to generalise **MET** to a calculus parameterised by such mode theories and instantiate this calculus differently to obtain different modal type systems.

More encodings. We discussed some extensions to the framework $\text{MET}(\mathcal{X})$ and the encodings that these extensions would enable in Section 7.5.3. Other interesting encodings include encoding different formalisms of local effects [Biernacki et al. 2019; de Vilhena and Pottier 2023]; encoding simply-typed **MET** back into System F^{1e} introduced in Section 7.4; and encoding between capturing types [Boruch-Gruszecki et al. 2023; Xu et al. 2025] and modal effect types.

12.1.2 Effect types

Higher-order effects. We discussed in Section 3.9 that the main situation where we need effect variables in **MET** is to define higher-order effects that are parameterised by the effects that their arguments or results may perform. A future extension is to support modular higher-order effects in **MET** without effect variables. A promising idea, as we discussed in an extended abstract [Boussaa et al. 2026], is to introduce an order on effects in effect contexts which characterises the order of handler stacks, so that we

can prevent higher-order operations of outer handlers from taking arguments that use effects of inner handlers. Another promising idea is to adopt the structural approach to higher-order effects [Bosman et al. 2024; Yang and Wu 2026; van den Berg and Schrijvers 2024; Yang 2024; Yang and Wu 2023; Yang et al. 2022] which extends the semantics of effect handlers to enable effects used by arguments of higher-order operations to be handled.

Control-flow linearity. We proposed *control-flow linearity* [Tang et al. 2024] which extends effect types to track whether linear resources are used in their continuations or not, resolving the soundness issue of combining linear resources with multi-shot effect handlers. We formalised control-flow linearity in a conventional effect system with a linear type system that uses kinds to specify linearity of types. A natural direction for future work is to extend modal effect types with control-flow linearity. More interestingly, we can use a modal type system for linearity, and explore how the modalities for effects and linearity interact with each other. Modal type systems are also used for tracking other properties of resources, such as uniqueness and locality [Lorenzen et al. 2024]. It is also interesting future work to explore how to combine these modal type systems with modal effect types without breaking their safety guarantees.

Abstraction safety. In MET, we only allow transformations of the form $\langle D \rangle \Rightarrow \langle D \rangle @ E$ for relative modalities to guarantee abstraction safety. Intuitively, abstraction safety means that effects not mentioned in the types will not be handled. This is demonstrated by the example in Section 3.6. It would be interesting future work to formally characterise the notion of abstraction safety for modal effect types. A promising direction is to define a logical relation for modal effect types following the work on parametricity for effect polymorphism [Biernacki et al. 2018, 2020; Zhang and Myers 2019]. It would also be interesting to explore the general condition for any effect structure to have abstraction safety in $\text{MET}(\mathcal{X})$.

Currying and n-ary functions. The classical type for a curried map function in a conventional effect system is:

$$\text{map} : \text{forall } a \ b \ e . (a \rightarrow_{\{e\}} b) \rightarrow \text{List } a \rightarrow_{\{e\}} \text{List } b$$

The first and last function arrows are annotated with the same effect variable e , while the middle function arrow is pure. The preferred type for it in MET is:

$$\text{map1} : \text{forall } a \ b . (a \rightarrow b) \rightarrow \text{List } a \rightarrow \text{List } b$$

Its interpretation differs slightly from the classical type: the middle function arrow is not pure, but may also perform effects from the ambient effect context. This is consistent with the fact that the above classical type is not a valid type in $\text{System } F^{1e}$ introduced in Section 7.4.1, so it cannot be precisely encoded in MET without effect variables. In order to express the fact that the middle function arrow is pure in MET, we need to use effect variables as shown in the encoding of $\text{System } F^e$ defined in Section 7.2.2.

```
map2 : forall a b e . []([e](a → b) → [e](List a → List b))
```

The differences between the above two types in MET are only observable when we do partial application `map f` under an empty effect context with an effectful argument `f`. We cannot do such partial application with an effectful argument function using `map1`, unless we η -expand this partial application as `fun xs → map1 f xs`. One lightweight solution to avoid writing η -expansions explicitly is to use n -ary functions as in the Frank language [Lindley et al. 2017; Convent et al. 2020], where a function type $A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A_n \rightarrow B$ is not a curried function, but a shorthand of $A_1 \times A_2 \times \dots \times A_n \rightarrow B$. Partial application of n -ary functions is implemented by implicit η -expansion. Another potential solution is to extend MET with a separate notion of pure function arrows, or more generally, function arrows that may perform some specific effects, separately from the standard function arrows which may perform any effects provided by the ambient effect context.

12.1.3 Type inference

First-class modal types. In Chapter 10, we presented the extension of Fresco to infer modal effect types in Maetel with no soundness or completeness proofs. It is interesting future work to further extend it to a general approach to type inference for first-class modal types, and prove its soundness and completeness.

First-class skeletons. Fresco only allows programmers to annotate terms with types but not skeletons. A natural extension is to support skeleton annotations and explore what extra expressiveness it brings to type inference.

Named ghosts. Fresco treats ghosts as anonymous, which makes ghosts easy to understand and reason about. However, it also leads to rejecting programs that are well-typed in other approaches as discussed in Section 8.1.9. A future extension is to support named ghosts while keeping a simple declarative specification.

Type inference for more features. The idea of mixing directions of information flow applies beyond type inference for first-class polymorphism and modal types. Extending Fresco to inference for more features, such as static overloading, GADTs, higher-kinded types, and intersection and union types, is an interesting direction for future work.

12.2 Some final thoughts

When I was applying for PhD programmes in autumn 2021, I wrote in my research proposal that reducing the programming effort involved in dealing with effect variables in effect systems was appealing to me. At that time, I did not imagine that I would spend almost two-thirds of my PhD on this topic. I am happy that I worked on this topic and ended up with something that I find exciting enough to write my thesis on.

The main benefit of modal effect types is that they save programmers from writing, reading, and understanding effect variables. Meanwhile, as of summer 2026, large language models (LLMs) have seemingly already saved programmers from writing, reading, and understanding any code. I claimed at the beginning of this thesis that, compared with traditional effect systems, modal effect types have better backward compatibility with existing programming languages: we do not have to rewrite many type signatures in existing codebases. But with LLMs, such rewriting of type signatures seems no longer a problem. LLMs can even rewrite an entire established project from one programming language to another. So is programming language design still something worth working on nowadays?

From a practical perspective, perhaps optimistically, one might argue that programmers want to check the code generated by LLMs, in which case what the code looks like and what information its type carries still matter; one might argue that type systems can provide strong guarantees that help LLMs generate more structured and sound code with fewer bugs; one might argue that designing a programming language targeted at LLMs and AI agents seems to be a promising direction that would get funded.

This is not the full story for me. I do not choose to work on programming languages merely because they are useful in practice. The most appealing part of programming language research to me is designing elegant language features and exploring their theoretical essence. For the topic of this thesis, if we only want something that works, parametric polymorphism with some syntactic sugar hiding effect variables is enough. But such a syntactic approach does not change the intrinsic complexity introduced by effect variables. Modal effect types better capture the essence of modularity in effectful programming without resorting to parametric polymorphism. Even though LLMs have become very good at programming and even at solving some open mathematical

problems, I believe that there is still a game for humans to play when design taste matters, and when we value the elegance of the theory beyond making things work. One may at least imagine me happy as I play this game.

Appendix

A Metatheory and proofs for MET

We provide metatheory and proofs for MET in Chapter 4 including some extensions from Chapter 5.

A.1 Lemmas for the mode theory

We first show that the action of μ on effect contexts gives a total monotone function.

Lemma 29 (Monotone modalities). *If $\Gamma \vdash E \leq F$, then $\Gamma \vdash \mu(E) \leq \mu(F)$.*

Proof. Straightforward case analysis on the shape of μ . For $\mu = \langle L|D \rangle$, the result follows by monotonicity of removing L and extending with D . $\square$

We prove the soundness lemma for modality transformation in Section 4.3.2.

Lemma 1 (Soundness of modality transformation). *If $\Gamma \vdash \mu \Rightarrow \nu @ F$, then $\Gamma \vdash \mu(F') \leq \nu(F')$ for all F' with $\Gamma \vdash F \leq F'$.*

Proof. By induction on the transformation derivation.

Case MT-ABS. Follows from Lemma 29 and transitivity of subeffecting.

Case MT-REL. Both modalities are $\langle L|D \rangle$, so their actions on every F' are equal.

Case MT-EXPAND. Suppose $(F - L) \equiv \ell, E$. For every $F \leq F'$, we have $(F' - L) \equiv \ell, E'$ for some E' . Inlining the definitions of modality action shows that both modalities act on F' as D, ℓ, E' .

Case MT-SHRINK. Symmetric to MT-EXPAND.

Case MT-TRANS. Follows from IHs and transitivity of subeffecting.

$\square$

We state some properties of the mode theory as the following lemmas for easier reference in proofs. Most of them directly follow from the definition.

Lemma 30 (Vertical composition). *If $\Gamma \vdash \mu \Rightarrow \nu @ F$ and $\Gamma \vdash \nu \Rightarrow \xi @ F$, then $\Gamma \vdash \mu \Rightarrow \xi @ F$.*

Proof. Directly by MT-TRANS. □

Lemma 31 (Horizontal composition). *If $\Gamma \vdash \mu \Rightarrow \mu' @ F$ and $\Gamma \vdash \nu \Rightarrow \nu' @ \mu(F)$, then $\Gamma \vdash \mu \circ \nu \Rightarrow \mu' \circ \nu' @ F$.*

Proof. By induction on the transformation derivations and case analysis on the modalities being composed. If $\mu \circ \nu$ is absolute, the result follows from Lemmas 1 and 29, transitivity of subeffecting, and MT-ABS. Otherwise all four modalities are relative. The result follows by inlining composition and reapplying MT-REL, MT-EXPAND, MT-SHRINK, and MT-TRANS. Expansion and shrinking always modify the mask and extension together, which is preserved by the definition of composition. □

Lemma 32 (Monotone modality transformation). *If $\Gamma \vdash \mu \Rightarrow \nu @ F$ and $\Gamma \vdash F \leq F'$, then $\Gamma \vdash \mu \Rightarrow \nu @ F'$.*

Proof. By induction on the transformation. The result is immediate for MT-REL. For MT-ABS, the premise and Lemma 29 give $E \leq \nu(F) \leq \nu(F')$. Our goal then follows from reapplying MT-ABS. The cases for MT-EXPAND and MT-SHRINK follow from the fact that their premises continue to hold after enlarging the ambient effect context. The case for MT-TRANS follows from IHs and reapplying MT-TRANS. □

Lemma 33 (Reflexivity of modality transformation). $\Gamma \vdash \mu \Rightarrow \mu @ F$.

Proof. By case analysis on μ . Use reflexivity of subeffecting and MT-ABS for an absolute modality, and use MT-REL for a relative modality. □

Lemma 34 (Comonadic modalities). *Suppose $\Gamma @ F$, $E = \mu(F)$, $\Gamma \vdash \mu \Rightarrow \langle \rangle @ F$, and $\Gamma \vdash \mu \Rightarrow \mu \circ \mu @ F$. Then $\Gamma, \mathbf{\mu}_\mu \vdash \mu \Rightarrow \langle \rangle @ E$ and $\Gamma, \mathbf{\mu}_\mu \vdash \mu \Rightarrow \mu \circ \mu @ E$.*

Proof. We have $\Gamma, \mathbf{\mu}_\mu @ E$. By case analysis on μ . If μ is absolute, then $\mu = [E]$ and both conclusions follow from MT-ABS. If μ is relative, transformation to $\langle \rangle$ preserves the action on F , so $E \equiv F$. The conclusions are therefore the two premises under the weakened context $\Gamma, \mathbf{\mu}_\mu$, up to effect equivalence. □

A.2 Lemmas for the calculus

We prove structural and substitution lemmas for MET as well as some other auxiliary lemmas for proving type soundness.

Lemma 35 (Canonical forms).

1. If V is a value, $\Gamma \vdash V : \mu A @ E$, and $\text{dom}(\Gamma) = \emptyset$, then either $V \rightsquigarrow V'$ for some V' , or V is of shape $\text{mod}_\mu W$.
2. If V is a value, $\Gamma \vdash V : A \rightarrow B @ E$, and $\text{dom}(\Gamma) = \emptyset$, then either $V \rightsquigarrow V'$ for some V' , or V is of shape $\lambda x^A.M$.
3. If V is a value, $\Gamma \vdash V : \forall \alpha^K.A @ E$, and $\text{dom}(\Gamma) = \emptyset$, then either $V \rightsquigarrow V'$ for some V' , or V is of shape $\Lambda \alpha^K.W$.

Proof. By simultaneous induction on the typing derivations. The interesting case is T-TAPP. By IH, its head either reduces, in which case the type application reduces by E-LIFT, or has shape $\Lambda \alpha^K.W$, in which case the type application reduces by E-TAPP. The remaining cases follow directly from the typing rules. $\square$

We have the following structural lemmas.

Lemma 36 (Structural rules). *The following structural rules are admissible.*

1. Variable weakening.

$$\frac{\Gamma, \Gamma' \vdash M : B @ E \quad \Gamma, x :_\mu A, \Gamma' @ E}{\Gamma, x :_\mu A, \Gamma' \vdash M : B @ E}$$

2. Variable swapping.

$$\frac{\Gamma, x :_\mu A, y :_\nu B, \Gamma' \vdash M : A' @ E}{\Gamma, y :_\nu B, x :_\mu A, \Gamma' \vdash M : A' @ E}$$

3. Lock weakening.

$$\frac{\Gamma, \mathbf{\mu}_\mu, \Gamma' \vdash M : A @ E \quad \Gamma @ F \quad \Gamma \vdash \mu \Rightarrow \nu @ F}{\Gamma, \mathbf{\mu}_\nu, \Gamma' \vdash M : A @ E'}$$

4. Label weakening.

$$\frac{\Omega \mid \Gamma, \Gamma' \vdash M : B @ E}{\Omega \mid \Gamma, \ell : A \twoheadrightarrow B', \Gamma' \vdash M : B @ E}$$

5. Runtime-label weakening.

$$\frac{\Omega \mid \Gamma \vdash M : B @ E}{\Omega, \ell : A \twoheadrightarrow B' \mid \Gamma \vdash M : B @ E}$$

6. Type variable weakening.

$$\frac{\Gamma, \Gamma' \vdash M : B @ E}{\Gamma, \alpha : K, \Gamma' \vdash M : B @ E}$$

7. Type variable swapping.

$$\frac{\Gamma_1, \Gamma_2, \alpha : K, \Gamma_3 \vdash M : A @ E}{\Gamma_1, \alpha : K, \Gamma_2, \Gamma_3 \vdash M : A @ E} \quad \frac{\alpha \notin \text{ftv}(\Gamma_2) \quad \Gamma_1, \alpha : K, \Gamma_2, \Gamma_3 \vdash M : A @ E}{\Gamma_1, \Gamma_2, \alpha : K, \Gamma_3 \vdash M : A @ E}$$

Proof. 1, 2, 4, 5, 6, and 7 follow from straightforward induction on the typing derivation. For 3, we also proceed by induction on the typing derivation. The most interesting case is T-VAR. Other cases mostly follow from IHs.

Case

$$\frac{\text{T-VAR} \quad \Gamma_1 \vdash (\mu', A) \Rightarrow \text{locks}(\Gamma_2) \text{ ①}}{\Gamma_1, x : \mu' A, \Gamma_2 \vdash x : A @ E}$$

Trivial when A has kind Abs. Otherwise, inversion on ① gives, for the F_1 such that $\Gamma_1 @ F_1$,

$$\Gamma_1 \vdash \mu' \Rightarrow \text{locks}(\Gamma_2) @ F_1 \text{ ②}.$$

We proceed by case analysis on where the lock weakening happens.

Case Γ . Supposing $\Gamma_1 = \Gamma, \blacksquare_{\mu}, \Gamma_0$ and after lock weakening we have $\Gamma'_1 = \Gamma, \blacksquare_{\nu}, \Gamma_0$. Let $\Gamma'_1 @ F'_1$. By context formation, we have $F_1 = \text{locks}(\Gamma_0)(\mu(F))$ and $F'_1 = \text{locks}(\Gamma_0)(\nu(F))$. By Lemma 1, the lock transformation gives $\mu(F) \leq \nu(F)$. Applying Lemma 29 to the modality $\text{locks}(\Gamma_0)$ gives $F_1 \leq F'_1$. By ② and Lemma 32, we have $\Gamma'_1 \vdash \mu' \Rightarrow \text{locks}(\Gamma_2) @ F'_1$. Reapply T-VAR.

Case Γ_2 . Suppose $\Gamma_2 = \Gamma_0, \blacksquare_{\mu}, \Gamma'$ and the context Γ_2 is weakened to $\Gamma'_2 = \Gamma_0, \blacksquare_{\nu}, \Gamma'$. By Lemma 33 and Lemma 31, we have $\Gamma_1 \vdash \text{locks}(\Gamma_2) \Rightarrow \text{locks}(\Gamma'_2) @ F_1$, where $\Gamma_1 @ F_1$. By Lemma 30 and ②, we have μ' transformed to $\text{locks}(\Gamma'_2)$. Finally reapply T-VAR.

Case

$$\frac{\text{T-Mod} \quad \Gamma, \mathbf{\mu}_\mu, \Gamma', \mathbf{\mu}_{\mu'} \vdash V : A @ \mu'(E) \text{ ①}}{\Gamma, \mathbf{\mu}_\mu, \Gamma' \vdash \text{mod}_{\mu'} V : \mu' A @ E}$$

By IH on ①, we have $\Gamma, \mathbf{\mu}_\mu, \Gamma', \mathbf{\mu}_{\mu'} \vdash V : A @ \mu'(E')$.

Then by T-Mod we have $\Gamma, \mathbf{\mu}_\mu, \Gamma' \vdash \text{mod}_{\mu'} V : \mu' A @ E'$.

Case

$$\frac{\text{T-LETMOD} \quad \Gamma, \mathbf{\mu}_\mu, \Gamma', \mathbf{\mu}_{v'} \vdash V : \mu' A @ v'(E) \text{ ①} \quad \Gamma, \mathbf{\mu}_\mu, \Gamma', x :_{v' \circ \mu'} A \vdash M : B @ E \text{ ②}}{\Gamma, \mathbf{\mu}_\mu, \Gamma' \vdash \text{let}_{v'} \text{mod}_{\mu'} x = V \text{ in } M : B @ E}$$

By IH on ①, we have $\Gamma, \mathbf{\mu}_\mu, \Gamma', \mathbf{\mu}_{v'} \vdash V : \mu' A @ v'(E')$.

By IH on ②, we have $\Gamma, \mathbf{\mu}_\mu, \Gamma', x :_{v' \circ \mu'} A \vdash M : B @ E'$.

Then by T-LETMOD, we have $\Gamma, \mathbf{\mu}_\mu, \Gamma' \vdash \text{let}_{v'} \text{mod}_{\mu'} x = V \text{ in } M : B @ E'$.

Case

$$\frac{\text{T-Do} \quad \Sigma \ni \ell : A_0 \twoheadrightarrow B_0 \quad \Gamma, \mathbf{\mu}_\mu, \Gamma' \vdash N : A_0 @ \ell, E_0 \text{ ①}}{\Gamma, \mathbf{\mu}_\mu, \Gamma' \vdash \text{do } \ell N : B_0 @ \ell, E_0}$$

Given $\Gamma @ F$, we have $\ell, E_0 = \text{locks}(\Gamma')(\mu(F))$ and $E' = \text{locks}(\Gamma')(v(F))$. By Lemma 1, $\mu(F) \leq v(F)$. Applying Lemma 29 to $\text{locks}(\Gamma')$ gives $\ell, E_0 \leq E'$, so we have $E' \equiv \ell, E'_0$ for some E'_0 . By IH on ①, we have $\Gamma, \mathbf{\mu}_\mu, \Gamma' \vdash N : A_0 @ \ell, E'_0$.

Then by T-Do, we have $\Gamma, \mathbf{\mu}_\mu, \Gamma' \vdash \text{do } \ell N : B_0 @ \ell, E'_0$.

Case T-TABS, T-TAPP, T-ABS, T-APP, T-MASK, T-LOCALEFFECT, and both T-HANDLE. Follow from IHs and reapplying the same rule. Similar to above cases.

□

As a corollary of Lemma 36.3, the following subeffecting rule is admissible.

Corollary 37 (Sub-effecting). *The following rule is admissible.*

$$\frac{\mathbf{\mu}_{[E]}, \Gamma \vdash M : A @ F \quad E \leq E' \quad \mathbf{\mu}_{[E']}, \Gamma @ F'}{\mathbf{\mu}_{[E']}, \Gamma \vdash M : A @ F'}$$

Proof. Follows from Lemma 36.3, since we have $\cdot \vdash [E] \Rightarrow [E'] @ \cdot$ by MT-ABS. □

The following lemma captures the intuition that pure values can be used in any effect context.

Lemma 38 (Pure Promotion). *The following promotion rule is admissible.*

$$\frac{\Gamma_1, \Gamma \vdash V : A @ E \quad \Gamma_1 \vdash A : \text{Abs} \quad \text{fv}(V) \subseteq \text{dom}(\Gamma_1) \quad \text{ftv}(V) \subseteq \text{dom}(\Gamma_1)}{\Gamma_1, \Gamma' \vdash V : A @ E'}$$

Proof. By induction on the typing derivation of V .

Case T-VAR. Trivial, since the variable belongs to Γ_1 and has a pure type.

Case

$$\frac{\text{T-MOD} \quad \Gamma_1, \Gamma, \blacksquare_\mu \vdash V : A @ \mu(E) \text{ ①}}{\Gamma_1, \Gamma \vdash \text{mod}_\mu V : \mu A @ E}$$

Case analysis on the shape of μ .

Subcase μ is relative. By inversion on the kinding premise, we have $\Gamma_1 \vdash A : \text{Abs}$.

By IH on ①, we have

$$\Gamma_1, \Gamma', \blacksquare_\mu \vdash V : A @ \mu(E').$$

Then by T-MOD we have

$$\Gamma_1, \Gamma' \vdash \text{mod}_\mu V : \mu A @ E'$$

Subcase μ is absolute. We have $\mu = [F_1]$ and $\text{locks}(\Gamma', \blacksquare_\mu) = [F_1] = \text{locks}(\Gamma, \blacksquare_\mu)$.

Thus, replacing the context $(\Gamma, \blacksquare_\mu)$ with $(\Gamma', \blacksquare_\mu)$ in ① does not influence usages of T-VAR for variables in Γ_1 . Thus, we have

$$\Gamma_1, \Gamma', \blacksquare_\mu \vdash V : A @ F_1.$$

Then by T-MOD we have

$$\Gamma_1, \Gamma' \vdash \text{mod}_\mu V : \mu A @ E'$$

Case

$$\frac{\text{T-TABS} \quad \Gamma_1, \Gamma, \alpha : K \vdash V : A @ E \text{ ①}}{\Gamma_1, \Gamma \vdash \Lambda \alpha^K. V : \forall \alpha^K. A @ E}$$

Case analysis on K .

Subcase $K = \text{Any}$. By inversion on the kinding premise, we have

$$\Gamma_1, \alpha : \text{Abs} \vdash A : \text{Abs} \text{ ②}.$$

We can change $\alpha : \text{Any}$ to $\alpha : \text{Abs}$ in ① by a straightforward induction. By Lemma 36.7, we therefore have

$$\Gamma_1, \alpha : \text{Abs}, \Gamma \vdash V : A @ E \text{ ③}.$$

By IH on ③ using ②, we have

$$\Gamma_1, \alpha : \text{Abs}, \Gamma' \vdash V : A @ E'.$$

Moving $\alpha : \text{Abs}$ back to the right of Γ' using Lemma 36.7 gives

$$\Gamma_1, \Gamma', \alpha : \text{Abs} \vdash V : A @ E' \text{ ④}.$$

Now by another straightforward induction on ④ and ①, observing that only contexts left of α are changed, we can change $\alpha : \text{Abs}$ back to $\alpha : \text{Any}$ in ④:

$$\Gamma_1, \Gamma', \alpha : \text{Any} \vdash V : A @ E' \text{ ⑤}.$$

Our goal follows from applying T-TAbs to ⑤.

Subcase $K = \text{Abs}$ or $K = \text{Effect}$. By inversion on the kinding premise, we have

$$\Gamma_1, \alpha : K \vdash A : \text{Abs} \text{ ②}.$$

Our goal follows from Lemma 36.7, IH on ① using ②, and T-TAbs.

Case

$$\frac{\text{T-TAPP} \quad \Gamma_1, \Gamma \vdash V : \forall \alpha^K. B @ E \text{ ①} \quad \Gamma_1, \Gamma \vdash A : K \text{ ②}}{\Gamma_1, \Gamma \vdash V A : B[A/\alpha] @ E}$$

A straightforward induction on the result type shows that, if $B[A/\alpha]$ has kind Abs , then B has kind Abs with $\alpha : \text{Abs}$ when the declared kind is not Effect , and with $\alpha : \text{Effect}$ otherwise. Thus, $\forall \alpha^K. B$ has kind Abs . By IH on ①, we may promote V . The free-type-variable side condition and ② ensure that A remains well-kinded under Γ_1, Γ' , so we may reapply T-TAPP.

Case T-Abs. Impossible since function types are impure.

□

Lemma 39 (Substitution). *The following substitution rules are admissible.*

1. *Preservation of kinds under type substitution.*

$$\frac{\Gamma \vdash A : K \quad \Gamma, \alpha : K, \Gamma' \vdash B : K'}{\Gamma, \Gamma'[A/\alpha] \vdash B[A/\alpha] : K'}$$

2. *Preservation of types under type substitution.*

$$\frac{\Gamma \vdash A : K \quad \Gamma, \alpha : K, \Gamma' \vdash M : B @ E}{\Gamma, \Gamma'[A/\alpha] \vdash M[A/\alpha] : B[A/\alpha] @ E[A/\alpha]}$$

3. *Preservation of types under value substitution.*

$$\frac{\Gamma, \mathbf{\mu}_\mu \vdash V : A @ F \quad \Gamma, x :_\mu A, \Gamma' \vdash M : B @ E}{\Gamma, \Gamma' \vdash M[V/x] : B @ E}$$

4. *Preservation of types under label substitution.*

$$\frac{\Omega \mid \Gamma, \ell : A' \twoheadrightarrow B', \Gamma' \vdash M : A @ E \quad \Omega \ni \ell' : A' \twoheadrightarrow B'}{\Omega \mid \Gamma, \Gamma'[\ell'/\ell] \vdash M[\ell'/\ell] : A[\ell'/\ell] @ E[\ell'/\ell]}$$

Proof.

1. By simultaneous induction on kinding and the well-formedness for effects, masks, and modalities.
2. By straightforward induction on the typing derivation of M .
3. By induction on the typing derivation of M . Trivial when variable x is not used. In the following induction we always assume x is used.

Case

$$\frac{\text{T-VAR} \quad \Gamma \vdash (\mu, A) \Rightarrow \text{locks}(\Gamma') \text{ ①}}{\Gamma, x :_\mu A, \Gamma' \vdash x : A @ E}$$

Case analysis on the kind of A .

Case Any. Inversion on ① gives $\Gamma @ F_0$ and $\Gamma \vdash \mu \Rightarrow \text{locks}(\Gamma') @ F_0$. By well-formedness of the value premise, $F = \mu(F_0)$. Then by $\Gamma, \mathbf{\mu}_\mu \vdash V : A @ F$ and Lemma 36.3, we have

$$\Gamma, \mathbf{\mu}_{\text{locks}(\Gamma')} \vdash V : A @ \text{locks}(\Gamma')(F_0).$$

Then, by context equivalence and Lemma 36.1, we have

$$\Gamma, \Gamma' \vdash V : A @ E.$$

Case Abs. By $\Gamma, \mathbf{\mu}_\mu \vdash V : A @ F$ and Lemma 38, we have

$$\Gamma, \Gamma' \vdash V : A @ E.$$

Case

$$\frac{\text{T-MOD} \quad \Gamma, x :_{\mu} A, \Gamma', \mathbf{A}_{\mu'} \vdash W : B @_{\mu'}(E) \text{ ①}}{\Gamma, x :_{\mu} A, \Gamma' \vdash \text{mod}_{\mu'} W : \mu' B @ E}$$

By IH on ① we have

$$\Gamma, \Gamma', \mathbf{A}_{\mu'} \vdash W[V/x] : B @_{\mu'}(E).$$

Then by T-MOD we have

$$\Gamma, \Gamma' \vdash (\text{mod}_{\mu'} W)[V/x] : \mu' B @ E$$

Case

$$\frac{\text{T-LETMOD} \quad \begin{array}{l} \Gamma, x :_{\mu} A, \Gamma', \mathbf{A}_v \vdash W : \mu' A' @ v(E) \text{ ①} \quad \Gamma, x :_{\mu} A, \Gamma', y :_{v \circ \mu'} A' \vdash M : B @ E \text{ ②} \end{array}}{\Gamma, x :_{\mu} A, \Gamma' \vdash \text{let}_v \text{mod}_{\mu'} y = W \text{ in } M : B @ E}$$

By IH on ①, we have

$$\Gamma, \Gamma', \mathbf{A}_v \vdash W[V/x] : \mu' A' @ v(E).$$

By IH on ②, we have

$$\Gamma, \Gamma', y :_{v \circ \mu'} A' \vdash M[V/x] : B @ E.$$

Then by T-LETMOD, we have

$$\Gamma, \Gamma' \vdash (\text{let}_v \text{mod}_{\mu'} y = W \text{ in } M)[V/x] : B @ E$$

Case T-TABS and T-TAPP. Follow from IHs and reapplying the same rule, after alpha-renaming in the T-TABS case.

Case

$$\frac{\text{T-MASK} \quad \Gamma, x :_{\mu} A, \Gamma', \mathbf{A}_{\langle L \rangle} \vdash M : B @ E - L \text{ ①}}{\Gamma, x :_{\mu} A, \Gamma' \vdash \text{mask}_L M : \langle L \rangle B @ E}$$

By IH on ①, we have

$$\Gamma, \Gamma', \mathbf{A}_{\langle L \rangle} \vdash M[V/x] : B @ E - L.$$

Then by T-MASK we have

$$\Gamma, \Gamma' \vdash (\text{mask}_L M)[V/x] : \langle L \rangle B @ E.$$

Case

$$\frac{\text{T-LOCALEFFECT} \quad \Omega \mid \Gamma, x :_{\mu} A, \Gamma', \ell : A_0 \rightarrow B_0 \vdash M : B @ E \text{ ①}}{\Omega \mid \Gamma, x :_{\mu} A, \Gamma' \vdash \text{local } \ell : A_0 \rightarrow B_0 \text{ in } M : B @ E}$$

Follows from IH on ① and reapplying T-LOCAL EFFECT.

Case

T-HANDLE

$$\frac{\begin{array}{c} D = \overline{\ell_i} \quad \Gamma, x :_{\mu} A, \Gamma', \mathbf{0}_{\langle D \rangle} \vdash M : A_0 @ D, E \text{ ①} \\ \Gamma, x :_{\mu} A, \Gamma', y :_{\langle D \rangle} A_0 \vdash N : B @ E \text{ ②} \\ [\Sigma \ni \ell_i : A_i \rightarrow B_i \quad \Gamma, x :_{\mu} A, \Gamma', p_i : A_i, r_i : B_i \rightarrow B \vdash N_i : B @ E \text{ ③}]_i \end{array}}{\Gamma, x :_{\mu} A, \Gamma' \vdash \text{handle } M \text{ with } \{\text{return } y \mapsto N\} \uplus \{\ell_i \ p_i \ r_i \mapsto N_i\}_i : B @ E}$$

Follows from IH on ①, ②, ③, and reapplying T-HANDLE.

Case Modality-parameterised T-HANDLE. Follows from IH on the handled computation, return clause, and operation clauses, and reapplying T-HANDLE.

Case T-ABS, T-APP, and T-DO. Follow from IH.

4. By straightforward induction on the typing derivation. $\square$

A.3 Progress

This section proves Theorem 2.

Theorem 2 (Progress). *If $\Gamma \vdash M : A @ E$ and $\text{dom}(\Gamma) = \emptyset$, then either there exists N such that $M \rightsquigarrow N$ or M is in a normal form with respect to E .*

For the extension with masks, the definition of normal forms uses $n\text{-free}(\ell, \mathcal{E})$ in place of $\ell \notin \text{bl}(\mathcal{E})$. For local labels, the theorem states that either there exist N and Ω' such that $M \mid \Omega \rightsquigarrow N \mid \Omega'$ or M is in a normal form with respect to E .

Proof. By induction on the typing derivation $\Gamma \vdash M : A @ E$. The most non-trivial case is T-HANDLE. Other cases follow from IHs and reduction rules, using Lemma 35.

Case M is a value V . Trivial. Base case.

Case T-VAR. Impossible, since the context contains no variable binding.

Case T-MOD. $\text{mod}_{\mu} V$ is a value.

Case T-LETMOD. $\text{let}_v \text{mod}_{\mu} x = V \text{ in } N$. By Lemma 35, either $V \rightsquigarrow V'$ or V has shape $\text{mod}_{\mu} W$.

Subcase $V \rightsquigarrow V'$. Then the term reduces by E-LIFT.

Subcase $V = \text{mod}_{\mu} W$. Then the term reduces by E-LETMOD.

Case T-TABs. $\Lambda\alpha^K.V$ is a value.

Case T-TAPP. $M A$. By IH on M .

Subcase $M \rightsquigarrow M'$. Then $M A \rightsquigarrow M' A$ by E-LIFT.

Subcase $M = \mathcal{E}[\text{do } \ell V]$ is in a normal form. Then $M A$ is in a normal form with evaluation context $\mathcal{E} A$.

Subcase M is a value. Then $M A$ is a value.

Case T-APP. $M N$. By IH on M .

Subcase $M \rightsquigarrow M'$. Then $M N \rightsquigarrow M' N$ by E-LIFT.

Subcase $M = \mathcal{E}[\text{do } \ell V]$ is in a normal form. Then $M N$ is in a normal form with evaluation context $\mathcal{E} N$.

Subcase M is a value. By IH on N .

Subcase $N \rightsquigarrow N'$. Then $M N \rightsquigarrow M N'$ by E-LIFT.

Subcase $N = \mathcal{E}[\text{do } \ell V]$ is in a normal form. Then $M N$ is in a normal form with evaluation context $M \mathcal{E}$.

Subcase N is a value. By Lemma 35, either $M \rightsquigarrow M'$ or M has shape $\lambda x^A.M'$.

Subcase $M \rightsquigarrow M'$. Then $M N$ reduces by E-LIFT.

Subcase $M = \lambda x^A.M'$. Then $M N$ reduces by E-APP.

Case T-DO. By IH on the argument M .

Subcase $M \rightsquigarrow M'$. Then $\text{do } \ell M \rightsquigarrow \text{do } \ell M'$ by E-LIFT.

Subcase $M = \mathcal{E}[\text{do } \ell' V]$ is in a normal form. Then $\text{do } \ell M$ is in a normal form with evaluation context $\text{do } \ell \mathcal{E}$.

Subcase M is a value. Then $\text{do } \ell M$ is in a normal form because ℓ belongs to the ambient effect context.

Case T-MASK. By IH on M .

Subcase $M \rightsquigarrow M'$. Then $\text{mask}_L M \rightsquigarrow \text{mask}_L M'$ by E-LIFT.

Subcase M is a value. Then $\text{mask}_L M$ is reducible by E-MASK.

Subcase $M = \mathcal{E}[\text{do } \ell \ V]$ is in a normal form. Then the whole term is in a normal form with evaluation context $\text{mask}_L \ \mathcal{E}$. The defining rule for $n\text{-free}(\ell, \mathcal{E})$ adds $\text{count}(\ell; L)$ to the mask count.

Case T-LOCAL EFFECT. Reducible by E-GEN.

Case

T-HANDLE

$$\begin{array}{c} H = \{\text{return } x \mapsto N\} \uplus \{\ell_i \ p_i \ r_i \mapsto N_i\}_i \\ D = \bar{\ell}_i \quad \Gamma, \mathbf{a}_{\langle D \rangle} \vdash M : A @ D, E \text{ ①} \quad \Gamma, x : \langle D \rangle \ A \vdash N : B @ E \\ [\Sigma \ni \ell_i : A_i \rightarrow B_i \quad \Gamma, p_i : A_i, r_i : B_i \rightarrow B \vdash N_i : B @ E]_i \\ \hline \Gamma \vdash \text{handle } M \text{ with } H : B @ E \end{array}$$

By IH on ①.

Subcase M is reducible. Trivial.

Subcase M is a value. By E-RET.

Subcase $M = \mathcal{E}[\text{do } \ell \ V]$ with $n\text{-free}(\ell, \mathcal{E})$. If $n = 0$ and ℓ is handled by H , by E-OP. Otherwise, since M is at effect context D, E , by a straightforward induction on the evaluation context $\mathcal{E}$ and the definition of $n\text{-free}(\ell, \mathcal{E})$ we know that $\ell \in E$. Thus the whole term is in a normal form with respect to E .

Case Modality-parameterised T-HANDLE.

T-HANDLE

$$\begin{array}{c} H = \{\text{return } x \mapsto N\} \uplus \{\ell_i \ p_i \ r_i \mapsto N_i\}_i \\ \mu(F) = E \quad \Gamma \vdash \mu \Rightarrow \langle \rangle @ F \quad \Gamma \vdash \mu \Rightarrow \mu \circ \mu @ F \\ D = \bar{\ell}_i \quad \Gamma, \mathbf{a}_\mu, \mathbf{a}_{\langle D \rangle} \vdash M : A @ D, E \text{ ①} \quad \Gamma, \mathbf{a}_\mu, x : \mu \circ \langle D \rangle \ A \vdash N : B @ E \\ [\Sigma \ni \ell_i : A_i \rightarrow B_i \quad \Gamma, \mathbf{a}_\mu, p_i : A_i, r_i :_\mu B_i \rightarrow B \vdash N_i : B @ E]_i \\ \hline \Gamma \vdash \text{handle}^\mu M \text{ with } H : B @ F \end{array}$$

By IH on ①.

Subcase M is reducible. Trivial.

Subcase M is a value. By E-RET.

Subcase $M = \mathcal{E}[\text{do } \ell \ V]$ with $n\text{-free}(\ell, \mathcal{E})$. If $n = 0$ and ℓ is handled by H , by E-OP. Otherwise, since M is at effect context D, E , by a straightforward induction on the evaluation context $\mathcal{E}$ and the definition of $n\text{-free}(\ell, \mathcal{E})$ we know that $\ell \in E$. By Lemma 1, $E \leq F$. Thus $\ell \in F$, and the whole term is in a normal form with respect to F .

□

A.4 Subject reduction

This section proves Theorem 3.

Theorem 3 (Subject Reduction). *If $\Gamma \vdash M : A @ E$ and $M \rightsquigarrow N$, then $\Gamma \vdash N : A @ E$.*

For local labels, we prove the corresponding statement for runtime configurations: if $\Omega \mid \Gamma \vdash M : A @ E$ and $M \mid \Omega \rightsquigarrow N \mid \Omega'$, then $\Omega' \mid \Gamma \vdash N : A @ E$. In displayed T-Do and ordinary T-HANDLE cases, the signature context Σ is read as Σ, Ω, Γ when local labels are enabled.

Proof. By induction on the typing derivation $\Gamma \vdash M : A @ E$.

Case T-VAR. Impossible as there is no further reduction.

Case T-MOD. Impossible as $\text{mod}_\mu V$ is a value and there is no further reduction.

Case

$$\frac{\text{T-LETMOD} \quad \Gamma, \mathbf{a}_v \vdash V : \mu A @ v(F) \text{ ①} \quad \Gamma, x :_{v \circ \mu} A \vdash M : B @ F \text{ ②}}{\Gamma \vdash \text{let}_v \text{ mod}_\mu x = V \text{ in } M : B @ F}$$

By case analysis on the reduction.

Subcase E-LIFT with $V \rightsquigarrow V'$. By definition of reduction, V' is a value. By IH on ① and reapplying T-LETMOD.

Subcase E-LETMOD. We have $V = \text{mod}_\mu W$ and

$$\text{let}_v \text{ mod}_\mu x = \text{mod}_\mu W \text{ in } M \rightsquigarrow M[W/x].$$

Inversion on ① gives

$$\Gamma, \mathbf{a}_v, \mathbf{a}_\mu \vdash W : A @ \mu(v(F)).$$

By context equivalence, we have

$$\Gamma, \mathbf{a}_{v \circ \mu} \vdash W : A @ (v \circ \mu)(F).$$

By Lemma 39.3 and ②, we have

$$\Gamma \vdash M[W/x] : B @ F.$$

Case T-ABS and T-TABS. Impossible as there is no further reduction.

Case

$$\frac{\text{T-TAPP} \quad \Gamma \vdash M : \forall \alpha^K. B @ E \text{ ①} \quad \Gamma \vdash A : K \text{ ②}}{\Gamma \vdash M A : B[A/\alpha] @ E}$$

By case analysis on the reduction.

Subcase E-LIFT with $M \rightsquigarrow N$. By IH on ① and reapplying T-APP.

Subcase E-TAPP. We have $M = \Lambda\alpha^K.V$ and

$$(\Lambda\alpha^K.V) A \rightsquigarrow V[A/\alpha].$$

Inversion on ① gives

$$\Gamma, \alpha : K \vdash V : B @ E.$$

Then by Lemma 39.2 on ②, we have

$$\Gamma \vdash V[A/\alpha] : B[A/\alpha] @ E[A/\alpha].$$

Since the conclusion of T-TABS is well-formed under Γ , α does not occur free in E . Hence $E[A/\alpha] = E$.

Case

$$\frac{\text{T-APP} \quad \Gamma \vdash M : A \rightarrow B @ E \text{ ①} \quad \Gamma \vdash N : A @ E \text{ ②}}{\Gamma \vdash M N : B @ E}$$

By case analysis on the reduction.

Case E-LIFT with $M \rightsquigarrow M'$. By IH on ① and reapplying T-APP.

Case E-LIFT with $N \rightsquigarrow N'$. By IH on ② and reapplying T-APP.

Case E-APP. We have $M = \lambda x^A.M'$, $N = V$, and

$$M N \rightsquigarrow M'[V/x].$$

Inversion on ① gives

$$\Gamma, x : A \vdash M' : B @ E.$$

By context equivalence, we may add an identity lock to the context of V .

Then by Lemma 39.3 we have

$$\Gamma \vdash M'[V/x] : B @ E.$$

Case T-Do. The only way to reduce is by E-LIFT. Follows from IH and reapplying T-Do.

Case

$$\frac{\text{T-MASK} \quad \Gamma, \mathbf{lock}_{\langle L \rangle} \vdash M : A @ F - L \text{ ①}}{\Gamma \vdash \text{mask}_L M : \langle L \rangle A @ F}$$

By case analysis on the reduction.

Case E-LIFT with $M \rightsquigarrow M'$. By IH on ① and reapplying T-MASK.

Case E-MASK. We have $M = V$ and

$$\text{mask}_L V \rightsquigarrow \text{mod}_{\langle L \rangle} V.$$

Premise ① is exactly the premise required by T-MOD for modality $\langle L \rangle$ at effect context F . Thus we have

$$\Gamma \vdash \text{mod}_{\langle L \rangle} V : \langle L \rangle A @ F.$$

Case

$$\frac{\text{T-LOCAL EFFECT} \quad \Omega \mid \Gamma, \ell : A \rightarrow B \vdash M : A' @ E \text{ ①}}{\Omega \mid \Gamma \vdash \text{local } \ell : A \rightarrow B \text{ in } M : A' @ E}$$

The only way to reduce is by E-GEN. By runtime-label weakening on ① and Lemma 39.4, we have

$$\Omega, \ell' : A \rightarrow B \mid \Gamma \vdash M[\ell'/\ell] : A' @ E.$$

The label ℓ does not appear in A' or E by well-formedness.

Case

$$\frac{\text{T-HANDLE} \quad \begin{array}{l} H = \{\text{return } x \mapsto N\} \uplus \{\ell_i \ p_i \ r_i \mapsto N_i\}_i \\ D = \bar{\ell}_i \quad \Gamma, \blacksquare_{\langle D \rangle} \vdash M : A @ D, F \text{ ①} \quad \Gamma, x : \langle D \rangle A \vdash N : B @ F \text{ ②} \\ [\Sigma \ni \ell_i : A_i \rightarrow B_i \quad \Gamma, p_i : A_i, r_i : B_i \rightarrow B \vdash N_i : B @ F \text{ ③}]_i \end{array}}{\Gamma \vdash \text{handle } M \text{ with } H : B @ F}$$

By case analysis on the reduction.

Case E-LIFT with $M \rightsquigarrow M'$. By IH on ① and reapplying T-HANDLE.

Case E-RET. We have $M = V$ and

$$\text{handle } V \text{ with } H \rightsquigarrow N[V/x].$$

Then by ② and Lemma 39.3 we have

$$\Gamma \vdash N[V/x] : B @ F.$$

Case E-OP. We have $M = \mathcal{E}[\text{do } \ell_j \ V], \ell_j \notin \text{bl}(\mathcal{E})$ in the core calculus, or $0\text{-free}(\ell_j, \mathcal{E})$ with masks, and $(\ell_j \ p_j \ r_j \mapsto N_j) \in H$. Thus,

$$\text{handle } M \text{ with } H \rightsquigarrow N_j[V/p_j, (\lambda y. \text{handle } \mathcal{E}[y] \text{ with } H)/r_j].$$

We know that V has type A_j by the unique operation signature $\ell_j : A_j \rightarrow B_j$ in the available label contexts. By ①, a straightforward induction on $\mathcal{E}$, and inversion on the typing judgement of $\text{do } \ell_j V$, we have

$$\Gamma, \Gamma' \vdash V : A_j @ E'$$

for some Γ' containing only locks and some E' . Since the operation signature is well-kinded, $A_j : \text{Abs}$ and $B_j : \text{Abs}$. By $A_j : \text{Abs}$ and Lemma 38, we have

$$\Gamma \vdash V : A_j @ F \text{ ④}.$$

Observe that $B_j : \text{Abs}$ allows y to be accessed in any context. By ① and a straightforward induction on $\mathcal{E}$ we have

$$\Gamma, y : B_j, \blacksquare_{\langle D \rangle} \vdash \mathcal{E}[y] : A @ D, F.$$

Then by variable weakening on ② and ③ and T-HANDLE, we have

$$\Gamma, y : B_j \vdash \text{handle } \mathcal{E}[y] \text{ with } H : B @ F.$$

Then by T-ABS we have

$$\Gamma \vdash \lambda y. \text{handle } \mathcal{E}[y] \text{ with } H : B_j \rightarrow B @ F \text{ ⑤}.$$

Finally, by ③, ④, ⑤, context equivalence, and Lemma 39.3 we have

$$\Gamma \vdash N_j[V/p_j, (\lambda y. \text{handle } \mathcal{E}[y] \text{ with } H)/r_j] : B @ F.$$

Case Modality-parameterised T-HANDLE.

T-HANDLE

$$\begin{array}{c} H = \{\text{return } x \mapsto N\} \uplus \{\ell_i p_i r_i \mapsto N_i\}_i \\ \mu(F) = E \quad \Gamma \vdash \mu \Rightarrow \langle \rangle @ F \quad \Gamma \vdash \mu \Rightarrow \mu \circ \mu @ F \\ D = \bar{\ell}_i \quad \Gamma, \blacksquare_{\mu}, \blacksquare_{\langle D \rangle} \vdash M : A @ D, E \text{ ①} \quad \Gamma, \blacksquare_{\mu}, x : \mu \circ \langle D \rangle A \vdash N : B @ E \text{ ②} \\ [\Sigma, \Omega, \Gamma \ni \ell_i : A_i \rightarrow B_i \quad \Gamma, \blacksquare_{\mu}, p_i : A_i, r_i : \mu B_i \rightarrow B \vdash N_i : B @ E \text{ ③}]_i \\ \hline \Gamma \vdash \text{handle}^{\mu} M \text{ with } H : B @ F \end{array}$$

By case analysis on the reduction.

Case E-LIFT with $M \rightsquigarrow M'$. By IH on ① and reapplying T-HANDLE.

Case E-RET. We have $M = V$ and

$$\text{handle}^{\mu} V \text{ with } H \rightsquigarrow N[V/x].$$

By ①, $\mu \Rightarrow \mu \circ \mu @ F$, Lemma 36.3, and context equivalence, we have

$$\Gamma, \blacksquare_{\mu}, \blacksquare_{\mu \circ \langle D \rangle} \vdash V : A @ E_1,$$

where $E_1 = (\mu \circ \langle D \rangle)(E) = D, \mu(E)$. By ② and Lemma 39.3, we have

$$\Gamma, \blacksquare_{\mu} \vdash N[V/x] : B @ E.$$

Then $\mu \Rightarrow \langle \rangle @F$, Lemma 36.3, and context equivalence give

$$\Gamma \vdash N[V/x] : B @F.$$

Case E-Op. We have $M = \mathcal{E}[\text{do } \ell_j V, 0\text{-free}(\ell_j, \mathcal{E})]$, and $(\ell_j p_j r_j \mapsto N_j) \in H$. Thus we have

$$\text{handle}^\mu M \text{ with } H \rightsquigarrow N_j[V/p_j, (\lambda y. \text{handle}^\mu \mathcal{E}[y] \text{ with } H)/r_j].$$

By ③, $\mu \Rightarrow \langle \rangle @F$, Lemma 36.3, and context equivalence, we have

$$\Gamma, p_j : A_j, r_j :_\mu B_j \rightarrow B \vdash N_j : B @F \text{ ④}.$$

The label ℓ_j belongs to the global, runtime, or local label context. Its unique signature is $A_j \rightarrow B_j$. By ①, a straightforward induction on $\mathcal{E}$, and inversion on the typing judgement of $\text{do } \ell_j V$, we have

$$\Gamma, \Gamma' \vdash V : A_j @E_1$$

for some Γ' containing only locks and some E_1 . Since operation signatures are well-kinded, $A_j : \text{Abs}$ and $B_j : \text{Abs}$. By Lemma 38, we have

$$\Gamma \vdash V : A_j @F \text{ ⑤}.$$

By ①, $\mu \Rightarrow \mu \circ \mu @F$, Lemma 36.3, and context equivalence, writing $\mu(E) = E_2$, we have

$$\Gamma, \mathbf{\mu}_\mu, \mathbf{\mu}_\mu, \mathbf{\mu}_{(D)} \vdash M : A @D, E_2.$$

Observe that $B_j : \text{Abs}$ allows y to be accessed in any context. A straightforward induction on $\mathcal{E}$ gives

$$\Gamma, \mathbf{\mu}_\mu, y : B_j, \mathbf{\mu}_\mu, \mathbf{\mu}_{(D)} \vdash \mathcal{E}[y] : A @D, E_2.$$

By Lemma 34, the two transformation premises remain valid under $\Gamma, \mathbf{\mu}_\mu$ at $E = \mu(F)$. We weaken these transformation judgements by $y : B_j$. Using the same lock duplication and variable weakening on ② and ③, we can reapply T-HANDLE to get

$$\Gamma, \mathbf{\mu}_\mu, y : B_j \vdash \text{handle}^\mu \mathcal{E}[y] \text{ with } H : B @E.$$

Then by T-Abs we have

$$\Gamma, \mathbf{\mu}_\mu \vdash \lambda y. \text{handle}^\mu \mathcal{E}[y] \text{ with } H : B_j \rightarrow B @E \text{ ⑥}.$$

Finally, by ④, ⑤, ⑥, context equivalence, and two uses of Lemma 39.3, we have

$$\Gamma \vdash N_j[V/p_j, (\lambda y. \text{handle}^\mu \mathcal{E}[y] \text{ with } H)/r_j] : B @F.$$

□

A.5 Remarks on the other extensions

The proofs in previous sections do not cover all extensions in Chapter 5. The proofs for the remaining extensions are mostly routine. We discuss the relation between the above proofs and the remaining extensions as follows.

1. Presence types only change the mode theory. We need to update the proofs for the lemmas on the mode theory.
2. Data types, shallow handlers, and the $\text{let}_\nu \text{ mod}_\mu \Lambda$ construct do not change the mode theory. We need to add routine cases to induction proofs for the lemmas and theorems on the calculus.
3. Parameterised effects change both the mode theory and typing rules. We need to update the proofs for both.

B Metatheory and proofs for $\mathbf{MET}(\mathcal{X})$

We provide metatheory and proofs for $\mathbf{MET}(\mathcal{X})$ introduced in Chapter 6. The proofs are based on the proofs for $\mathbf{MET}$ but are parameterised over the effect structure $\mathcal{X}$. The progress proof requires the effect structure to satisfy the validity conditions in Definition 7.

B.1 Lemmas for the mode theory of $\mathbf{MET}(\mathcal{X})$

Our proofs for type soundness rely on some properties of the mode theory of $\mathbf{MET}(\mathcal{X})$. The following lemmas correspond to the lemmas in Appendix A.1.

We first show that the action of μ on effect contexts gives a total monotone function.

Lemma 40 (Monotone modalities). *If $\Gamma \vdash E \leq F$, then $\Gamma \vdash \mu(E) \leq \mu(F)$.*

Proof. By straightforward case analysis on the shape of μ . Trivial for an absolute modality. For a relative modality $\langle D \rangle$, we have $\Gamma \vdash D, E \leq D, F$ by the definition of subeffecting and the premise. $\square$

We prove the soundness lemma for modality transformation.

Lemma 41 (Soundness of modality transformation). *If $\Gamma \vdash \mu \Rightarrow \nu @ F$, then $\Gamma \vdash \mu(F') \leq \nu(F')$ for every $\Gamma \vdash F \leq F'$.*

Proof. By case analysis on the transformation rule.

Case MT-ABS. Follows from Lemma 40 and transitivity of subeffecting.

Case MT-REL. Directly from the premise of the rule. $\square$

We prove some properties on the categorical structures of the mode theory.

Lemma 42 (Vertical composition). *If $\Gamma \vdash \mu \Rightarrow \nu @ F$ and $\Gamma \vdash \nu \Rightarrow \xi @ F$, then $\Gamma \vdash \mu \Rightarrow \xi @ F$.*

Proof. Suppose first that μ is absolute. By Lemma 41 and transitivity of subeffecting, we have $\mu(F) \leq \xi(F)$, so the goal follows from MT-Abs. Otherwise, all three modalities are relative. For every $F \leq F'$, two uses of Lemma 41 and transitivity give $\mu(F') \leq \xi(F')$. The goal follows from MT-REL. $\square$

Lemma 43 (Horizontal composition). *If $\Gamma \vdash \mu \Rightarrow \mu' @ F$ and $\Gamma \vdash \nu \Rightarrow \nu' @ \mu(F)$, then $\Gamma \vdash \mu \circ \nu \Rightarrow \mu' \circ \nu' @ F$.*

Proof. If $\mu \circ \nu$ is absolute, soundness of the two premises, monotonicity of ν' , and transitivity give

$$\nu(\mu(F)) \leq \nu'(\mu(F)) \leq \nu'(\mu'(F)).$$

The goal follows from MT-Abs. Otherwise, all four modalities are relative. Consider any $F \leq F'$. Monotonicity of μ gives $\mu(F) \leq \mu(F')$. Soundness of the two premises and monotonicity of ν' give

$$\nu(\mu(F')) \leq \nu'(\mu(F')) \leq \nu'(\mu'(F')).$$

The goal follows from MT-REL. $\square$

Lemma 44 (Monotone modality transformation). *If $\Gamma \vdash \mu \Rightarrow \nu @ F$ and $\Gamma \vdash F \leq F'$, then $\Gamma \vdash \mu \Rightarrow \nu @ F'$.*

Proof. By case analysis on the transformation rule. For MT-Abs, the premise and Lemma 40 give $E \leq \nu(F) \leq \nu(F')$. The result follows by reapplying MT-Abs. For MT-REL, every effect context above F' is also above F , so the premise of MT-REL continues to hold. $\square$

Lemma 45 (Reflexivity of modality transformation). $\Gamma \vdash \mu \Rightarrow \mu @ F$.

Proof. Trivial case analysis on μ . $\square$

Lemma 46 (Comonadic modalities). *Suppose $\Gamma @ F$, $E = \mu(F)$, $\Gamma \vdash \mu \Rightarrow \langle \rangle @ F$, and $\Gamma \vdash \mu \Rightarrow \mu \circ \mu @ F$. Then $\Gamma, \blacksquare_\mu \vdash \mu \Rightarrow \langle \rangle @ E$ and $\Gamma, \blacksquare_\mu \vdash \mu \Rightarrow \mu \circ \mu @ E$.*

Proof. We have $\Gamma, \blacksquare_\mu @ E$. By case analysis on μ . If μ is absolute, then $\mu = [E]$ and both conclusions follow from MT-Abs. Suppose $\mu = \langle D \rangle$, so $E = D, F$. Consider any $E \leq F_1$. By the definition of subeffecting, there is an E_1 such that $D, F, E_1 \equiv F_1$. We also have $F \leq F, E_1$. Soundness of the first modality transformation at F, E_1 gives

$$D, F, E_1 \leq F, E_1.$$

Prefixing both sides with D and using $D, F, E_1 \equiv F_1$ gives $D, F_1 \leq F_1$. Thus, MT-REL and context weakening give $\Gamma, \blacksquare_\mu \vdash \mu \Rightarrow \langle \rangle @ E$. Soundness of the second premise at F, E_1 gives

$$D, F, E_1 \leq D, D, F, E_1.$$

Using $D, F, E_1 \equiv F_1$, this gives $F_1 \leq D, F_1$. Monotonicity of $\langle D \rangle$ then gives $D, F_1 \leq D, D, F_1$. Thus, MT-REL and context weakening give $\Gamma, \mathbf{a}_\mu \vdash \mu \Rightarrow \mu \circ \mu @ E$. $\square$

B.2 Lemmas for the calculus

We state the counterparts of the lemmas in Appendix A.2 for MET($\mathcal{X}$).

Lemma 47 (Canonical forms). *Suppose Γ contains no term-variable bindings.*

1. If $\Gamma \vdash U : \mu A @ E$, then U is of shape $\text{mod}_\mu U'$.
2. If $\Gamma \vdash U : A \rightarrow B @ E$, then U is of shape $\lambda x^A.M$.
3. If $\Gamma \vdash U : \forall \alpha^K.A @ E$, then U is of shape $\Lambda \alpha^K.U'$.
4. If $\Gamma \vdash U : 1 @ E$, then $U = ()$.

Proof. By inversion on the typing rules and the syntax of U . $\square$

We have the following structural lemmas.

Lemma 48 (Structural rules). *The following structural rules are admissible.*

1. Variable weakening.

$$\frac{\Gamma, \Gamma' \vdash M : B @ E \quad \Gamma, x :_\mu A, \Gamma' @ E}{\Gamma, x :_\mu A, \Gamma' \vdash M : B @ E}$$

2. Variable swapping.

$$\frac{\Gamma, x :_\mu A, y :_\nu B, \Gamma' \vdash M : A' @ E}{\Gamma, y :_\nu B, x :_\mu A, \Gamma' \vdash M : A' @ E}$$

3. Lock weakening.

$$\frac{\Gamma, \mathbf{a}_\mu, \Gamma' \vdash M : A @ E \quad \Gamma @ F \quad \Gamma \vdash \mu \Rightarrow \nu @ F}{\Gamma, \mathbf{a}_\nu, \Gamma' \vdash M : A @ E'}$$

4. Label weakening.

$$\frac{\Omega \mid \Gamma, \Gamma' \vdash M : B @ E}{\Omega \mid \Gamma, \ell : A \twoheadrightarrow B', \Gamma' \vdash M : B @ E}$$

5. Runtime-label weakening.

$$\frac{\Omega \mid \Gamma \vdash M : B @ E}{\Omega, \ell : A \twoheadrightarrow B' \mid \Gamma \vdash M : B @ E}$$

6. Type variable weakening.

$$\frac{\Gamma, \Gamma' \vdash M : B @ E}{\Gamma, \alpha : K, \Gamma' \vdash M : B @ E}$$

7. Type variable swapping.

$$\frac{\Gamma_1, \Gamma_2, \alpha : K, \Gamma_3 \vdash M : A @ E}{\Gamma_1, \alpha : K, \Gamma_2, \Gamma_3 \vdash M : A @ E} \quad \frac{\alpha \notin \text{ftv}(\Gamma_2) \quad \Gamma_1, \alpha : K, \Gamma_2, \Gamma_3 \vdash M : A @ E}{\Gamma_1, \Gamma_2, \alpha : K, \Gamma_3 \vdash M : A @ E}$$

Proof. Follow the proof of Lemma 36, using the mode-theory lemmas in Appendix B.1. $\square$

As a corollary of Lemma 48.3, the following subeffecting rule is admissible.

Corollary 49 (Sub-effecting). *The following rule is admissible.*

$$\frac{\mathbf{A}_{[E]}, \Gamma \vdash M : A @ F \quad E \leq E' \quad \mathbf{A}_{[E']}, \Gamma @ F'}{\mathbf{A}_{[E']}, \Gamma \vdash M : A @ F'}$$

Proof. Follow the proof of Corollary 37, using Lemma 48.3 and MT-Abs. $\square$

Lemma 50 (Pure Promotion). *The following promotion rule is admissible.*

$$\frac{\Gamma_1, \Gamma \vdash U : A @ E \quad \Gamma_1 \vdash A : \text{Abs} \quad \Gamma_1, \Gamma' @ E' \quad \text{fv}(U) \subseteq \text{dom}(\Gamma_1) \quad \text{ftv}(U) \subseteq \text{dom}(\Gamma_1)}{\Gamma_1, \Gamma' \vdash U : A @ E'}$$

Proof. By induction on the typing derivation of U , following Lemma 38. The type-abstraction case uses Lemma 48.7 and applies the IH to its normal-form body. Modality elimination and type application are excluded by the syntax of U . $\square$

Lemma 51 (Generalised Pure Promotion). *Given $\Gamma_1, \Gamma \vdash M : A @ E$, if, for every $x \in \text{fv}(M)$, we have $\Gamma_1 \ni x :_{\mu} B$ such that either $\Gamma_1 \vdash B : \text{Abs}$ or μ is an absolute modality, then $\Gamma_1, \Gamma' \vdash M : A @ E'$ for $E \leq E'$ and $\Gamma_1, \Gamma' @ E'$.*

Proof. By straightforward induction on typing judgements in $\text{MET}(\mathcal{X})$. The most non-trivial case is to show the accessibility of every variable $x \in \text{fv}(M)$. Variables with types of kind **Abs** are always accessible. For variables annotated with an absolute modality, the modality transformation relation MT-Abs still holds because $E \leq E'$. $\square$

Lemma 52 (Substitution). *The following substitution rules are admissible.*

1. *Preservation of kinds under type substitution.*

$$\frac{\Gamma \vdash A : K \quad \Gamma, \alpha : K, \Gamma' \vdash B : K'}{\Gamma, \Gamma'[A/\alpha] \vdash B[A/\alpha] : K'}$$

2. *Preservation of types under type substitution.*

$$\frac{\Gamma \vdash A : K \quad \Gamma, \alpha : K, \Gamma' \vdash M : B @ E}{\Gamma, \Gamma'[A/\alpha] \vdash M[A/\alpha] : B[A/\alpha] @ E[A/\alpha]}$$

3. *Preservation of types under value substitution.*

$$\frac{\Gamma, \mathbf{\mu}_\mu \vdash U : A @ F \quad \Gamma, x :_\mu A, \Gamma' \vdash M : B @ E}{\Gamma, \Gamma' \vdash M[U/x] : B @ E}$$

4. *Preservation of types under label substitution.*

$$\frac{\Omega \mid \Gamma, \ell : A' \rightarrow B', \Gamma' \vdash M : A @ E \quad \Omega \ni \ell' : A' \rightarrow B'}{\Omega \mid \Gamma, \Gamma'[\ell'/\ell] \vdash M[\ell'/\ell] : A[\ell'/\ell] @ E[\ell'/\ell]}$$

Proof. Follow the proof of Lemma 39, using Lemmas 48 and 50. In part 3, substitution of U preserves the syntax of values and value normal forms, by structural induction. $\square$

B.3 Progress

This section proves Theorem 8.

Theorem 8 (Progress). *In $\text{MET}(\mathcal{X})$ where $\mathcal{X}$ satisfies the validity conditions, if $\Omega \mid \Gamma \vdash M : A @ E$ and $\text{dom}(\Gamma) = \emptyset$, then either there exist N and Ω' such that $M \mid \Omega \rightsquigarrow N \mid \Omega'$, or M is in a normal form with respect to E .*

Proof. By induction on the typing derivation. Cases whose terms are value normal forms are immediate. For T-TABS, the IH on its value body gives either a reduction, which lifts by E-LIFT, or a value normal form, which makes the abstraction a value normal form. The remaining cases follow the proof for MET in Appendix A.3, using the IHs, the reduction rules, and canonical forms. We expand the handler case, which is the only case that uses a validity condition of the general effect structure.

Case

T-HANDLE

$$\begin{array}{c}
 E = \mu(F) \quad \Gamma \vdash \mu \Rightarrow \langle \rangle @ F \\
 \Gamma \vdash \mu \Rightarrow \mu \circ \mu @ F \quad \Gamma, \blacksquare_\mu, \blacksquare_{\langle \ell \rangle} \vdash M : A @ \ell, E \textcircled{1} \quad \Sigma, \Gamma \ni \ell : A' \twoheadrightarrow B' \\
 \Gamma, \blacksquare_\mu, x : \mu \circ \langle \ell \rangle \quad A \vdash N : B @ E \quad \Gamma, \blacksquare_\mu, p : A', r :_\mu B' \rightarrow B \vdash N' : B @ E \\
 \hline
 \Gamma \vdash \text{handle}^\mu M \text{ with } \{\text{return } x \mapsto N, \ell \ p \ r \mapsto N'\} : B @ F
 \end{array}$$

By IH on $\textcircled{1}$. The reducible and value normal form subcases follow from E-LIFT and E-RET, respectively. Suppose $M = \mathcal{E}[\text{do } \ell' U]$ is in normal form, so $\ell' \leq \ell, E$ and $\ell' \notin \text{bl}(\mathcal{E})$. If $\ell = \ell'$, the term reduces by E-OP. Otherwise, the second validity condition in Definition 7 gives $\ell' \leq E$. By Lemma 41 on $\mu \Rightarrow \langle \rangle @ F$, we have $E \leq F$. Therefore $\ell' \leq F$, and the whole term is in normal form with respect to F .

□

B.4 Subject reduction

This section proves Theorem 9.

Theorem 9 (Subject Reduction). *In $\text{MET}(\mathcal{X})$, if $\Omega \mid \Gamma \vdash M : A @ E$ and $M \mid \Omega \rightsquigarrow N \mid \Omega'$, then $\Omega' \mid \Gamma \vdash N : A @ E$.*

Proof. By induction on the typing derivation. The cases shared with MET follow the proof in Appendix A.4, using the mode-theory lemmas above. In particular, the handler case uses Lemma 46 in place of its MET counterpart. For reductions under type abstraction, apply the IH to the body and reapply T-TABS, since reduction preserves value syntax. No subject-reduction case uses either validity condition. □

C Proofs of encodings

We prove the type preservation and semantics preservation theorems in Chapter 7. When referring to the name of a rule, we sometimes also mention the calculus name to disambiguate. For instance, $\text{T-VAR-SYSTEM } F^\epsilon$ refers to the rule T-VAR of System F^ϵ .

C.1 Proofs of encoding System F^ϵ in $\text{MET}(\mathcal{R}_{\text{scp}})$

The source calculus and its encoding are defined in Sections 7.2.1 and 7.2.2. The proofs rely on the following lemmas.

Lemma 53 (Pure Values). *Given a typing judgement $\Gamma \vdash V : A$ in System F^ϵ , if $\llbracket \Gamma \rrbracket \vdash \llbracket V \rrbracket : \llbracket A \rrbracket @ \cdot$ then $\llbracket \Gamma \rrbracket_E \vdash \llbracket V \rrbracket : \llbracket A \rrbracket @ \llbracket E \rrbracket$ for any E .*

Proof. By lock weakening and MT-ABS. □

Lemma 54 (Translation of Values). *If V is a value in System F^ϵ , then $\llbracket V \rrbracket$ is a value normal form in $\text{MET}(\mathcal{R}_{\text{scp}})$.*

Proof. By induction on V . The unit and variable cases are immediate. Type abstractions translate to type abstractions whose bodies are value normal forms by IH. Function and handler values translate to modality introductions whose inner terms are lambda abstractions. □

Lemma 55 (Translation of Evaluation Contexts). *For the translation $\llbracket - \rrbracket$ from System F^ϵ to $\text{MET}(\mathcal{R}_{\text{scp}})$, we have $\llbracket \mathcal{E}[M] \rrbracket = \llbracket \mathcal{E} \rrbracket[\llbracket M \rrbracket]$ for any evaluation context $\mathcal{E}$ and term M .*

Proof. By straightforward case analysis on evaluation contexts of System F^ϵ . □

Theorem 11 (Type Preservation). *If $\Gamma \vdash M : A \mid E$ in System F^ϵ , then $\llbracket \Gamma \rrbracket_E \vdash \llbracket M \rrbracket : \llbracket A \rrbracket @ \llbracket E \rrbracket$ in $\text{MET}(\mathcal{R}_{\text{scp}})$. Similarly for typing judgements of values.*

Proof. By induction on typing judgements $\Gamma \vdash M : A \mid E$ in System F^ϵ . Most cases follow from using IH trivially. We elaborate interesting cases. When referring to the name of a rule, we sometimes also mention the calculus name to disambiguate. For instance, $\text{T-VAR-SYSTEM } F^\epsilon$ refers to the rule T-VAR of System F^ϵ .

Case $\boxed{()}$ By T-UNIT-SYSTEM F^c and T-UNIT-MET($\mathcal{R}_{\text{SCP}}$).

Case $\boxed{x}$ All translated types have kind Abs in MET($\mathcal{R}_{\text{SCP}}$) and thus can always be accessed by T-VAR-MET($\mathcal{R}_{\text{SCP}}$).

Case $\boxed{\Lambda \alpha^K.V}$ By IH, type-variable exchange, T-TABS-SYSTEM F^c , and T-TABS-MET($\mathcal{R}_{\text{SCP}}$).

Case $\boxed{V A}$ By IH, T-TAPP-SYSTEM F^c and T-TAPP-MET($\mathcal{R}_{\text{SCP}}$).

Case $\boxed{\lambda^E x^A.M}$

$$\frac{\text{T-Abs} \quad \Gamma, x : A \vdash M : B \mid E \text{ ①}}{\Gamma \vdash \lambda^E x^A.M : A \rightarrow^E B}$$

By IH on ①, weakening, and exchange, we have

$$\llbracket \Gamma \rrbracket, \blacksquare_{\llbracket E \rrbracket}, x : \llbracket A \rrbracket \vdash \llbracket M \rrbracket : \llbracket B \rrbracket @ \llbracket E \rrbracket$$

By T-ABS-MET($\mathcal{R}_{\text{SCP}}$) and T-MOD-MET($\mathcal{R}_{\text{SCP}}$), we have

$$\llbracket \Gamma \rrbracket. \vdash \text{mod}_{\llbracket E \rrbracket} (\lambda x^{\llbracket A \rrbracket}. \llbracket M \rrbracket) : \llbracket E \rrbracket (\llbracket A \rrbracket \rightarrow \llbracket B \rrbracket) @ \cdot$$

Case $\boxed{\text{return } V}$ By Lemma 53.

Case $\boxed{\text{let } x = M \text{ in } N}$ By IH, syntactic sugar, and T-APP-MET($\mathcal{R}_{\text{SCP}}$).

Case $\boxed{V W}$

$$\frac{\text{T-App} \quad \Gamma \vdash V : A \rightarrow^E B \text{ ①} \quad \Gamma \vdash W : A \text{ ②}}{\Gamma \vdash V W : B \mid E}$$

By IH on ① and Lemma 53, we have

$$\llbracket \Gamma \rrbracket_E \vdash \llbracket V \rrbracket : \llbracket E \rrbracket (\llbracket A \rrbracket \rightarrow \llbracket B \rrbracket) @ \llbracket E \rrbracket$$

By IH on ② and Lemma 53, we have

$$\llbracket \Gamma \rrbracket_E \vdash \llbracket W \rrbracket : \llbracket A \rrbracket @ \llbracket E \rrbracket$$

By T-LETMOD-MET($\mathcal{R}_{\text{SCP}}$) and T-APP-MET($\mathcal{R}_{\text{SCP}}$), we have

$$\llbracket \Gamma \rrbracket_E \vdash \text{let mod}_{\llbracket E \rrbracket} x = \llbracket V \rrbracket \text{ in } x \llbracket W \rrbracket : \llbracket B \rrbracket @ \llbracket E \rrbracket$$

Case $\boxed{\text{do } \ell V}$ By IH, Lemma 53, T-DO-SYSTEM F^c and T-DO-MET($\mathcal{R}_{\text{SCP}}$).

Case $\boxed{\text{handler } \{\ell \ p \ r \mapsto N\}}$

$$\frac{\text{T-HANDLER} \quad \Sigma \ni \ell : A' \rightarrow B' \quad \Gamma, p : A', r : B' \rightarrow^E A \vdash N : A \mid E \text{ ①}}{\Gamma \vdash \text{handler } \{\ell \ p \ r \mapsto N\} : (1 \rightarrow^{\ell, E} A) \rightarrow^E A}$$

The translated operation clause is

$$\ell \ p \ r \mapsto \text{let } r' = \text{mod}_{[[E]]} r \text{ in } [[N]][r'/r]$$

By the IH on ①, alpha-renaming its translated continuation binding to r' , Lemma 51, weakening, and exchange, we have

$$\begin{aligned} & [[\Gamma]], \blacksquare_{[[E]]}, f : [[\ell, E]](1 \rightarrow [[A]]), \blacksquare_{[[E]]}, p : [[A']], \text{ ②} \\ & r : [[E]] [[B']] \rightarrow [[A]], r' : [[E]] ([[B']] \rightarrow [[A]]) \vdash [[N]][r'/r] : [[A]] @ [[E]] \end{aligned}$$

The type of r' is the translation of $B' \rightarrow^E A$. By T-VAR-MET($\mathcal{R}_{\text{SCP}}$), T-MOD-MET($\mathcal{R}_{\text{SCP}}$), and ②, we obtain the operation-clause premise

$$\begin{aligned} & [[\Gamma]], \blacksquare_{[[E]]}, f : [[\ell, E]](1 \rightarrow [[A]]), \blacksquare_{[[E]]}, p : [[A']], r : [[E]] [[B']] \rightarrow [[A]] \text{ ③} \\ & \vdash \text{let } r' = \text{mod}_{[[E]]} r \text{ in } [[N]][r'/r] : [[A]] @ [[E]] \end{aligned}$$

Since $[[A]] : \text{Abs}$, T-VAR-MET($\mathcal{R}_{\text{SCP}}$) gives the return-clause premise

$$[[\Gamma]], \blacksquare_{[[E]]}, f : [[\ell, E]](1 \rightarrow [[A]]), \blacksquare_{[[E]]}, x : [[E]] \circ \langle \ell \rangle [[A]] \vdash x : [[A]] @ [[E]] \text{ ④}$$

By T-VAR-MET($\mathcal{R}_{\text{SCP}}$), T-LETMOD-MET($\mathcal{R}_{\text{SCP}}$), and T-APP-MET($\mathcal{R}_{\text{SCP}}$), we have the handled-computation premise

$$\begin{aligned} & [[\Gamma]], \blacksquare_{[[E]]}, f : [[\ell, E]](1 \rightarrow [[A]]), \blacksquare_{[[E]]}, \blacksquare_{\langle \ell \rangle} \text{ ⑤} \\ & \vdash \text{let } \text{mod}_{[[\ell, E]]} f' = f \text{ in } f' () : [[A]] @ [[\ell, E]] \end{aligned}$$

By ③, ④, ⑤, and T-HANDLE-MET($\mathcal{R}_{\text{SCP}}$), we have

$$\begin{aligned} & [[\Gamma]], \blacksquare_{[[E]]}, f : [[\ell, E]](1 \rightarrow [[A]]) \vdash \\ & \text{handle}^{[[E]]} (\text{let } \text{mod}_{[[\ell, E]]} f' = f \text{ in } f' ()) \\ & \text{with } \{\text{return } x \mapsto x, \ell \ p \ r \mapsto \text{let } r' = \text{mod}_{[[E]]} r \text{ in } [[N]][r'/r]\} : [[A]] @ [[E]] \end{aligned}$$

Finally, T-ABS-MET($\mathcal{R}_{\text{SCP}}$) and T-MOD-MET($\mathcal{R}_{\text{SCP}}$) give:

$$[[\Gamma]]. \vdash [[\text{handler } \{\ell \ p \ r \mapsto N\}]] : ((1 \rightarrow^{\ell, E} A) \rightarrow^E A) @ \cdot$$

□

Theorem 12 (Semantics Preservation). *If M is well-typed and $M \rightsquigarrow N$ in System F^e , then $[[M]] \rightsquigarrow^* [[N]]$ in $\text{MET}(\mathcal{R}_{\text{SCP}})$ where $\rightsquigarrow^*$ denotes the reflexive-transitive closure of $\rightsquigarrow$.*

Proof. By induction on M and case analysis on the next reduction rule. By Lemma 54, translations of values are value normal forms in $\text{MET}(\mathcal{R}_{\text{SCP}})$.

Case E-APP We have

$$\llbracket (\lambda^E x^A. M) V \rrbracket = \text{let mod}_{\llbracket E \rrbracket} x = \text{mod}_{\llbracket E \rrbracket} (\lambda x^{\llbracket A \rrbracket}. \llbracket M \rrbracket) \text{ in } x \llbracket V \rrbracket$$

Our goal follows from E-LETMOD and E-APP in $\text{MET}(\mathcal{R}_{\text{SCP}})$. It is obvious that translation preserves value substitution.

Case E-TAPP By E-TAPP in $\text{MET}(\mathcal{R}_{\text{SCP}})$. It is obvious that translation preserves type substitution.

Case E-LET By syntactic sugar and E-APP in $\text{MET}(\mathcal{R}_{\text{SCP}})$.

Case E-HANDLER Suppose the effect row of the whole term is E .

$$\text{handler } H \ V \rightsquigarrow \text{handle } V \ () \text{ with } H$$

We have

$$\begin{aligned} \llbracket \text{LHS} \rrbracket &= \text{let mod}_{\llbracket E \rrbracket} x = \llbracket \text{handler } H \rrbracket \text{ in } x \llbracket V \rrbracket \\ \llbracket \text{handler } H \rrbracket &= \text{mod}_{\llbracket E \rrbracket} (\lambda f. \text{handle}^{\llbracket E \rrbracket} (\text{let mod}_{\llbracket \ell, E \rrbracket} f' = f \text{ in } f' ()) \text{ with } \llbracket H^E \rrbracket) \end{aligned}$$

By E-LETMOD and E-APP in $\text{MET}(\mathcal{R}_{\text{SCP}})$, $\llbracket \text{LHS} \rrbracket$ reduces to

$$\text{handle}^{\llbracket E \rrbracket} (\text{let mod}_{\llbracket \ell, E \rrbracket} f' = \llbracket V \rrbracket \text{ in } f' ()) \text{ with } \llbracket H^E \rrbracket$$

which is equal to $\llbracket \text{RHS} \rrbracket$ of the above reduction step.

Case E-RET By E-RET in $\text{MET}(\mathcal{R}_{\text{SCP}})$.

Case E-OP Suppose the effect row of the whole term is E .

$$\text{handle } \mathcal{E}[\text{do } \ell \ V] \text{ with } H \rightsquigarrow N[V/p, (\lambda y. \text{handle } \mathcal{E}[\text{return } y] \text{ with } H)/r]$$

where $\ell \notin \text{bl}(\mathcal{E})$ and $H \ni \ell \ p \ r \mapsto N$. We have

$$\llbracket \text{LHS} \rrbracket = \text{handle}^{\llbracket E \rrbracket} \llbracket \mathcal{E}[\text{do } \ell \ V] \rrbracket \text{ with } \llbracket H \rrbracket$$

By Lemma 55, we have

$$\llbracket \mathcal{E}[\text{do } \ell \ V] \rrbracket = \llbracket \mathcal{E} \rrbracket [\text{do } \ell \ \llbracket V \rrbracket]$$

Rule E-OP-MET($\mathcal{R}_{\text{SCP}}$) substitutes the raw continuation for r in the translated clause. The clause first reboxes it with $\llbracket E \rrbracket$ and binds the result to r' . One administrative E-APP step therefore gives

$$\llbracket N \rrbracket [\llbracket V \rrbracket / p, (\text{mod}_{\llbracket E \rrbracket} (\lambda y. \text{handle}^{\llbracket E \rrbracket} \llbracket \mathcal{E}[y] \rrbracket \text{ with } \llbracket H \rrbracket)) / r].$$

Translation preserves substitution, so this is $\llbracket \text{RHS} \rrbracket$.

Case E-LIFT By IH and Lemma 55.

□

C.2 Proofs of encoding System C in MET(S)

The source calculus and its encoding are defined in Sections 7.3.1 and 7.3.2. The proofs rely on the following lemmas.

Lemma 56 (Subeffecting). *Given a typing judgement $\Gamma \vdash M : A \mid C$ in System C, if $\llbracket \Gamma \rrbracket_C \vdash \llbracket M \rrbracket : \llbracket A \rrbracket @ \llbracket C \rrbracket$ and $C \subseteq C'$, then $\llbracket \Gamma \rrbracket_{C'} \vdash \llbracket M \rrbracket : \llbracket A \rrbracket @ \llbracket C' \rrbracket$. Similarly for values and blocks.*

Proof. By straightforward induction on typing judgements in System C. $\square$

Lemma 57 (Translation of Values and Blocks). *The translation of every System C value or block is a value in MET(S).*

Proof. Straightforward mutual induction on values and blocks. $\square$

Theorem 15 (Type Preservation). *If $\Gamma \vdash M : A \mid C$ in System C, then $\llbracket \Gamma \rrbracket_C \vdash \llbracket M \rrbracket : \llbracket A \rrbracket @ \llbracket C \rrbracket$ in MET(S). Similarly for typing judgements of values and blocks.*

Proof. By induction on typing judgements in System C. We use variable and label weakening to add fresh target bindings introduced by the translation. As a visual aid, for each non-trivial case we repeat its typing rule in System C. We replace each premise by the MET(S) judgement of the translated premise implied by the induction hypothesis. We replace the conclusion by the MET(S) judgement of the translated conclusion that we need to prove.

Case $\boxed{()}$ By T-UNIT-SYSTEM C and T-UNIT-MET(S).

Case $\boxed{x}$ By T-VAR-SYSTEM C and T-VAR-MET(S). Variables are always accessible after translation as translations of value types always have kind Abs.

Case $\boxed{\text{box } P}$

$$\frac{\llbracket \Gamma \rrbracket_C \vdash \llbracket P \rrbracket : \llbracket T \rrbracket @ \llbracket C \rrbracket \text{ ①}}{\llbracket \Gamma \rrbracket. \vdash \text{mod}_{\llbracket C \rrbracket} \llbracket P \rrbracket : \llbracket T \text{ at } C \rrbracket @.}$$

We have $\llbracket T \text{ at } C \rrbracket = \llbracket C \rrbracket \llbracket T \rrbracket$. By ①, lock composition, and context equivalence, we have

$$\llbracket \Gamma \rrbracket. \vdash_{\llbracket C \rrbracket} \llbracket P \rrbracket : \llbracket T \rrbracket @ \llbracket C \rrbracket$$

Our goal follows from T-MOD-MET(S).

Case $\boxed{f \text{ transparent}}$

$$\frac{\Gamma \ni f :^C T}{\llbracket \Gamma \rrbracket_C \vdash \hat{f} : \llbracket T \rrbracket @ \llbracket C \rrbracket}$$

Suppose $\Gamma = \Gamma_1, f :^C T, \Gamma_2$. We have

$$\llbracket \Gamma_1, f :^C T, \Gamma_2 \rrbracket_C = \llbracket \Gamma_1 \rrbracket, \hat{f} :_{\llbracket C \rrbracket} \llbracket T \rrbracket, \llbracket \Gamma_2 \rrbracket, \blacksquare_{\llbracket C \rrbracket}$$

Our goal follows from T-VAR-MET(S).

Case $\boxed{f \text{ tracked}}$

$$\frac{\Gamma \ni f :^* T}{\llbracket \Gamma \rrbracket_{\{f\}} \vdash \hat{f} : \llbracket T \rrbracket @ f^*}$$

Suppose $\Gamma = \Gamma_1, f :^* T, \Gamma_2$. We have

$$\llbracket \Gamma_1, f :^* T, \Gamma_2 \rrbracket_{\{f\}} = \llbracket \Gamma_1 \rrbracket, f^*, \hat{f} :_{\llbracket f^* \rrbracket} \llbracket T \rrbracket, \llbracket \Gamma_2 \rrbracket, \blacksquare_{\llbracket f^* \rrbracket}$$

Our goal follows from T-VAR-MET(S) and Lemma 45.

Case $\boxed{\{(x : \bar{A}, \bar{f} : \bar{T}) \Rightarrow M\}}$

$$\frac{\llbracket \Gamma, \bar{x} : \bar{A}, \bar{f} :^* T \rrbracket_{C \cup \{\bar{f}\}} \vdash \llbracket M \rrbracket : \llbracket B \rrbracket @ \llbracket C \cup \{\bar{f}\} \rrbracket \textcircled{1}}{\llbracket \Gamma \rrbracket_C \vdash \Lambda \bar{f}^*. \text{mod}_{\langle \bar{f}^* \rangle} (\lambda x \llbracket A \rrbracket f^{\llbracket f^* \rrbracket} \llbracket T \rrbracket}. \text{let mod}_{\llbracket f^* \rrbracket} \hat{f} = f \text{ in } \llbracket M \rrbracket) : \llbracket (\bar{A}, \bar{f} : \bar{T}) \Rightarrow B \rrbracket @ \llbracket C \rrbracket}$$

By $\textcircled{1}$, weakening, and exchange, we have

$$\llbracket \Gamma \rrbracket_C, \bar{f}^*, \blacksquare_{\langle \bar{f}^* \rangle}, x : \llbracket A \rrbracket, \bar{f} :_{\llbracket f^* \rrbracket} \llbracket T \rrbracket, \hat{f} :_{\llbracket f^* \rrbracket} \llbracket T \rrbracket \vdash \llbracket M \rrbracket : \llbracket B \rrbracket @ \llbracket C \rrbracket, \bar{f}^*$$

Our goal follows from T-ABS-MET(S), T-LETMOD-MET(S), and T-MOD-MET(S).

Case $\boxed{\text{unbox } V}$

$$\frac{\llbracket \Gamma \rrbracket. \vdash \llbracket V \rrbracket : \llbracket T \text{ at } C \rrbracket @ \cdot \textcircled{1}}{\llbracket \Gamma \rrbracket_C \vdash \text{let mod}_{\llbracket C \rrbracket} x = \llbracket V \rrbracket \text{ in } x : \llbracket T \rrbracket @ \llbracket C \rrbracket}$$

We have $\llbracket T \text{ at } C \rrbracket = \llbracket C \rrbracket \llbracket T \rrbracket$. By $\textcircled{1}$ and Lemma 56, we have

$$\llbracket \Gamma \rrbracket_C \vdash \llbracket V \rrbracket : \llbracket C \rrbracket \llbracket T \rrbracket @ \llbracket C \rrbracket$$

Our goal follows from T-LETMOD-MET(S) and MT-ABS ($\llbracket C \rrbracket \Rightarrow \langle \rangle @ \llbracket C \rrbracket$).

Case $\boxed{\text{let } x = M \text{ in } N}$ By IH, Lemma 56, T-LET-SYSTEM C, and the syntactic sugar of let-binding.

Case $\boxed{\text{def } f = P \text{ in } M}$

$$\frac{\llbracket \Gamma \rrbracket_{C'} \vdash \llbracket P \rrbracket : \llbracket T \rrbracket @ \llbracket C' \rrbracket \textcircled{1} \quad \llbracket \Gamma, f :^{C'} T \rrbracket_C \vdash \llbracket M \rrbracket : \llbracket A \rrbracket @ \llbracket C \rrbracket}{\llbracket \Gamma \rrbracket_C \vdash \text{let } f = \text{mod}_{\llbracket C' \rrbracket} \llbracket P \rrbracket \text{ in let mod}_{\llbracket C' \rrbracket} \hat{f} = f \text{ in } \llbracket M \rrbracket : \llbracket A \rrbracket @ \llbracket C \rrbracket}$$

By $\textcircled{1}$ and context equivalence, we have

$$\llbracket \Gamma \rrbracket_C, \blacksquare_{\llbracket C' \rrbracket} \vdash \llbracket P \rrbracket : \llbracket T \rrbracket @ \llbracket C' \rrbracket$$

Our goal follows from T-MOD-MET(S), T-LETMOD-MET(S), and the syntactic sugar of let-binding.

Case $\boxed{P(\overline{V_i}, \overline{Q_j})}$

$$\frac{\overline{\llbracket \Gamma \rrbracket. \vdash \llbracket V_i \rrbracket : \llbracket A_i \rrbracket @ \cdot} \textcircled{1} \quad \overline{\llbracket \Gamma \rrbracket_{C_j} \vdash \llbracket Q_j \rrbracket : \llbracket T_j \rrbracket @ \llbracket C_j \rrbracket} \textcircled{2}}{\overline{\llbracket \Gamma \rrbracket_C \vdash \llbracket P \rrbracket : \llbracket (\overline{A_i}, \overline{f_j} : \overline{T_j}) \Rightarrow B \rrbracket @ \llbracket C \rrbracket} \quad C' := C \cup \overline{C_j}} \quad \overline{\llbracket \Gamma \rrbracket_{C'} \vdash \text{let mod}_{\langle \llbracket C_j \rrbracket \rangle} x = \llbracket P \rrbracket \llbracket C_j \rrbracket \text{ in } x \llbracket V_i \rrbracket (\text{mod}_{\llbracket C_j \rrbracket} \llbracket Q_j \rrbracket) : \llbracket B \rrbracket [\llbracket C_j \rrbracket / \overline{f_j^*}] @ \llbracket C' \rrbracket}$$

We have $\llbracket (\overline{A_i}, \overline{f_j} : \overline{T_j}) \Rightarrow B \rrbracket = \forall \overline{f_j^*}. \langle \overline{f_j^*} \rangle (\llbracket A_i \rrbracket \rightarrow \llbracket \overline{f_j^*} \rrbracket \llbracket T_j \rrbracket \rightarrow \llbracket B \rrbracket)$. By $\textcircled{1}$ and Lemma 56 we have

$$\overline{\llbracket \Gamma \rrbracket_{C'} \vdash \llbracket V_i \rrbracket : \llbracket A_i \rrbracket @ \llbracket C' \rrbracket} \textcircled{3}$$

By $\textcircled{2}$ and context equivalence, we have

$$\overline{\llbracket \Gamma \rrbracket, \blacksquare_{\llbracket C_j \rrbracket} \vdash \llbracket Q_j \rrbracket : \llbracket T_j \rrbracket @ \llbracket C_j \rrbracket}$$

Then by T-MOD-MET(S) we have

$$\overline{\llbracket \Gamma \rrbracket_{C'} \vdash \text{mod}_{\llbracket C_j \rrbracket} \llbracket Q_j \rrbracket : \llbracket C_j \rrbracket \llbracket T_j \rrbracket @ \llbracket C' \rrbracket} \textcircled{4}$$

Also note that translation preserves type substitution of capability variables, which gives

$$\overline{\llbracket B \rrbracket [\llbracket C_j \rrbracket / \overline{f_j^*}] = \llbracket B[C_j / \overline{f_j}] \rrbracket} \textcircled{5}$$

By Lemma 56, the IH for P also holds at $\llbracket C' \rrbracket$. Finally our goal follows from this judgement, $\textcircled{3}$, $\textcircled{4}$, $\textcircled{5}$, T-LETMOD-MET(S), T-APP-MET(S), and T-TAPP-MET(S).

Case $\boxed{\text{return } V}$ By IH.

Case $\boxed{\text{subtyping of blocks and computations}}$ By IH and Lemma 56.

Case $\boxed{\text{try } \{f^{A' \Rightarrow B'} \Rightarrow M\} \text{ with } \{p \ r \mapsto N\}}$

$$\frac{\overline{\llbracket \Gamma, f :^* A' \Rightarrow B' \rrbracket_{C \cup \{f\}} \vdash \llbracket M \rrbracket : \llbracket A \rrbracket @ \llbracket C \cup \{f\} \rrbracket} \textcircled{1}}{\overline{\llbracket \Gamma, p : A', r :^C B' \Rightarrow A \rrbracket_C \vdash \llbracket N \rrbracket : \llbracket A \rrbracket @ \llbracket C \rrbracket} \textcircled{2}} \quad \overline{\llbracket \Gamma \rrbracket_C \vdash \text{local } \ell_f : \llbracket A' \rrbracket \rightarrow \llbracket B' \rrbracket \text{ in let}_{\llbracket C \rrbracket} \text{mod}_{\langle \ell_f \rangle} g = M_1 \text{ in } : \llbracket A \rrbracket @ \llbracket C \rrbracket \text{ handle}^{\llbracket C \rrbracket} g \ M_2 \text{ with } \{\text{return } x \mapsto x, \ell_f \ p \ r \mapsto N_2\}}$$

where

$$\begin{aligned} M_1 &= (\Lambda f^*. \text{mod}_{\langle f^* \rangle} (\lambda f. \text{let mod}_{[f^*]} \hat{f} = f \text{ in } \llbracket M \rrbracket)) \ell_f \\ M_2 &= \text{mod}_{[\ell_f]} (\text{mod}_{\langle \rangle} (\lambda x^{\llbracket A' \rrbracket}. \text{do } \ell_f \ x)) \\ N_2 &= \text{let } r' = \text{mod}_{\llbracket C \rrbracket} (\text{mod}_{\langle \rangle} r) \text{ in let mod}_{\llbracket C \rrbracket} \hat{r} = r' \text{ in } \llbracket N \rrbracket \end{aligned}$$

For the translations of the contexts in ① and ②, we have:

$$\llbracket \Gamma, f :^* A' \Rightarrow B' \rrbracket_{\text{CU}\{f\}} = \llbracket \Gamma \rrbracket, f^*, \hat{f} :_{[f^*]} \llbracket A' \Rightarrow B' \rrbracket \blacksquare_{\llbracket \text{CU}\{f\} \rrbracket}$$

$$\llbracket \Gamma, p : A', r :^C B' \Rightarrow A \rrbracket_C = \llbracket \Gamma \rrbracket, p : \llbracket A' \rrbracket, \hat{r} :_{\llbracket C \rrbracket} \llbracket B' \Rightarrow A \rrbracket, \blacksquare_{\llbracket C \rrbracket}$$

Let $\Gamma_1 = \llbracket \Gamma \rrbracket, \ell_f : \llbracket A' \rrbracket \rightarrow \llbracket B' \rrbracket$ and $A_1 = [\ell_f] \langle \rangle (\llbracket A' \rrbracket \rightarrow \llbracket B' \rrbracket)$. By ①, weakening, exchange, and the rules for the translation of a block, we have

$$\Gamma_1, \blacksquare_{\llbracket C \rrbracket} \vdash M_1 : \langle \ell_f \rangle (A_1 \rightarrow \llbracket A \rrbracket) @ \llbracket C \rrbracket \quad \textcircled{3}$$

The outer T-LETMOD-MET(S) rule therefore binds

$$g :_{\llbracket C \rrbracket \circ \langle \ell_f \rangle} A_1 \rightarrow \llbracket A \rrbracket \equiv g :_{[\ell_f, \llbracket C \rrbracket]} A_1 \rightarrow \llbracket A \rrbracket$$

By various typing rules, we have

$$\Gamma_1, g :_{\llbracket C \rrbracket \circ \langle \ell_f \rangle} A_1 \rightarrow \llbracket A \rrbracket \vdash g M_2 : \llbracket A \rrbracket @ \ell_f, \llbracket C \rrbracket$$

By Lemma 51, we obtain the handled-computation premise

$$\Gamma_1, g :_{\llbracket C \rrbracket \circ \langle \ell_f \rangle} A_1 \rightarrow \llbracket A \rrbracket, \blacksquare_{\llbracket C \rrbracket}, \blacksquare_{\langle \ell_f \rangle} \vdash g M_2 : \llbracket A \rrbracket @ \ell_f, \llbracket C \rrbracket \quad \textcircled{4}$$

By ② and various typing rules, we have

$$\Gamma_1, g :_{\llbracket C \rrbracket \circ \langle \ell_f \rangle} A_1 \rightarrow \llbracket A \rrbracket, p : \llbracket A' \rrbracket, r :_{\llbracket C \rrbracket} \llbracket B' \rrbracket \rightarrow \llbracket A \rrbracket \vdash N_2 : \llbracket A \rrbracket @ \llbracket C \rrbracket$$

Again by Lemma 51, we obtain the operation-clause premise

$$\Gamma_1, g :_{\llbracket C \rrbracket \circ \langle \ell_f \rangle} A_1 \rightarrow \llbracket A \rrbracket, \blacksquare_{\llbracket C \rrbracket}, p : \llbracket A' \rrbracket, r :_{\llbracket C \rrbracket} \llbracket B' \rrbracket \rightarrow \llbracket A \rrbracket \vdash N_2 : \llbracket A \rrbracket @ \llbracket C \rrbracket \quad \textcircled{5}$$

Since $\llbracket A \rrbracket : \text{Abs}$, T-VAR-MET(S) gives the return-clause premise

$$\Gamma_1, g :_{\llbracket C \rrbracket \circ \langle \ell_f \rangle} A_1 \rightarrow \llbracket A \rrbracket, \blacksquare_{\llbracket C \rrbracket}, x :_{\llbracket C \rrbracket \circ \langle \ell_f \rangle} \llbracket A \rrbracket \vdash x : \llbracket A \rrbracket @ \llbracket C \rrbracket \quad \textcircled{6}$$

By ④, ⑤, ⑥, and T-HANDLE-MET(S), we have

$$\begin{aligned} &\Gamma_1, g :_{\llbracket C \rrbracket \circ \langle \ell_f \rangle} A_1 \rightarrow \llbracket A \rrbracket \vdash \\ &\quad \text{handle}^{\llbracket C \rrbracket} g M_2 \text{ with } \{\text{return } x \mapsto x, \ell_f p r \mapsto N_2\} : \llbracket A \rrbracket @ \llbracket C \rrbracket \end{aligned}$$

Finally, ③, T-LETMOD-MET(S), and T-LOCALEFFECT-MET(S) give

$$\llbracket \Gamma \rrbracket_C \vdash \llbracket \text{try } \{f^{A' \Rightarrow B'} \Rightarrow M\} \text{ with } \{p r \mapsto N\} \rrbracket : \llbracket A \rrbracket @ \llbracket C \rrbracket$$

□

C.3 Proofs of encoding System Ξ in MET(S)

The source calculus and its encoding are defined in Sections 7.3.3 and 7.3.4. The proofs rely on the following lemmas.

Lemma 58 (Translation of Values and Blocks). *If V is a value in System Ξ , then $\llbracket V \rrbracket$ is a value normal form in $\text{MET}(\mathcal{S})$. If P is a block in System Ξ , then $\llbracket P \rrbracket$ is a value normal form in $\text{MET}(\mathcal{S})$.*

Proof. By mutual induction on values and blocks. □

Lemma 59 (Translation of Evaluation Contexts). *For the translation $\llbracket - \rrbracket$ from System Ξ to $\text{MET}(\mathcal{S})$, we have $\llbracket \mathcal{E}[M] \rrbracket = \llbracket \mathcal{E} \rrbracket[\llbracket M \rrbracket]$ for any evaluation context $\mathcal{E}$ and term M .*

Proof. By straightforward induction on evaluation contexts of System Ξ . □

Theorem 16 (Type Preservation). *If $\Gamma \vdash M : A$ in System Ξ , then $\llbracket \Gamma \rrbracket \vdash \llbracket M \rrbracket : \llbracket A \rrbracket @ \cdot$ in $\text{MET}(\mathcal{S})$. Similarly for values and blocks.*

Proof. By induction on typing judgements in System Ξ . As a visual aid, for each non-trivial case we repeat its typing rule in System Ξ . We replace each premise by the $\text{MET}(\mathcal{S})$ judgement of the translated premise implied by the induction hypothesis. We replace the conclusion by the $\text{MET}(\mathcal{S})$ judgement of the translated conclusion that we need to prove.

Case $\boxed{()}$ By T-UNIT-SYSTEM Ξ and T-UNIT-MET($\mathcal{S}$).

Case $\boxed{x}$ By T-VAR-SYSTEM Ξ and T-VAR-MET($\mathcal{S}$). Variables are always accessible after translation because translations of value types have kind Abs.

Case $\boxed{f}$ By T-BLOCKVAR-SYSTEM Ξ and T-VAR-MET($\mathcal{S}$).

Case $\boxed{\{(x : \bar{A}, \bar{f} : \bar{T}) \Rightarrow M\}}$

$$\frac{\llbracket \Gamma \rrbracket, x : \llbracket \bar{A} \rrbracket, \bar{f} : \llbracket \bar{T} \rrbracket \vdash \llbracket M \rrbracket : \llbracket B \rrbracket @ \cdot}{\llbracket \Gamma \rrbracket \vdash \lambda x^{\llbracket \bar{A} \rrbracket} \bar{f}^{\llbracket \bar{T} \rrbracket}. \llbracket M \rrbracket : \llbracket (\bar{A}, \bar{T}) \Rightarrow B \rrbracket @ \cdot}$$

We have $\llbracket (\bar{A}, \bar{T}) \Rightarrow B \rrbracket = \llbracket \bar{A} \rrbracket \rightarrow \llbracket \bar{T} \rrbracket \rightarrow \llbracket B \rrbracket$. Our goal follows from T-ABS-MET($\mathcal{S}$).

Case $\boxed{\text{return } V}$ By IH.

Case $\boxed{\text{let } x = M \text{ in } N}$ By IH, T-LET-SYSTEM Ξ , and the syntactic sugar of let-binding.

Case $\boxed{\text{def } f = P \text{ in } N}$ By IH, T-DEF-SYSTEM Ξ , and the syntactic sugar of let-binding.

Case $\boxed{P(\overline{V_i}, \overline{Q_j})}$

$$\frac{\frac{\frac{\llbracket \Gamma \rrbracket \vdash \llbracket P \rrbracket : \llbracket (\overline{A_i}, \overline{T_j}) \Rightarrow B \rrbracket @ \cdot}{\llbracket \Gamma \rrbracket \vdash \llbracket V_i \rrbracket : \llbracket A_i \rrbracket @ \cdot} \quad \frac{}{\llbracket \Gamma \rrbracket \vdash \llbracket Q_j \rrbracket : \llbracket T_j \rrbracket @ \cdot}}{\llbracket \Gamma \rrbracket \vdash \llbracket P \rrbracket \overline{\llbracket V_i \rrbracket} \overline{\llbracket Q_j \rrbracket} : \llbracket B \rrbracket @ \cdot}$$

We have $\llbracket (\overline{A_i}, \overline{T_j}) \Rightarrow B \rrbracket = \overline{\llbracket A_i \rrbracket} \rightarrow \overline{\llbracket T_j \rrbracket} \rightarrow \llbracket B \rrbracket$. Our goal follows from T-APP-MET(S).

Case $\boxed{\text{try } \{f^{A' \Rightarrow B'} \Rightarrow M\} \text{ with } \{p \ r \mapsto N\}}$

$$\frac{\frac{\llbracket \Gamma \rrbracket, f : \llbracket A' \Rightarrow B' \rrbracket \vdash \llbracket M \rrbracket : \llbracket A \rrbracket @ \cdot \textcircled{1}}{\llbracket \Gamma \rrbracket, p : \llbracket A' \rrbracket, r : \llbracket B' \Rightarrow A \rrbracket \vdash \llbracket N \rrbracket : \llbracket A \rrbracket @ \cdot \textcircled{2}}}{\frac{\llbracket \Gamma \rrbracket \vdash \text{local } \ell_f : \llbracket A' \rrbracket \rightarrow \llbracket B' \rrbracket \text{ in } \text{handle } (\lambda f. \llbracket M \rrbracket) (\lambda x^{\llbracket A' \rrbracket}. \text{do } \ell_f \ x) \text{ with } \{\text{return } x \mapsto x, \ell_f \ p \ r \mapsto \llbracket N \rrbracket\}}{\llbracket \Gamma \rrbracket \vdash \text{local } \ell_f : \llbracket A' \rrbracket \rightarrow \llbracket B' \rrbracket \text{ in } \text{handle } (\lambda f. \llbracket M \rrbracket) (\lambda x^{\llbracket A' \rrbracket}. \text{do } \ell_f \ x) \text{ with } \{\text{return } x \mapsto x, \ell_f \ p \ r \mapsto \llbracket N \rrbracket\}} : \llbracket A \rrbracket @ \cdot}$$

By $\textcircled{1}$, $\llbracket A' \Rightarrow B' \rrbracket = \llbracket A' \rrbracket \rightarrow \llbracket B' \rrbracket$, MT-REL, lock and label weakening, and context equivalence, we have

$$\llbracket \Gamma \rrbracket, \ell_f : \llbracket A' \rrbracket \rightarrow \llbracket B' \rrbracket, \mathbf{a}_{\langle \ell_f \rangle}, f : \llbracket A' \rrbracket \rightarrow \llbracket B' \rrbracket \vdash \llbracket M \rrbracket : \llbracket A \rrbracket @ \ell_f$$

which further gives the handled-computation premise

$$\llbracket \Gamma \rrbracket, \ell_f : \llbracket A' \rrbracket \rightarrow \llbracket B' \rrbracket, \mathbf{a}_{\langle \ell_f \rangle} \vdash (\lambda f. \llbracket M \rrbracket) (\lambda x^{\llbracket A' \rrbracket}. \text{do } \ell_f \ x) : \llbracket A \rrbracket @ \ell_f \textcircled{3}$$

By $\textcircled{2}$, $\llbracket B' \Rightarrow A \rrbracket = \llbracket B' \rrbracket \rightarrow \llbracket A \rrbracket$, label weakening, and context equivalence, we have the operation-clause premise

$$\llbracket \Gamma \rrbracket, \ell_f : \llbracket A' \rrbracket \rightarrow \llbracket B' \rrbracket, p : \llbracket A' \rrbracket, r : \langle \rangle \llbracket B' \rrbracket \rightarrow \llbracket A \rrbracket \vdash \llbracket N \rrbracket : \llbracket A \rrbracket @ \cdot \textcircled{4}$$

Since $\llbracket A \rrbracket : \text{Abs}$, T-VAR-MET(S) gives the return-clause premise

$$\llbracket \Gamma \rrbracket, \ell_f : \llbracket A' \rrbracket \rightarrow \llbracket B' \rrbracket, x : \langle \rangle \circ \langle \ell_f \rangle \llbracket A \rrbracket \vdash x : \llbracket A \rrbracket @ \cdot \textcircled{5}$$

Our goal follows from $\textcircled{3}$, $\textcircled{4}$, $\textcircled{5}$, context equivalence, T-HANDLE-MET(S), and T-LOCAL-EFFECT-MET(S).

□

Theorem 17 (Semantics Preservation). *If M is well-typed and $M \mid \Omega \rightsquigarrow N \mid \Omega'$ in System Ξ , then $\llbracket M \rrbracket \mid \llbracket \Omega \rrbracket \rightsquigarrow^* \llbracket N \rrbracket \mid \llbracket \Omega' \rrbracket$ in MET(S).*

Proof. By induction on M and case analysis on the next reduction rule.

Case $\boxed{\text{E-LET}}$ Let-binding in MET(S) is syntactic sugar of lambda application. By E-APP-MET(S).

Case $\boxed{\text{E-DEF}}$ Similar to the above case.

Case $\boxed{\text{E-CALL}}$

$$\{(x : \overline{A}, \overline{f} : \overline{T}) \Rightarrow M\}(\overline{V}, \overline{Q}) \rightsquigarrow M[\overline{V}/x, \overline{Q}/\overline{f}]$$

We have

$$\llbracket \{(x : \overline{A}, \overline{f} : \overline{T}) \Rightarrow M\}(\overline{V}, \overline{Q}) \rrbracket = (\lambda x \llbracket \overline{A} \rrbracket \overline{f} \llbracket \overline{T} \rrbracket. \llbracket M \rrbracket) \llbracket \overline{V} \rrbracket \llbracket \overline{Q} \rrbracket$$

By multiple usages of $\text{E-APP-MET}(\mathcal{S})$.

Case $\boxed{\text{E-GEN}}$

$$\text{try } \{f^{A' \Rightarrow B'} \Rightarrow M\} \text{ with } H \mid \Omega \rightsquigarrow \text{try}_\ell M[\text{cap}_\ell/f] \text{ with } H \mid \Omega, \ell : A' \Rightarrow B'$$

We have

$$\llbracket \text{LHS} \rrbracket = \text{local } \ell_f : \llbracket A' \rrbracket \rightarrow \llbracket B' \rrbracket \text{ in handle } (\lambda f. \llbracket M \rrbracket) (\lambda x. \text{do } \ell_f x) \text{ with } \llbracket H \rrbracket$$

$$\llbracket \text{RHS} \rrbracket = \text{handle } \llbracket M[\text{cap}_\ell/f] \rrbracket \text{ with } \llbracket H \rrbracket$$

$$\llbracket \text{cap}_\ell \rrbracket = \lambda x. \text{do } \ell x$$

Our goal follows from $\text{E-GEN-MET}(\mathcal{S})$, $\text{E-APP-MET}(\mathcal{S})$, and the fact that translation preserves substitution.

Case $\boxed{\text{E-RET}}$ By E-RET in $\text{MET}(\mathcal{S})$.

Case $\boxed{\text{E-OP}}$

$$\text{try}_\ell \mathcal{E}[\text{cap}_\ell(V)] \text{ with } H \mid \Omega \rightsquigarrow N[V/p, \{(y) \Rightarrow \text{try}_\ell \mathcal{E}[\text{return } y] \text{ with } H\}/r] \mid \Omega$$

where $\Omega \ni \ell : A' \Rightarrow B'$. We have

$$\llbracket \text{LHS} \rrbracket = \text{handle } \llbracket \mathcal{E}[\text{cap}_\ell(V)] \rrbracket \text{ with } \llbracket H^\ell \rrbracket$$

By Lemma 59, we have

$$\llbracket \mathcal{E}[\text{cap}_\ell(V)] \rrbracket = \llbracket \mathcal{E} \rrbracket \llbracket \llbracket \text{cap}_\ell(V) \rrbracket \rrbracket = \llbracket \mathcal{E} \rrbracket \llbracket (\lambda x. \text{do } \ell x) \llbracket V \rrbracket \rrbracket$$

Our goal follows from $\text{E-APP-MET}(\mathcal{S})$ and $\text{E-OP-MET}(\mathcal{S})$.

Case $\boxed{\text{E-LIFT}}$ Follows from IH and Lemma 59.

□

C.4 Proofs of encoding System $F^{\epsilon+sn}$ in $\text{MET}(\mathcal{S})$

The source calculus and its encoding are defined in Sections 7.2.3 and 7.2.4. The proofs rely on the following lemmas.

Lemma 60 (Pure Values). *Given a typing judgement $\Gamma \vdash V : A$ in System $F^{\epsilon+sn}$, if $\llbracket \Gamma \rrbracket. \vdash \llbracket V \rrbracket : \llbracket A \rrbracket @ \cdot$ then $\llbracket \Gamma \rrbracket_E \vdash \llbracket V \rrbracket : \llbracket A \rrbracket @ \llbracket E \rrbracket$ for any E .*

Proof. By lock weakening and MT-Abs. □

Lemma 61 (Translation of Values). *If V is a value in System $F^{\epsilon+sn}$, then $\llbracket V \rrbracket$ is a value normal form in $MET(S)$.*

Proof. By induction on V . The cases shared with System F^ϵ follow as in Lemma 54. Named handlers and runtime evidence translate to modality introductions whose enclosed terms are lambda abstractions. $\square$

Lemma 62 (Translation of Evaluation Contexts). *For the translation $\llbracket - \rrbracket$ from System $F^{\epsilon+sn}$ to $MET(S)$, we have $\llbracket \mathcal{E}[M] \rrbracket = \llbracket \mathcal{E} \rrbracket[\llbracket M \rrbracket]$ for any evaluation context $\mathcal{E}$ and term M . Moreover, supposing no two entries of Ω contain the same scope variable, if $\Omega \ni h : \ell^b$ and $h \notin BH(\mathcal{E})$, then $\ell_b \notin bl(\llbracket \mathcal{E} \rrbracket)$.*

Proof. The first claim follows as in Lemma 55. The second claim follows by induction on $\mathcal{E}$. The only non-trivial case is $\mathcal{E} = \text{handle}_{h'} \mathcal{E}'$ with H . We have $h \neq h'$ by assumption. Since distinct entries of Ω have distinct scope variables, the labels that translate h and h' are distinct. The goal then follows from the IH on $\mathcal{E}'$. $\square$

Theorem 13 (Type Preservation). *If $\Gamma \vdash M : A \mid E$ in System $F^{\epsilon+sn}$, then $\llbracket \Gamma \rrbracket_E \vdash \llbracket M \rrbracket : \llbracket A \rrbracket @ \llbracket E \rrbracket$ in $MET(S)$. Similarly for typing judgements of values.*

Proof. By induction on typing judgements $\Gamma \vdash M : A \mid E$ in System $F^{\epsilon+sn}$. We use variable and label weakening to add fresh bindings introduced by the translation. The cases shared with System F^ϵ follow as in its proof in Appendix C.1. We elaborate cases relevant to named handlers.

Case $\boxed{\text{do } V \ W}$

$$\frac{\text{T-DoNAME} \quad \Sigma \ni \ell : A \twoheadrightarrow B \quad \Gamma \vdash V : \text{ev } \ell^a \text{ ①} \quad \Gamma \vdash W : A \text{ ②}}{\Gamma \vdash \text{do } V \ W : B \mid \ell^a, E}$$

By IH on ① and ② and Lemma 60, we have

$$\begin{aligned} \llbracket \Gamma \rrbracket_{\ell^a, E} \vdash \llbracket V \rrbracket : \llbracket a \rrbracket(\llbracket A \rrbracket \rightarrow \llbracket B \rrbracket) @ a, \llbracket E \rrbracket \\ \llbracket \Gamma \rrbracket_{\ell^a, E} \vdash \llbracket W \rrbracket : \llbracket A \rrbracket @ a, \llbracket E \rrbracket \end{aligned}$$

By T-LETMOD- $MET(S)$ and T-APP- $MET(S)$, we have

$$\llbracket \Gamma \rrbracket_{\ell^a, E} \vdash \text{let mod}_{[a]} x = \llbracket V \rrbracket \text{ in } x \llbracket W \rrbracket : \llbracket B \rrbracket @ a, \llbracket E \rrbracket$$

Case $\boxed{\text{nhandler } \{\ell \ p \ r \mapsto N\}}$

$$\frac{\text{T-NAMEDHANDLER} \quad \Sigma \ni \ell : A' \twoheadrightarrow B' \quad \Gamma, p : A', r : B' \rightarrow^E A \vdash N : A \mid E \text{ ①}}{\Gamma \vdash \text{nhandler } \{\ell \ p \ r \mapsto N\} : (\forall a^{\text{Scope}(\ell)} . \text{ev } \ell^a \rightarrow^{\ell^a, E} A) \rightarrow^E A}$$

Let

$$\begin{aligned} A_1 &= \forall a^{\text{Effect}}. [a, \llbracket E \rrbracket]([a](\llbracket A' \rrbracket \rightarrow \llbracket B' \rrbracket) \rightarrow \llbracket A \rrbracket) \\ M_0 &= \text{let } x = f \ell_a \text{ in let mod}_{[\ell_a, \llbracket E \rrbracket]} f' = x \text{ in } f' (\text{mod}_{[\ell_a]}(\lambda x^{\llbracket A' \rrbracket}. \text{do } \ell_a x)) \\ N_2 &= \text{let } r' = \text{mod}_{[\llbracket E \rrbracket]} r \text{ in } \llbracket N \rrbracket[r'/r] \\ \Gamma_1 &= [\Gamma], \blacksquare_{[\llbracket E \rrbracket]}, f : A_1, \ell_a : \llbracket A' \rrbracket \twoheadrightarrow \llbracket B' \rrbracket \end{aligned}$$

By the IH on ①, alpha-renaming its translated continuation binding to r' , Lemma 51, weakening, and exchange, we have

$$\begin{aligned} \Gamma_1, \blacksquare_{[\llbracket E \rrbracket]}, p : \llbracket A' \rrbracket, r : [\llbracket E \rrbracket] \llbracket B' \rrbracket \rightarrow \llbracket A \rrbracket, r' : [\llbracket E \rrbracket](\llbracket B' \rrbracket \rightarrow \llbracket A \rrbracket) & \text{②} \\ \vdash \llbracket N \rrbracket[r'/r] : \llbracket A \rrbracket @ [\llbracket E \rrbracket] & \end{aligned}$$

The type of r' is the translation of $B' \rightarrow^E A$. By ②, T-VAR-MET($\mathcal{S}$), T-MOD-MET($\mathcal{S}$), and the syntactic sugar of let-binding, we obtain the operation-clause premise

$$\Gamma_1, \blacksquare_{[\llbracket E \rrbracket]}, p : \llbracket A' \rrbracket, r : [\llbracket E \rrbracket] \llbracket B' \rrbracket \rightarrow \llbracket A \rrbracket \vdash N_2 : \llbracket A \rrbracket @ [\llbracket E \rrbracket] \text{ ③}$$

Since $\llbracket A \rrbracket : \text{Abs}$, T-VAR-MET($\mathcal{S}$) gives the return-clause premise

$$\Gamma_1, \blacksquare_{[\llbracket E \rrbracket]}, x : [\llbracket E \rrbracket] \circ \langle \ell_a \rangle \llbracket A \rrbracket \vdash x : \llbracket A \rrbracket @ [\llbracket E \rrbracket] \text{ ④}$$

By various typing rules, we obtain the handled-computation premise

$$\Gamma_1, \blacksquare_{[\llbracket E \rrbracket]}, \blacksquare_{\langle \ell_a \rangle} \vdash M_0 : \llbracket A \rrbracket @ \ell_a, [\llbracket E \rrbracket] \text{ ⑤}$$

By ③, ④, ⑤, and T-HANDLE-MET($\mathcal{S}$), we have

$$\Gamma_1 \vdash \text{handle}^{[\llbracket E \rrbracket]} M_0 \text{ with } \{\text{return } x \mapsto x, \ell_a p r \mapsto N_2\} : \llbracket A \rrbracket @ [\llbracket E \rrbracket]$$

By T-LOCALEFFECT-MET($\mathcal{S}$), T-ABS-MET($\mathcal{S}$), and T-MOD-MET($\mathcal{S}$), we conclude

$$[\Gamma]. \vdash \llbracket \text{nhandler } \{\ell p r \mapsto N\} \rrbracket : \llbracket (\forall a^{\text{Scope}(\ell)}. \text{ev } \ell^a \rightarrow^{\ell^a, E} A) \rightarrow^E A \rrbracket @ \cdot$$

□

Theorem 14 (Semantics Preservation). *Suppose no two entries of Ω contain the same scope variable. If M is well-typed and $M \mid \Omega \rightsquigarrow N \mid \Omega'$ in System $F^{\epsilon+sn}$, then $\llbracket M \rrbracket \mid \llbracket \Omega \rrbracket \rightsquigarrow^* \llbracket N \rrbracket \mid \llbracket \Omega' \rrbracket$ in $\text{MET}(\mathcal{S})$, where $\rightsquigarrow^*$ denotes the reflexive-transitive closure of $\rightsquigarrow$.*

Proof. By induction on M and case analysis on the next reduction rule. By Lemma 61, translations of values are value normal forms in $\text{MET}(\mathcal{S})$. The cases shared with System F^ϵ follow as in its proof in Appendix C.1. We show new cases relevant to named handlers.

Case E-GEN Suppose the effect row of the whole term is E .

$$\text{nhandler } H \mid V \mid \Omega \rightsquigarrow \text{handle}_h (\text{let } x = V \text{ b in } x \text{ ev}_h) \text{ with } H \mid \Omega, h : \ell^b$$

where b, h fresh and $\Sigma \ni \ell : A' \twoheadrightarrow B'$. We have

$$\begin{aligned} \llbracket \text{LHS} \rrbracket &= \text{let mod}_{\llbracket E \rrbracket} g = \llbracket \text{nhandler } H \rrbracket \text{ in } g \llbracket V \rrbracket \\ \llbracket \text{nhandler } H \rrbracket &= \text{mod}_{\llbracket E \rrbracket} (\lambda f. \text{local } \ell_a : \llbracket A' \rrbracket \twoheadrightarrow \llbracket B' \rrbracket \text{ in} \\ &\quad \text{handle}^{\llbracket E \rrbracket} (\text{let } x = f \ell_a \text{ in let mod}_{[\ell_a, \llbracket E \rrbracket]} f' = x \text{ in} \\ &\quad \quad f' (\text{mod}_{[\ell_a]} (\lambda x. \text{do } \ell_a x))) \\ &\quad \text{with } \llbracket H \rrbracket) \end{aligned}$$

By E-LETMOD, E-APP, and E-GEN, using the runtime label ℓ_b , $\llbracket \text{LHS} \rrbracket$ reduces in $\text{MET}(\mathcal{S})$ to

$$\begin{aligned} &\text{handle}^{\llbracket E \rrbracket} (\text{let } x = \llbracket V \rrbracket \ell_b \text{ in} \\ &\quad \text{let mod}_{[\ell_b, \llbracket E \rrbracket]} f' = x \text{ in } f' (\text{mod}_{[\ell_b]} (\lambda x. \text{do } \ell_b x))) \text{ with } \llbracket H \rrbracket \end{aligned}$$

which is equal to $\llbracket \text{RHS} \rrbracket$ of the above reduction step. Note that the runtime generated scope variable b is translated to ℓ_b .

Case $\boxed{\text{E-NRET}}$ By E-RET in $\text{MET}(\mathcal{S})$.

Case $\boxed{\text{E-NOP}}$ Suppose the effect row of the whole term is E .

$$\text{handle}_h \mathcal{E}[\text{do ev}_h V] \text{ with } H \rightsquigarrow N[V/p, (\lambda y. \text{handle}_h \mathcal{E}[\text{return } y] \text{ with } H)/r]$$

where $\Omega \ni h : \ell^b$ and $h \notin \text{BH}(\mathcal{E})$. We have

$$\llbracket \text{LHS} \rrbracket = \text{handle}^{\llbracket E \rrbracket} \llbracket \mathcal{E}[\text{do ev}_h V] \rrbracket \text{ with } \llbracket H \rrbracket$$

By Lemma 62, we have

$$\begin{aligned} \llbracket \mathcal{E}[\text{do ev}_h V] \rrbracket &= \llbracket \mathcal{E} \rrbracket [\text{let mod}_{[\ell_b]} f = \llbracket \text{ev}_h \rrbracket \text{ in } f \llbracket V \rrbracket] \\ &= \llbracket \mathcal{E} \rrbracket [\text{let mod}_{[\ell_b]} f = \text{mod}_{[\ell_b]} (\lambda x. \text{do } \ell_b x) \text{ in } f \llbracket V \rrbracket] \end{aligned}$$

Then by E-LETMOD and E-APP in $\text{MET}(\mathcal{S})$, $\llbracket \text{LHS} \rrbracket$ reduces to

$$\text{handle}^{\llbracket E \rrbracket} \llbracket \mathcal{E} \rrbracket [\text{do } \ell_b \llbracket V \rrbracket] \text{ with } \llbracket H \rrbracket$$

Lemma 62 gives $\ell_b \notin \text{bl}(\llbracket \mathcal{E} \rrbracket)$, as required by the E-OP rule. Rule E-OP- $\text{MET}(\mathcal{S})$ substitutes the raw continuation for r in the translated operation clause. The clause first reboxes it with $\llbracket E \rrbracket$ and binds the result to r' . One administrative E-APP step therefore gives the translation of the source continuation. Translation preserves substitution, so the resulting term is $\llbracket \text{RHS} \rrbracket$.

□

C.5 Proofs of encoding $\mathbf{F}^{1\epsilon}$ in $\text{MET}(\mathcal{R}_{\text{scp}})$

The source calculus and its encoding are defined in Sections 7.4.1 and 7.4.2. This section proves Theorem 18.

The target calculus $\text{MET}(\mathcal{R}_{\text{scp}})$ is extended with masks. We have the following lemmas on relative modalities.

Lemma 63 (Relative modality transformation). *If $L_1 \leq F$ and $L_2 \leq F$ and $L_1 - D_1 \equiv L_2 - D_2$ and $D_1 - L_1 \equiv D_2 - L_2$, then $\Gamma \vdash \langle L_1 | D_1 \rangle \Rightarrow \langle L_2 | D_2 \rangle @ F$.*

Proof. By the two equivalence premises and MT-REL, we have $\Gamma \vdash \langle L_1 - D_1 | D_1 - L_1 \rangle \Rightarrow \langle L_2 - D_2 | D_2 - L_2 \rangle @ F$. Since $L_1 \leq F$ and $L_2 \leq F$, our goal follows by repeatedly applying MT-SHRINK and MT-EXPAND. $\square$

Lemma 64 (First Modality Transformation). *For all E_1, E_2, E_3 :*

$$\Gamma \vdash \langle E_1 - E_2 | E_2 - E_1 \rangle \circ \langle E_2 - E_3 | E_3 - E_2 \rangle \Rightarrow \langle E_1 - E_3 | E_3 - E_1 \rangle @ E_1$$

$$\Gamma \vdash \langle E_1 - E_3 | E_3 - E_1 \rangle \Rightarrow \langle E_1 - E_2 | E_2 - E_1 \rangle \circ \langle E_2 - E_3 | E_3 - E_2 \rangle @ E_1$$

Proof. Suppose the composition result is $\langle L | D \rangle$. Straightforward row calculation gives

$$L - D \equiv E_1 - E_3 \quad \text{and} \quad D - L \equiv E_3 - E_1.$$

Moreover, $L \leq E_1$ and $E_1 - E_3 \leq E_1$. Both goals follow from Lemma 63. $\square$

Lemma 65 (Second Modality Transformation). *For all L, E, F :*

$$\Gamma \vdash \langle L, (E - F) | F - E \rangle \Rightarrow \langle (L, E) - F | F - (L, E) \rangle @ L, E$$

Proof. Both masks are subeffects of L, E . Row calculation gives

$$\begin{aligned} (L, (E - F)) - (F - E) &\equiv ((L, E) - F) - (F - (L, E)), \\ (F - E) - (L, (E - F)) &\equiv (F - (L, E)) - ((L, E) - F). \end{aligned}$$

Our goal follows from Lemma 63. $\square$

Lemma 66 (Third Modality Transformation). *For all L, E, F :*

$$\Gamma \vdash \langle L \rangle \circ \langle (L, E) - F | F - (L, E) \rangle \Rightarrow \langle E - F | F - E \rangle @ E$$

Proof. The composition result is $\langle E - F | F - E \rangle$. Our goal follows from MT-REL. $\square$

Lemma 67 (Translating Instantiated Types). *For every $\text{System } F^{1\epsilon}$ type A , we have $\llbracket A \rrbracket_F = \llbracket A[E/] \rrbracket_{F,E}$.*

Proof. Straightforward induction on A . $\square$

Theorem 18 (Type Preservation). *If $\Gamma \vdash M : A \mid \{E, \varepsilon\}$, then $\llbracket \Gamma \rrbracket_E \vdash \llbracket M \rrbracket : \llbracket A \rrbracket_E @ E$ in $\text{MET}(\mathcal{R}_{\text{scp}})$. Similarly, if $\Gamma \vdash V : A \mid \{\textcolor{red}{E}, \varepsilon\}$, then $\llbracket \Gamma \rrbracket_E \vdash \llbracket V \rrbracket : \llbracket A \rrbracket_E @ \textcolor{blue}{E}$ in $\text{MET}(\mathcal{R}_{\text{scp}})$.*

Proof. By mutual induction on the value and computation typing derivations. We prove this for each rule of the translation. As a visual aid, we repeat each rule where we replace the translation premises by the target judgement implied by the induction hypothesis and the translation in the conclusion by the target judgement we need to prove.

Case T1-UNIT. The translation is $\text{mod}_{\langle \rangle} ()$. The goal follows from T-UNIT and T-MOD.

Case

$$\begin{array}{c} \text{T1-VAR} \\ \hline \llbracket \Gamma_1, x : A, \Gamma_2 \rrbracket_E \vdash \text{mod}_{\text{top}(\llbracket A \rrbracket_E)} x : \llbracket A \rrbracket_E @ \textcolor{blue}{E} \end{array}$$

We proceed by cases on A .

Case $A = 1$. We can use the T-VAR rule since $\cdot \vdash 1 : \text{Abs}$.

Case $A = \forall.A'$. Then $\llbracket A \rrbracket_F = \llbracket \llbracket A' \rrbracket \rrbracket_F$ for all F . By rule MT-ABS, the absolute modality transforms into the required suffix, so we can use the T-VAR rule.

Case $A = A' \rightarrow^F B'$. Since the System $F^{1\epsilon}$ judgement is well-scoped, $\text{locks}(\Gamma_2)$ is the composition of the relative modalities introduced by ambient-row markers. By repeated applications of Lemma 64, composing this suffix with $\langle E - F \mid F - E \rangle$ transforms to the relative binder modality of x at the current ambient effect. The T-VAR rule therefore applies.

Case

$$\begin{array}{c} \text{T1-VALUE} \\ \llbracket \Gamma \rrbracket_E \vdash V' : \llbracket A \rrbracket_E @ \textcolor{blue}{E} \\ \hline \llbracket \Gamma \rrbracket_E \vdash V' : \llbracket A \rrbracket_E @ \textcolor{blue}{E} \end{array}$$

Simply by IH.

Case

$$\begin{array}{c} \text{T1-LET} \\ \llbracket \Gamma \rrbracket_E \vdash M' : \llbracket A \rrbracket_E @ \textcolor{blue}{E} \quad \llbracket \Gamma, x : A \rrbracket_E \vdash N' : \llbracket B \rrbracket_E @ \textcolor{blue}{E} \quad \llbracket A \rrbracket_E = \mu A' \\ \hline \llbracket \Gamma \rrbracket_E \vdash (\lambda x_0^{\llbracket A \rrbracket_E} . \text{let mod}_{\mu} x = x_0 \text{ in } N') M' : \llbracket B \rrbracket_E @ \textcolor{blue}{E} \end{array}$$

Our goal follows from the two IHs, T-LETMOD, T-ABS, and T-APP.

Case

$$\begin{array}{c}
 \text{T1-APP} \\
 \frac{\begin{array}{c} \llbracket \Gamma \rrbracket_E \vdash M' : \llbracket A \rightarrow^E B \rrbracket_E @ E \\ \llbracket \Gamma \rrbracket_E \vdash N' : \llbracket A \rrbracket_E @ E \quad x \text{ fresh} \end{array}}{\llbracket \Gamma \rrbracket_E \vdash \text{let mod}_{\langle \rangle} x = M' \text{ in } x N' : \llbracket B \rrbracket_E @ E}
 \end{array}$$

We have $\llbracket A \rightarrow^E B \rrbracket_E = \langle \rangle (\llbracket A \rrbracket_E \rightarrow \llbracket B \rrbracket_E)$. The claim follows by the T-LETMOD and T-APP rules.

Case

$$\begin{array}{c}
 \text{T1-ABS} \\
 \frac{\llbracket \Gamma, \blacklozenge_{E'} x : A \rrbracket_F \vdash M' : \llbracket B \rrbracket_F @ F \quad \nu := \langle E - F \mid F - E \rangle \quad \mu := \text{top}(\llbracket A \rrbracket_F)}{\llbracket \Gamma \rrbracket_E \vdash \text{mod}_{\nu} (\lambda x^{\llbracket A \rrbracket_F}. \text{let mod}_{\mu} x = x \text{ in } M') : \llbracket A \rightarrow^F B \rrbracket_E @ E}
 \end{array}$$

We have $\llbracket \Gamma, \blacklozenge_{E'} x : A \rrbracket_F = \llbracket \Gamma \rrbracket_E, \blacksquare_{\langle E-F \mid F-E \rangle}, x :_{\mu} A'$ where $\mu A' = \llbracket A \rrbracket_F$. Further, $\llbracket A \rightarrow^F B \rrbracket_E = \langle E - F \mid F - E \rangle (\llbracket A \rrbracket_F \rightarrow \llbracket B \rrbracket_F)$. The claim follows from the T-LETMOD, T-ABS, and T-MOD rules.

Case

$$\begin{array}{c}
 \text{T1-EABS} \\
 \frac{\llbracket \Gamma, \Lambda \varepsilon', \blacklozenge_E \rrbracket. \vdash V' : \llbracket A \rrbracket. @ \cdot}{\llbracket \Gamma \rrbracket_E \vdash \text{mod}_{\square} V' : \llbracket \forall. A \rrbracket_E @ E}
 \end{array}$$

We have $\llbracket \Gamma, \Lambda \varepsilon', \blacklozenge_E \rrbracket. = \llbracket \Gamma \rrbracket_E, \blacksquare_{\square}$. Further, $\llbracket \forall. A \rrbracket_E = \square \llbracket A \rrbracket.$. The claim follows from the T-MOD rule.

Case

$$\begin{array}{c}
 \text{T1-EAPP} \\
 \frac{\llbracket \Gamma \rrbracket_E \vdash V' : \llbracket \forall. A \rrbracket_E @ E \quad x \text{ fresh}}{\llbracket \Gamma \rrbracket_E \vdash \text{let mod}_{\square} x = V' \text{ in inst}(x; A; F; E) : \llbracket A[F/] \rrbracket_E @ E}
 \end{array}$$

We have $\llbracket \forall. A \rrbracket_E = \square \llbracket A \rrbracket.$. By T-LETMOD, it remains to show:

$$\llbracket \Gamma \rrbracket_E, x :_{\square} \llbracket A \rrbracket. \vdash \text{inst}(x; A; F; E) : \llbracket A[F/] \rrbracket_E @ E$$

We proceed by case analysis on A .

Case $A = 1$. Both $\llbracket A \rrbracket.$ and $\llbracket A[F/] \rrbracket_E$ are $\langle \rangle 1$, which has kind Abs. The goal follows from T-VAR.

Case $A = \forall. A'$. Substitution stops at effect abstraction, so both $\llbracket A \rrbracket.$ and $\llbracket A[F/] \rrbracket_E$ are $\square \llbracket A' \rrbracket.$, which has kind Abs. The goal follows from T-VAR.

Case $A = A' \rightarrow^{E'} B'$. Let

$$\xi = \langle E - F \mid F - E \rangle, \quad \mu = \langle E' \rangle, \quad \nu = \langle E - (E', F) \mid (E', F) - E \rangle,$$

and let $A_1 = \llbracket A' \rrbracket_{E'} \rightarrow \llbracket B' \rrbracket_{E'}$. By Lemma 67 applied to A' and B' , we have

$$\llbracket A \rrbracket. = \mu A_1 \quad \text{and} \quad \llbracket A[F/] \rrbracket_E = \nu A_1.$$

By MT-ABS, $[] \Rightarrow \xi$ at E because $\cdot \leq \xi(E) = F$. Thus x can be used under the ξ lock. By Lemma 64, $\xi \circ \mu \Rightarrow \nu$ at E . Thus $y :_{\xi \circ \mu} A_1$ can be used under the ν lock. The goal follows from T-VAR, T-LETMOD, and T-MOD.

Case

$$\begin{array}{c} \text{T1-Do} \\ \ell : A \rightarrow B \in \Sigma \\ \frac{\llbracket \Gamma \rrbracket_{\ell, E} \vdash M' : \llbracket A \rrbracket_{\ell, E} @ \ell, E}{\llbracket \Gamma \rrbracket_{\ell, E} \vdash \text{do } \ell \ M' : \llbracket B \rrbracket_{\ell, E} @ \ell, E} \end{array}$$

Because we only allow pure values in the operation arrows of System $F^{1\epsilon}$, we have that $\llbracket A \rrbracket_{\ell, E} = \llbracket A \rrbracket.$ and $\llbracket B \rrbracket_{\ell, E} = \llbracket B \rrbracket.$, where $\ell : \llbracket A \rrbracket. \rightarrow \llbracket B \rrbracket.$ in $\text{MET}(\mathcal{R}_{\text{scp}})$. The claim follows directly by the T-Do rule.

Case

$$\begin{array}{c} \text{T1-HANDLER} \\ \Sigma \ni \ell : A' \rightarrow B' \quad \llbracket \Gamma, \blacklozenge_F, p : A', r : B' \rightarrow^E A \rrbracket_E \vdash \llbracket N \rrbracket : \llbracket A \rrbracket_E @ E \text{ (1)} \\ \nu := \langle F - E \mid E - F \rangle \quad \llbracket A \rrbracket_{\ell, E} = \mu A_0 \quad \llbracket A \rrbracket_E = \mu' A_0 \\ \llbracket A' \rrbracket_E = \llbracket A' \rrbracket. = \mu'' A'_0 \quad \llbracket B' \rrbracket_E = \llbracket B' \rrbracket. \quad T := \langle \ell \rangle (\llbracket 1 \rrbracket_{\ell, E} \rightarrow \llbracket A \rrbracket_{\ell, E}) \\ N_0 := \text{let}_{\langle \ell \rangle} \text{mod}_{\mu} y = x \text{ in } \text{mod}_{\mu'} y \quad N_1 := \text{let } \text{mod}_{\mu''} p' = p \text{ in } (\llbracket N \rrbracket)[p'/p] \\ \frac{\llbracket \Gamma \rrbracket_F \vdash \text{mod}_{\nu} (\lambda f^T. \text{let } \text{mod}_{\langle \ell \rangle} f' = f \text{ in } \text{handle } f' (\text{mod}_{\langle \rangle} ())) \text{ with } @ F}{\{\text{return } x \mapsto N_0, \ell \ p \ r \mapsto N_1\} : \llbracket (1 \rightarrow^{\ell, E} A) \rightarrow^E A \rrbracket_F} \end{array}$$

Expanding the translated context and alpha-renaming p to p' in (1) gives

$$\llbracket \Gamma \rrbracket_F, \blacklozenge_{\nu}, p' :_{\mu''} A'_0, r : \llbracket B' \rrbracket_E \rightarrow \llbracket A \rrbracket_E \vdash (\llbracket N \rrbracket)[p'/p] : \llbracket A \rrbracket_E @ E \text{ (2)}$$

Let

$$\Gamma' := \llbracket \Gamma \rrbracket_F, \blacklozenge_{\nu}, f : T, f' :_{\langle \ell \rangle} \llbracket 1 \rrbracket_{\ell, E} \rightarrow \llbracket A \rrbracket_{\ell, E}$$

By (2), context equivalence, variable weakening, variable swapping, and T-LETMOD, we obtain the operation-clause premise

$$\Gamma', p : \llbracket A' \rrbracket., r : \llbracket B' \rrbracket. \rightarrow \llbracket A \rrbracket_E \vdash N_1 : \llbracket A \rrbracket_E @ E \text{ (3)}$$

For the return clause, the handler binds $x :_{\langle \ell \rangle} \mu A_0$. By a case analysis on A with Lemma 66, rules T-VAR, T-LETMOD, and T-MOD give

$$\Gamma', x :_{\langle \ell \rangle} \mu A_0 \vdash N_0 : \llbracket A \rrbracket_E @ E \text{ (4)}$$

By T-VAR, T-UNIT, T-MOD, and T-APP, we have the handled-computation premise

$$\Gamma', \mathbf{A}_{\langle \ell \rangle} \vdash f'(\text{mod}_{\langle \rangle} ()) : \llbracket A \rrbracket_{\ell, E} @ \ell, E \quad (5)$$

By (3), (4), (5), and T-HANDLE, we have

$$\Gamma' \vdash \text{handle } f'(\text{mod}_{\langle \rangle} ()) \text{ with } \{\text{return } x \mapsto N_0, \ell \ p \ r \mapsto N_1\} : \llbracket A \rrbracket_E @ E$$

Finally, T-LETMOD, T-ABS, and T-MOD give

$$\llbracket \Gamma \rrbracket_F \vdash \llbracket \text{handler } \{\ell \ p \ r \mapsto N\} \rrbracket : \llbracket (1 \rightarrow^{\ell, E} A) \rightarrow^E A \rrbracket_F @ F$$

□

D Proofs of algorithms

D.1 Lemmas

It is obvious that our declarative judgements are preserved by information increase. For instance, for the typing judgement, if $\Theta_0 \vdash_m M : A$ and $\Theta_0 \sqsubseteq \Theta_1$, then $\Theta_1 \vdash_m M : A$. In the proofs, we refer to this property as *stability*.

Lemma 68 (Suffix moving). *If $\Theta_1, \Theta_2, \Xi \vdash_m M : A$ where Ξ does not depend on entries in Θ_2 , then $\Theta_1, \Xi, \Theta_2 \vdash_m M : A$.*

Proof. Straightforward induction on the typing derivation. $\square$

Lemma 69 (Suffix erasure). *If $\Theta, \Xi \vdash_m M : A$ and Ξ is fresh for A , then $\Theta \vdash_m M : A$.*

Proof. Straightforward induction on the typing derivation. $\square$

Lemma 70 (Splitting refines). *If $\Theta_0 \vdash P_0 \triangleright P_1 \dashv \Theta_1$, then $\Theta_1 \vdash P_0 \blacktriangleright P_1$.*

Proof. Straightforward induction on the splitting derivation. $\square$

Lemma 71 (Precision extends skeleton subtyping). *If $\Theta \vdash P' \leq P$ and $\Theta \vdash_m^{\text{sk}} P \leq Q$, then $\Theta \vdash_m^{\text{sk}} P' \leq Q$.*

Proof. Straightforward induction on the subtyping derivation. $\square$

Lemma 72 (Precision preserves refinement). *If $\Theta \vdash P \leq Q$ and $\Theta \vdash Q \blacktriangleright Q'$, then $\Theta \vdash P \blacktriangleright Q'$.*

Proof. Straightforward induction on the refinement derivation. $\square$

Definition 13 (Context precision). *We write $\Theta' \leq_{\text{ctx}} \Theta$ if Θ' has the same non-variable entries as Θ , and every term-variable entry in Θ' is more precise than the corresponding entry in Θ :*

$$\frac{}{\cdot \leq_{\text{ctx}} \cdot} \quad \frac{e \neq x : P \quad \Theta' \leq_{\text{ctx}} \Theta}{\Theta', e \leq_{\text{ctx}} \Theta, e} \quad \frac{\Theta' \leq_{\text{ctx}} \Theta \quad \Theta' \vdash P' \leq P}{\Theta', x : P' \leq_{\text{ctx}} \Theta, x : P}$$

Lemma 73 (Context precision preserves skeleton inference). *If $\Theta' \leq_{\text{ctx}} \Theta$ and $\Theta \vdash_m M \Rightarrow Q$, then $\Theta' \vdash_m M \Rightarrow Q$.*

Proof. By induction on $\Theta \vdash_m M \Rightarrow Q$. Refinement and subtyping judgements that do not involve a term-variable lookup are preserved because the contexts have the same non-variable entries. The only interesting case is a variable. If $\Theta \ni x : P_x$ and $\Theta' \ni x : P'_x$ with $\Theta' \vdash P'_x \leq P_x$, we apply Lemma 71 to derive the needed subtyping judgement. $\square$

D.2 Type inference

In this section, we prove Theorems 21 and 22 for the type inference algorithm in Section 9.2. Since we have proved the soundness and completeness of the syntax-directed type system in Section 8.4, we directly use the syntax-directed type system for proofs. To avoid repeating the subtyping premises in SD-UNIT, SD-VAR, and SD-ANNO, we use an equivalent presentation in this section. We write $O ::= () \mid x \mid M : A$ for base terms and use the following four rules:

$$\begin{array}{c}
 \text{SD-UNIT} \\
 \hline
 \Gamma \vdash_I () : 1
 \end{array}
 \quad
 \begin{array}{c}
 \text{SD-VAR} \\
 \Gamma \ni x : A \\
 \hline
 \Gamma \vdash_I x : A
 \end{array}
 \quad
 \begin{array}{c}
 \text{SD-ANNO} \\
 \Gamma \vdash_C M : A \\
 \hline
 \Gamma \vdash_I (M : A) : A
 \end{array}
 \quad
 \begin{array}{c}
 \text{SD-SUB} \\
 \Gamma \vdash_I O : A \quad \Gamma \vdash_m A \leq B \\
 \hline
 \Gamma \vdash_m O : B
 \end{array}$$

This presentation is local to the proofs.

Theorem 21 (Soundness of type inference). *Given a type inference problem $(\Theta_0; m; P; M)$, if $\Theta_0 [P]_m M \Rightarrow A \vdash \Theta_1$, then $(\Theta_0 \sqsubseteq \Theta_1; A)$ is a solution.*

Proof. Our goal is to show that $\Theta_1 \vdash_m M : A$ where $\Theta_1 \vdash P \blacktriangleright A$ and $\Theta_0 \sqsubseteq \Theta_1$ is a valid metasubstitution. By induction on the derivation of $\Theta_0 [P]_m M \Rightarrow A \vdash \Theta_1$.

Case

$$\begin{array}{c}
 \text{I-UNIT} \\
 \Theta_0 [P]_m^{\text{ty}} 1 \leq A \vdash \Theta_1 \text{ (1)} \\
 \hline
 \Theta_0 [P]_m () \Rightarrow A \vdash \Theta_1
 \end{array}$$

Theorem 27 on (1) gives that $(\Theta_0 \sqsubseteq \Theta_1; A)$ is a solution to the subtyping problem $(\Theta_0; \text{ty}; m; P; 1)$, which gives $\Theta_1 \vdash_m^{\text{ty}} 1 \leq A$ (2) where $\Theta_1 \vdash P \blacktriangleright A$ (3). By (2), SD-UNIT, and SD-SUB we have $\Theta_1 \vdash_m () : A$ (4). Our goal follows from (3) and (4).

Case

$$\begin{array}{c}
 \text{I-VAR} \\
 \Theta_0 \ni x : A \quad \Theta_0 [P]_m^{\text{ty}} A \leq B \vdash \Theta_1 \text{ (1)} \\
 \hline
 \Theta_0 [P]_m x \Rightarrow B \vdash \Theta_1
 \end{array}$$

Theorem 27 on ① gives that $(\Theta_0 \sqsubseteq \Theta_1; B)$ is a solution to the subtyping problem $(\Theta_0; \text{ty}; m; P; A)$, which gives $\Theta_1 \vdash_m^{\text{ty}} A \leq B$ ② where $\Theta_1 \vdash P \blacktriangleright B$ ③. By $\Theta_0 \ni x : A$ and $\Theta_0 \sqsubseteq \Theta_1$, we have $\Theta_1 \ni x : A$. Then by $\Theta_1 \ni x : A$, ②, SD-VAR, and SD-SUB we have $\Theta_1 \vdash_m x : B$ ④. Our goal follows from ③ and ④.

Case

$$\text{I-ANNO} \quad \frac{\Theta_0 [A] \vdash_c M \Rightarrow A \dashv \Theta_1 \text{ ①} \quad \Theta_1 [P] \vdash_m^{\text{ty}} A \leq B \dashv \Theta_2 \text{ ②}}{\Theta_0 [P] \vdash_m (M : A) \Rightarrow B \dashv \Theta_2}$$

IH on ① gives that $(\Theta_0 \sqsubseteq \Theta_1; A)$ is a solution to the type inference problem $(\Theta_0; C; A; M)$, which gives $\Theta_1 \vdash_c M : A$. Then by SD-ANNO we have $\Theta_1 \vdash_I (M : A) : A$ ③. Theorem 27 on ② gives that $(\Theta_1 \sqsubseteq \Theta_2; B)$ is a solution to the subtyping problem $(\Theta_1; \text{ty}; m; P; A)$, which further gives $\Theta_2 \vdash_m^{\text{ty}} A \leq B$ ④ where $\Theta_2 \vdash P \blacktriangleright B$ ⑤. By transitivity of information increase we have $\Theta_0 \sqsubseteq \Theta_2$ ⑥. By stability on ③, ④, and SD-SUB we have $\Theta_2 \vdash_m (M : A) : B$ ⑦. Our goal follows from ⑤, ⑥, and ⑦.

Case

$$\text{I-FREEZE} \quad \frac{\Theta_0 [\mathbb{A}] \vdash_I M \Rightarrow B \dashv \Theta_1 \text{ ①} \quad \Theta_1 \vdash B \sim A \dashv \Theta_2 \text{ ②}}{\Theta_0 [A] \vdash_c [M] \Rightarrow A \dashv \Theta_2}$$

IH on ① gives that $(\Theta_0 \sqsubseteq \Theta_1; B)$ is a solution to the type inference problem $(\Theta_0; I; \mathbb{A}; M)$, which further gives $\Theta_1 \vdash_I M : B$ ③. Lemma 25 on ② gives that $\Theta_1 \sqsubseteq \Theta_2$ is a solution to the consistency problem $(\Theta_1; B; A)$, which further gives $\Theta_2 \vdash B \equiv A$ ④. By stability on ③, we have $\Theta_2 \vdash_I M : B$ ⑤. By ⑤, ④, and SD-FREEZE we have $\Theta_2 \vdash_c [M] : A$ ⑥. By transitivity of information increase we have $\Theta_0 \sqsubseteq \Theta_2$ ⑦. Our goal follows from ⑥, ⑦, and $\Theta_2 \vdash A \blacktriangleright A$.

Case

$$\text{I-ABSCHECK} \quad \frac{\Theta_0, \bar{\alpha} \vdash A_0 \blacktriangleright A \rightarrow B \dashv \Theta_1, \bar{\alpha} \text{ ⑦} \quad \Theta_1, \bar{\alpha}, x : A [B] \vdash_c M \Rightarrow B \dashv \Theta_2, \bar{\alpha}, x : A, \Xi \text{ ①}}{\Theta_0 [\forall \bar{\alpha}. A_0] \vdash_c \lambda x. M \Rightarrow \forall \bar{\alpha}. A \rightarrow B \dashv \Theta_2}$$

IH on ① gives that $(\Theta_1, \bar{\alpha}, x : A \sqsubseteq \Theta_2, \bar{\alpha}, x : A, \Xi \text{ ⑥}; B)$ is a solution to the type inference problem $(\Theta_1, \bar{\alpha}, x : A; C; B; M)$, which further gives

$$\Theta_2, \bar{\alpha}, x : A, \Xi \vdash_c M : B \text{ ②}$$

By Lemma 69, since all variables in Ξ do not appear in B , we have

$$\Theta_2, \bar{\alpha}, x : A \vdash_c M : B \text{ ③}$$

By ③ and SD-ABSCHECK, we have

$$\Theta_2 \vdash_c \lambda x.M : \forall \bar{\alpha}. A \rightarrow B \text{ ④}$$

By ⑥ and definition of metasubstitution, we have $\Theta_1 \sqsubseteq \Theta_2$. By ⑦, Lemma 70, and stability, we have $\Theta_2, \bar{\alpha} \vdash A_0 \blacktriangleright A \rightarrow B$. Thus $\Theta_2 \vdash \forall \bar{\alpha}. A_0 \blacktriangleright \forall \bar{\alpha}. A \rightarrow B$ ⑤. By ⑦, ⑥, definition of metasubstitution, and transitivity of information increase, we have $\Theta_0 \sqsubseteq \Theta_2$ ⑩. Our goal follows from ④, ⑩, and ⑤.

Case

$$\begin{array}{c} \text{I-ABSANNOCHECK} \\ \Theta_0, \bar{\alpha} \vdash A_0 \blacktriangleright A' \rightarrow B \vdash \Theta_1, \bar{\alpha} \text{ ①} \quad \Theta_1, \bar{\alpha} \vdash A' \sim A \vdash \Theta_2, \bar{\alpha} \text{ ②} \\ \Theta_2, \bar{\alpha}, x : A [B] \vdash_c M \Rightarrow B \vdash \Theta_3, \bar{\alpha}, x : A, \Xi \text{ ③} \\ \hline \Theta_0 [\forall \bar{\alpha}. A_0] \vdash_c \lambda x^A.M \Rightarrow \forall \bar{\alpha}. A \rightarrow B \vdash \Theta_3 \end{array}$$

Lemma 25 on ② gives that

$$\Theta_1, \bar{\alpha} \sqsubseteq \Theta_2, \bar{\alpha} \text{ ④}$$

is a solution to the consistency problem $(\Theta_1, \bar{\alpha}; A'; A)$, which further gives $\Theta_2, \bar{\alpha} \vdash A' \equiv A$ ⑤. IH on ③ gives that

$$(\Theta_2, \bar{\alpha}, x : A \sqsubseteq \Theta_3, \bar{\alpha}, x : A, \Xi \text{ ⑥}; B)$$

is a solution to the type inference problem $(\Theta_2, \bar{\alpha}, x : A; C; B; M)$, which further gives

$$\Theta_3, \bar{\alpha}, x : A, \Xi \vdash_c M : B \text{ ⑦}$$

By Lemma 69, since all variables in Ξ do not appear in B , we have

$$\Theta_3, \bar{\alpha}, x : A \vdash_c M : B \text{ ⑧}$$

By ⑧ and $\Theta_3, \bar{\alpha} \vdash A \equiv A$ and SD-ABSANNOCHECK, we have

$$\Theta_3 \vdash_c \lambda x^A.M : \forall \bar{\alpha}. A \rightarrow B \text{ ⑨}$$

By ①, Lemma 70, and stability, we have $\Theta_3, \bar{\alpha} \vdash A_0 \blacktriangleright A' \rightarrow B$. By ⑤ and stability, we have $\Theta_3, \bar{\alpha} \vdash A' \equiv A$. Thus $\Theta_3 \vdash \forall \bar{\alpha}. A_0 \blacktriangleright \forall \bar{\alpha}. A \rightarrow B$ ⑩. By ①, ④, ⑥, definition of metasubstitution, and transitivity of information increase, we have $\Theta_0 \sqsubseteq \Theta_3$ ⑪. Our goal follows from ⑨, ⑪, and ⑩.

Case

$$\begin{array}{c} \text{I-Abs} \\ m \neq c \quad \Theta_0 \vdash P_0 \blacktriangleright P \rightarrow Q \vdash \Theta_1 \text{ ⑦} \\ \text{guess}(P) = (\bar{\hat{\alpha}}; A) \text{ ④} \quad \Theta_1, \bar{\hat{\alpha}}, x : A [Q] \vdash_{m-} M \Rightarrow B \vdash \Theta_2, x : A, \Xi \text{ ⑧} \\ \hline \Theta_0 [P_0] \vdash_m \lambda x.M \Rightarrow A \rightarrow B \vdash \Theta_2, \Xi \end{array}$$

IH on ① gives that $(\Theta_1, \bar{\alpha}, x : A \sqsubseteq \Theta_2, x : A, \Xi \text{ ⑧}; B)$ is a solution to the type inference problem $(\Theta_1, \bar{\alpha}, x : A; m-; Q; M)$, which further gives

$$\Theta_2, x : A, \Xi \vdash_m M : B \text{ ②}$$

$$\Theta_2, x : A, \Xi \vdash Q \blacktriangleright B \text{ ⑨}$$

By Lemma 68 on ② and SD-Abs, we have

$$\Theta_2, \Xi \vdash_m \lambda x. M : A \rightarrow B \text{ ③}$$

By ④ and definition of $\text{guess}(P)$, we have $\Theta_1, \bar{\alpha} \vdash P \blacktriangleright A$. Then by ⑧, definition of metasubstitution, and stability, we have $\Theta_2, \Xi \vdash P \blacktriangleright A$. Since refinement judgements are stable under dropping term variables, ⑨ gives $\Theta_2, \Xi \vdash Q \blacktriangleright B$. Therefore, we have $\Theta_2, \Xi \vdash P \rightarrow Q \blacktriangleright A \rightarrow B$. By Lemma 70 on ⑦ and stability, we have $\Theta_2, \Xi \vdash P_0 \blacktriangleright P \rightarrow Q$. Then by transitivity of refinement we further have $\Theta_2, \Xi \vdash P_0 \blacktriangleright A \rightarrow B$ ⑤. By ⑦, ⑧, definition of metasubstitution, and transitivity of information increase, we have $\Theta_0 \sqsubseteq \Theta_2, \Xi$ ⑥. Our goal follows from ③, ⑤, and ⑥.

Case

I-ABSANNO

$$\frac{\begin{array}{c} m \neq c \quad \Theta_0 \vdash P_0 \blacktriangleright P \rightarrow Q \vdash \Theta_1 \text{ ⑦} \\ \Theta_1 \vdash A \sim P \vdash \Theta_2 \text{ ④} \quad \Theta_2, x : A [Q]_{m-} M \Rightarrow B \vdash \Theta_3, x : A, \Xi \text{ ①} \end{array}}{\Theta_0 [P_0]_m \lambda x^A. M \Rightarrow A \rightarrow B \vdash \Theta_3, \Xi}$$

IH on ① gives that $(\Theta_2, x : A \sqsubseteq \Theta_3, x : A, \Xi \text{ ⑧}; B)$ is a solution to the type inference problem $(\Theta_2, x : A; m-; Q; M)$, which further gives

$$\Theta_3, x : A, \Xi \vdash_m M : B \text{ ②}$$

$$\Theta_3, x : A, \Xi \vdash Q \blacktriangleright B \text{ ③}$$

By Lemma 25 on ④, we have $\Theta_1 \sqsubseteq \Theta_2$ and $\Theta_2 \vdash P \blacktriangleright A$. Then by ⑧, definition of metasubstitution, and stability, we have $\Theta_3, \Xi \vdash P \blacktriangleright A$ ⑤. By $\Theta_3, \Xi \vdash A \equiv A$ and Lemma 68 on ② and SD-ABSANNO, we have

$$\Theta_3, \Xi \vdash_m \lambda x^A. M : A \rightarrow B \text{ ⑨}$$

Since refinement judgements are stable under dropping term variables, ③ gives $\Theta_3, \Xi \vdash Q \blacktriangleright B$. Therefore, we have $\Theta_3, \Xi \vdash P \rightarrow Q \blacktriangleright A \rightarrow B$ ⑥. By Lemma 70 on ⑦ and stability, we have $\Theta_3, \Xi \vdash P_0 \blacktriangleright P \rightarrow Q$. Then by transitivity of refinement we have $\Theta_3, \Xi \vdash P_0 \blacktriangleright A \rightarrow B$. By ⑧ and definition of metasubstitution, we have $\Theta_2 \sqsubseteq \Theta_3, \Xi$. ⑦ gives $\Theta_0 \sqsubseteq \Theta_1$. Then by $\Theta_1 \sqsubseteq \Theta_2$, and $\Theta_2 \sqsubseteq \Theta_3, \Xi$, and transitivity of metasubstitution, we have $\Theta_0 \sqsubseteq \Theta_3, \Xi$. Our goal follows from ⑨, $\Theta_3, \Xi \vdash P_0 \blacktriangleright A \rightarrow B$, and $\Theta_0 \sqsubseteq \Theta_3, \Xi$.

Case

I-APP

$$\frac{\Theta_0 [\mathbb{A}] \vdash_c N \Rightarrow P \dashv \Theta_1 \text{ (1)} \quad \Theta_1 [P \rightarrow Q] \vdash_{Fm} M \Rightarrow A \rightarrow B \dashv \Theta_2 \text{ (2)} \quad \Theta_2 [A] \vdash_c N \Rightarrow A' \dashv \Theta_3 \text{ (3)}}{\Theta_0 [Q] \vdash_m M N \Rightarrow B \dashv \Theta_3}$$

Theorem 23 on (1) gives that $(\Theta_0 \sqsubseteq \Theta_1; P)$ is a solution to the skeleton inference problem $(\Theta_0; c; \mathbb{A}; N)$, which further gives

$$\Theta_1 \vdash_c N \Rightarrow P \text{ (4)}$$

IH on (2) gives that $(\Theta_1 \sqsubseteq \Theta_2; A \rightarrow B)$ is a solution to the type inference problem $(\Theta_1; Fm; P \rightarrow Q; M)$, which further gives

$$\Theta_2 \vdash_{Fm} M : A \rightarrow B \text{ (5)}$$

$$\Theta_2 \vdash P \rightarrow Q \blacktriangleright A \rightarrow B \text{ (6)}$$

By definition of refinement, (6) gives $\Theta_2 \vdash P \blacktriangleright A$ (7) and $\Theta_2 \vdash Q \blacktriangleright B$ (10).

IH on (3) gives that $(\Theta_2 \sqsubseteq \Theta_3; A')$ is a solution to the type inference problem $(\Theta_2; c; A; N)$, which further gives

$$\Theta_3 \vdash_c N : A' \text{ (8)}$$

$$\Theta_3 \vdash A \blacktriangleright A' \text{ (13)}$$

(1) gives $\Theta_0 \sqsubseteq \Theta_1$. (2) gives $\Theta_1 \sqsubseteq \Theta_2$. (3) gives $\Theta_2 \sqsubseteq \Theta_3$. By transitivity of metasubstitution, we have $\Theta_0 \sqsubseteq \Theta_3$ (12). By stability, (4), (7), and (5), we have

$$\Theta_3 \vdash_c N \Rightarrow P \quad \Theta_3 \vdash P \blacktriangleright A \quad \Theta_3 \vdash_{Fm} M : A \rightarrow B$$

By (13), we have $\Theta_3 \vdash A \equiv A'$. Thus by (8) and (13), we have $\Theta_3 \vdash_c N : A$. Then by SD-APP, we have

$$\Theta_3 \vdash_m M N : B \text{ (9)}.$$

By stability on (10), we have $\Theta_3 \vdash Q \blacktriangleright B$ (11). Our goal follows from (9), (11), and (12).

□

Theorem 22 (Completeness and optimality of type inference). *Given a type inference problem $(\Theta_0; m; P; M)$, if there exists a solution $(\theta :: \Theta_0 \sqsubseteq \Theta; B)$, then $\Theta_0 [P] \vdash_m M \Rightarrow A \dashv \Theta_1$ gives its optimal solution $(\Theta_0 \sqsubseteq \Theta_1; A)$.*

Proof. It is enough to prove that if $(\theta :: \Theta_0 \sqsubseteq \Theta; B)$ is a solution to the type inference problem $(\Theta_0; m; P; M)$, then we have $\Theta_0 [P] \vdash_m M \Rightarrow A \dashv \Theta_1$ and $\theta_1 :: \Theta_1 \sqsubseteq \Theta$ and $\Theta \vdash \theta_1 A \equiv B$ and $\theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta$. By induction on the derivation of typing judgement $\Theta \vdash_m M : B$ where $\Theta \vdash \theta P \blacktriangleright B$ and $\theta :: \Theta_0 \sqsubseteq \Theta$.

Case

SD-UNIT

$$\frac{}{\Theta \vdash_{\mathbf{I}} () : 1}$$

In this case $m = \mathbf{I}$ and hence $P = \hat{\mathbb{A}}$. By SI-INFER and I-UNIT we have $\Theta_0 [\hat{\mathbb{A}}]_{\mathbf{I}} () \Rightarrow 1 \dashv \Theta_0$. Our goal follows from $\theta :: \Theta_0 \sqsubseteq \Theta$ and $\Theta \vdash \theta 1 \equiv 1$.

Case

SD-VAR

$$\frac{\Theta \ni x : A}{\Theta \vdash_{\mathbf{I}} x : A}$$

In this case $m = \mathbf{I}$ and hence $P = \hat{\mathbb{A}}$. By $\theta :: \Theta_0 \sqsubseteq \Theta$ and $\Theta \ni x : A$, we have $\Theta_0 \ni x : A'$ and $\Theta \vdash \theta A' \equiv A$. By SI-INFER and I-VAR we have $\Theta_0 [\hat{\mathbb{A}}]_{\mathbf{I}} x \Rightarrow A' \dashv \Theta_0$ where $\Theta_0 \ni x : A'$. Our goal follows from $\theta :: \Theta_0 \sqsubseteq \Theta$ and $\Theta \vdash \theta A' \equiv A$.

Case

SD-ANNO

$$\frac{\Theta \vdash_{\mathbf{C}} M : A \text{ ①}}{\Theta \vdash_{\mathbf{I}} (M : A) : A}$$

In this case $m = \mathbf{I}$ and hence $P = \hat{\mathbb{A}}$. Since annotations A contain no flexible variables, $(\theta :: \Theta_0 \sqsubseteq \Theta; A)$ is a solution to the type inference problem $(\Theta_0; \mathbf{C}; A; M)$. IH on ① gives

$$\Theta_0 [A]_{\mathbf{C}} M \Rightarrow A \dashv \Theta_1 \text{ ②}$$

$$\theta_1 :: \Theta_1 \sqsubseteq \Theta \text{ ③}$$

$$\Theta \vdash \theta_1 A \equiv A \text{ ④}$$

$$\theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \text{ ⑤}$$

By SI-INFER, we have $\Theta_1 [\hat{\mathbb{A}}]_{\mathbf{I}}^{\text{ty}} A \leq A \dashv \Theta_1 \text{ ⑥}$. By ②, ⑥, and I-ANNO, we have

$$\Theta_0 [\hat{\mathbb{A}}]_{\mathbf{I}} (M : A) \Rightarrow A \dashv \Theta_1 \text{ ⑦}.$$

Our goal follows from ③, ④, ⑤, and ⑦.

Case

SD-SUB

$$\frac{\Theta \vdash_{\mathbf{I}} O : A \text{ ①} \quad \Theta \vdash_{\mathbf{m}} A \leq B \text{ ②}}{\Theta \vdash_{\mathbf{m}} O : B}$$

IH on ① gives

$$\Theta_0 [\hat{\mathbb{A}}]_{\mathbf{I}} O \Rightarrow A' \dashv \Theta_1 \text{ ③}$$

$$\theta_1 :: \Theta_1 \sqsubseteq \Theta \text{ ④}$$

$$\Theta \vdash \theta_1 A' \equiv A \text{ ⑤}$$

$$\theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \text{ ⑥}$$

By ② and ⑤, we have $\Theta \vdash_m \theta_1 A' \leq B$ ⑦. By ⑥ and definition of metasubstitution, the original refinement $\Theta \vdash \theta P \blacktriangleright B$ gives $\Theta \vdash \theta_1 P \blacktriangleright B$ ⑭. By ⑦ and ⑭, we have that $(\theta_1 :: \Theta_1 \sqsubseteq \Theta; B)$ is a solution to the subtyping problem $(\Theta_1; \text{ty}; m; P; A')$. Then by Theorem 28, and since precision on types is equivalence, we have

$$\Theta_1 [P]_{\text{m}}^{\text{ty}} A' \leq B' \dashv \Theta_2 \text{ ⑧}$$

$$\theta_2 :: \Theta_2 \sqsubseteq \Theta \text{ ⑮}$$

$$\Theta \vdash \theta_2 B' \equiv B \text{ ⑩}$$

$$\theta_1 \equiv \theta_2 :: \Theta_1 \sqsubseteq \Theta \text{ ⑪}$$

By ⑥, ⑪, and $\Theta_0 \sqsubseteq \Theta_1$, we have $\theta \equiv \theta_2 :: \Theta_0 \sqsubseteq \Theta$ ⑬.

Case analysis on the shape of O (reusing the same meta-numbers in different case branches).

Subcase $O = ()$. Inversion on ③ gives $\Theta_1 = \Theta_0$ and $A' = 1$, so I-UNIT on ⑧ gives $\Theta_0 [P]_{\text{m}} O \Rightarrow B' \dashv \Theta_2$ ⑨.

Subcase $O = x$. Inversion on ③ gives $\Theta_0 \ni x : A'$ and $\Theta_1 = \Theta_0$, so I-VAR on ⑧ gives $\Theta_0 [P]_{\text{m}} O \Rightarrow B' \dashv \Theta_2$ ⑨.

Subcase $O = (M : A)$. Inversion on ③ gives $\Theta_0 [A]_{\text{c}} M \Rightarrow A \dashv \Theta_1$ and $A' = A$, so I-ANNO on ③ and ⑧ gives $\Theta_0 [P]_{\text{m}} O \Rightarrow B' \dashv \Theta_2$ ⑨.

Our goal follows from ⑨, ⑮, ⑩, and ⑬.

Case

$$\begin{array}{c} \text{SD-FREEZE} \\ \Theta \vdash_{\text{I}} M : B \text{ ①} \quad \Theta \vdash B \equiv A \text{ ⑨} \\ \hline \Theta \vdash_{\text{c}} [M] : A \end{array}$$

In this case $m = \text{c}$. By $P :^{\text{ty}} \text{c}$, we have $P = A_p$ for some concrete type A_p . IH on ① gives

$$\Theta_0 [\Omega]_{\text{I}} M \Rightarrow B' \dashv \Theta_1 \text{ ②}$$

$$\theta_1 :: \Theta_1 \sqsubseteq \Theta \text{ ③}$$

$$\Theta \vdash \theta_1 B' \equiv B \text{ ④}$$

$$\theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \text{ ⑥}$$

Since $(\theta; A)$ is a solution, we have $\Theta \vdash \theta A_p \blacktriangleright A$. Then by ⑥, we have $\Theta \vdash \theta_1 A_p \blacktriangleright A$ ⑬. By ④ and ⑨, we have $\Theta \vdash \theta_1 B' \equiv A$ ⑭. By ⑭ and ⑬, we

have $(\theta_1 :: \Theta_1 \sqsubseteq \Theta; A)$ is a solution to the consistency problem $(\Theta_1; A_p; B')$. Then by Lemma 26, and since precision on types is equivalence, we have

$$\begin{aligned} \Theta_1 \vdash B' &\sim A_p \dashv \Theta_2 \text{ (5)} \\ \theta_2 :: \Theta_2 &\sqsubseteq \Theta \text{ (7)} \\ \Theta \vdash \theta_2 A_p &\equiv A \text{ (8)} \\ \theta_1 &\equiv \theta_2 :: \Theta_1 \sqsubseteq \Theta \text{ (10)} \end{aligned}$$

Applying I-FREEZE to (2) and (5) gives $\Theta_0 [A_p]_c [M] \Rightarrow A_p \dashv \Theta_2$ (11). By (6), (10), and $\Theta_0 \sqsubseteq \Theta_1$, we have $\theta \equiv \theta_2 :: \Theta_0 \sqsubseteq \Theta$ (12). Our goal follows from (11), (7), (8), and (12).

Case

$$\frac{\text{SD-ABSCHECK} \quad \Theta, \bar{\alpha}, x : A \vdash_c M : B \text{ (1)}}{\Theta \vdash_c \lambda x. M : \forall \bar{\alpha}. A \rightarrow B}$$

The type inference problem is $(\Theta_0; C; A_p; \lambda x. M)$ where $A_p :^{ty} c$. The solution is $(\theta :: \Theta_0 \sqsubseteq \Theta; \forall \bar{\alpha}. A \rightarrow B)$. We have $\Theta \vdash \theta A_p \blacktriangleright \forall \bar{\alpha}. A \rightarrow B$. By this refinement, the only possible shapes of A_p are $\forall \bar{\alpha}. A' \rightarrow B'$ and $\forall \bar{\alpha}. \hat{\alpha}$. Case analysis on the shape of A_p .

Subcase $A_p = \forall \bar{\alpha}. A' \rightarrow B'$. By the solution $(\theta; \forall \bar{\alpha}. A \rightarrow B)$, we have

$$\Theta, \bar{\alpha} \vdash \theta A' \equiv A \text{ (12)} \quad \text{and} \quad \Theta, \bar{\alpha} \vdash \theta B' \equiv B \text{ (13)}.$$

Therefore $(\theta :: \Theta_0, \bar{\alpha}, x : A' \sqsubseteq \Theta, \bar{\alpha}, x : A; B)$ is a solution to the type inference problem $(\Theta_0, \bar{\alpha}, x : A'; C; B'; M)$. IH on (1) gives

$$\begin{aligned} \Theta_0, \bar{\alpha}, x : A' [B']_c M &\Rightarrow B_1 \dashv \Theta_1, \bar{\alpha}, x : A', \Xi \text{ (2)} \\ \theta_1 :: \Theta_1, \bar{\alpha}, x : A', \Xi &\sqsubseteq \Theta, \bar{\alpha}, x : A \text{ (3)} \\ \Theta, \bar{\alpha}, x : A \vdash \theta_1 B_1 &\equiv B \text{ (4)} \\ \theta \equiv \theta_1 :: \Theta_0, \bar{\alpha}, x : A' &\sqsubseteq \Theta, \bar{\alpha}, x : A \text{ (5)} \end{aligned}$$

By definition of splitting, we have

$$\Theta_0, \bar{\alpha} \vdash A' \rightarrow B' \triangleright A' \rightarrow B' \dashv \Theta_0, \bar{\alpha} \text{ (11)}.$$

By (11), (2), and I-ABSCHECK, we have

$$\Theta_0 [\forall \bar{\alpha}. A' \rightarrow B']_c \lambda x. M \Rightarrow \forall \bar{\alpha}. A' \rightarrow B_1 \dashv \Theta_1 \text{ (6)}.$$

By (3) and metasubstitution weakening, we have $\theta_1 :: \Theta_1 \sqsubseteq \Theta$ (7). By (5) and definition of metasubstitution, we have $\theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta$ (8). By (5) and (12), we have $\Theta, \bar{\alpha} \vdash \theta_1 A' \equiv A$ (15). By (15) and (4), we have $\Theta \vdash \theta_1 (\forall \bar{\alpha}. A' \rightarrow B_1) \equiv \forall \bar{\alpha}. A \rightarrow B$ (14). Our goal follows from (6), (7), (14), and (8).

Subcase $A_p = \forall \bar{\alpha}. \hat{\alpha}$. By the original solution, we have $\Theta, \bar{\alpha} \vdash \theta \hat{\alpha} \equiv A \rightarrow B$ ⑫.

Assuming fresh $\hat{\beta}$ and $\hat{\gamma}$, we have

$$\Theta_0, \bar{\alpha} \vdash \hat{\alpha} \triangleright \hat{\beta} \rightarrow \hat{\gamma} \vdash \Theta_1, \bar{\alpha} \text{ ⑬.}$$

Since $\theta \hat{\alpha}$ is equivalent to $A \rightarrow B$, and by definition of splitting $\hat{\beta}$ and $\hat{\gamma}$ are introduced immediately before the assignment for $\hat{\alpha}$, the components A and B are well-scoped at the positions of $\hat{\beta}$ and $\hat{\gamma}$ in Θ_1 . By ⑫ and ⑬, we have a metasubstitution θ' extending θ with $\hat{\beta} \mapsto A$ and $\hat{\gamma} \mapsto B$ such that

$$\theta' :: \Theta_1, \bar{\alpha} \sqsubseteq \Theta, \bar{\alpha} \text{ ⑯}$$

$$\theta \equiv \theta' :: \Theta_0, \bar{\alpha} \sqsubseteq \Theta, \bar{\alpha} \text{ ⑰}$$

Thus $(\theta' :: \Theta_1, \bar{\alpha}, x : \hat{\beta} \sqsubseteq \Theta, \bar{\alpha}, x : A; B)$ is a solution to the type inference problem $(\Theta_1, \bar{\alpha}, x : \hat{\beta}; \mathbf{c}; \hat{\gamma}; M)$. IH on ① gives

$$\Theta_1, \bar{\alpha}, x : \hat{\beta} [\hat{\gamma}] \vdash_{\mathbf{c}} M \Rightarrow B' \vdash \Theta_2, \bar{\alpha}, x : \hat{\beta}, \Xi \text{ ②}$$

$$\theta_1 :: \Theta_2, \bar{\alpha}, x : \hat{\beta}, \Xi \sqsubseteq \Theta, \bar{\alpha}, x : A \text{ ③}$$

$$\Theta, \bar{\alpha}, x : A \vdash \theta_1 B' \equiv B \text{ ④}$$

$$\theta' \equiv \theta_1 :: \Theta_1, \bar{\alpha}, x : \hat{\beta} \sqsubseteq \Theta, \bar{\alpha}, x : A \text{ ⑤}$$

By ⑬, ②, and I-AbsCHECK, we have

$$\Theta_0 [\forall \bar{\alpha}. \hat{\alpha}] \vdash_{\mathbf{c}} \lambda x. M \Rightarrow \forall \bar{\alpha}. \hat{\beta} \rightarrow B' \vdash \Theta_2 \text{ ⑥.}$$

By ③ and metasubstitution weakening, we have $\theta_1 :: \Theta_2 \sqsubseteq \Theta$ ⑦. By ⑤ and definition of metasubstitution, we have $\theta' \equiv \theta_1 :: \Theta_1, \bar{\alpha} \sqsubseteq \Theta, \bar{\alpha}$ ⑧. By ⑰, ⑧, and definition of metasubstitution, we have $\theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta$ ⑭. By ③ and ④, we have $\Theta \vdash \theta_1 (\forall \bar{\alpha}. \hat{\beta} \rightarrow B') \equiv \forall \bar{\alpha}. A \rightarrow B$ ⑮. Our goal follows from ⑥, ⑦, ⑮, and ⑭.

Case

$$\frac{\text{SD-Abs} \quad \mathbf{m} \neq \mathbf{C} \quad \Theta, x : A \vdash_{\mathbf{m}} M : B \text{ ①}}{\Theta \vdash_{\mathbf{m}} \lambda x. M : A \rightarrow B}$$

The type inference problem is $(\Theta_0; \mathbf{m}; P_0; \lambda x. M)$ where $P_0 :^{\text{ty}} \mathbf{m}$. The solution is $(\theta :: \Theta_0 \sqsubseteq \Theta; A \rightarrow B)$. We know that $P_0 = \mathbf{L}$ or P_0 is of function shape. Let

$$\Theta_0 \vdash P_0 \triangleright P \rightarrow Q \vdash \Theta_1 \text{ ⑫}$$

and let $(\bar{\alpha}; A') = \text{guess}(P)$, i.e., each ghost in P is filled with a fresh flexible variable $\hat{\alpha}$. Since $(\theta; A \rightarrow B)$ is a solution, we have $\Theta \vdash \theta P_0 \blacktriangleright A \rightarrow B$. Then by ⑫, definition of splitting, we have a metasubstitution θ' such that

$$\theta' :: \Theta_1 \sqsubseteq \Theta \quad (13)$$

$$\theta \equiv \theta' :: \Theta_0 \sqsubseteq \Theta \quad (14)$$

$$\Theta \vdash \theta'(P \rightarrow Q) \blacktriangleright A \rightarrow B \quad (15)$$

Thus $\Theta \vdash \theta' P \blacktriangleright A$, i.e., A refines each ghost in P with a monotype. Let ζ be the substitution that maps each fresh $\hat{\alpha}$ to the monotype in A that refines the corresponding ghost in P . Then we have a metasubstitution $\theta'' = (\theta', \zeta)$ such that

$$\theta'' :: \Theta_1, \bar{\alpha} \sqsubseteq \Theta \quad (16)$$

$$\Theta \vdash \theta'' A' \equiv A \quad (17)$$

$$\Theta \vdash \theta'' Q \blacktriangleright B \quad (18)$$

By ①, ⑪, and ⑫, we have $(\theta'' :: \Theta_1, \bar{\alpha}, x : A' \sqsubseteq \Theta, x : A; B)$ is a solution to the type inference problem $(\Theta_1, \bar{\alpha}, x : A'; \text{m-}; Q; M)$. By IH on this solution, we have

$$\Theta_1, \bar{\alpha}, x : A' [Q]_{\text{m-}} M \Rightarrow B' \vdash \Theta_2, x : A', \Xi \quad (2)$$

$$\theta_1 :: \Theta_2, x : A', \Xi \sqsubseteq \Theta, x : A \quad (3)$$

$$\Theta, x : A \vdash \theta_1 B' \equiv B \quad (5)$$

$$\theta'' \equiv \theta_1 :: \Theta_1, \bar{\alpha}, x : A' \sqsubseteq \Theta, x : A \quad (6)$$

By ⑭, ⑥, and definition of metasubstitution, we have

$$\theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \quad (7).$$

By ⑫, $\text{guess}(P) = (\bar{\alpha}; A')$, ②, and I-Abs, we have

$$\Theta_0 [P_0]_{\text{m}} \lambda x. M \Rightarrow A' \rightarrow B' \vdash \Theta_2, \Xi \quad (4).$$

By ③ and metasubstitution weakening, we have $\theta_1 :: \Theta_2, \Xi \sqsubseteq \Theta$ ⑧. By ⑥ and ⑪, we have $\Theta \vdash \theta_1 A' \equiv A$ ⑨. By ⑨ and ⑤, we have $\Theta \vdash \theta_1 (A' \rightarrow B') \equiv A \rightarrow B$ ⑩. Our goal follows from ④, ⑧, ⑩, and ⑦.

Case

$$\frac{\text{SD-AbsAnnoCheck} \quad \Theta, \bar{\alpha} \vdash A \equiv A' \quad (9) \quad \Theta, \bar{\alpha}, x : A \vdash_c M : B \quad (1)}{\Theta \vdash_c \lambda x^A. M : \forall \bar{\alpha}. A' \rightarrow B}$$

The type inference problem is $(\Theta_0; C; A_p; \lambda x^A. M)$ where $A_p :^{\text{ty}} C$. The solution is $(\theta :: \Theta_0 \sqsubseteq \Theta; \forall \bar{\alpha}. A' \rightarrow B)$. We have $\Theta \vdash \theta A_p \blacktriangleright \forall \bar{\alpha}. A' \rightarrow B$. By this refinement, the only possible shapes of A_p are $\forall \bar{\alpha}. A_0 \rightarrow B_0$ and $\forall \bar{\alpha}. \hat{\alpha}$. Case analysis on the shape of A_p .

Subcase $A_p = \forall \bar{\alpha}. A_0 \rightarrow B_0$. By the solution $(\theta; \forall \bar{\alpha}. A' \rightarrow B)$, we have

$$\Theta, \bar{\alpha} \vdash \theta A_0 \equiv A' \text{ (12)} \quad \text{and} \quad \Theta, \bar{\alpha} \vdash \theta B_0 \equiv B \text{ (13)}.$$

By (12) and (9), $(\theta :: \Theta_0, \bar{\alpha} \sqsubseteq \Theta, \bar{\alpha}; A)$ is a solution to the consistency problem $(\Theta_0, \bar{\alpha}; A; A_0)$. Then by Lemma 26, and since precision on types is equivalence, we have

$$\Theta_0, \bar{\alpha} \vdash A_0 \sim A \dashv \Theta_1, \bar{\alpha} \text{ (14)}$$

$$\theta' :: \Theta_1, \bar{\alpha} \sqsubseteq \Theta, \bar{\alpha} \text{ (15)}$$

$$\theta \equiv \theta' :: \Theta_0, \bar{\alpha} \sqsubseteq \Theta, \bar{\alpha} \text{ (16)}$$

By (13) and (16), $(\theta' :: \Theta_1, \bar{\alpha}, x : A \sqsubseteq \Theta, \bar{\alpha}, x : A; B)$ is a solution to the type inference problem $(\Theta_1, \bar{\alpha}, x : A; C; B_0; M)$. IH on (1) gives

$$\Theta_1, \bar{\alpha}, x : A \vdash_{B_0} M \Rightarrow B_1 \dashv \Theta_2, \bar{\alpha}, x : A, \Xi \text{ (2)}$$

$$\theta_1 :: \Theta_2, \bar{\alpha}, x : A, \Xi \sqsubseteq \Theta, \bar{\alpha}, x : A \text{ (3)}$$

$$\Theta, \bar{\alpha}, x : A \vdash \theta_1 B_1 \equiv B \text{ (4)}$$

$$\theta' \equiv \theta_1 :: \Theta_1, \bar{\alpha}, x : A \sqsubseteq \Theta, \bar{\alpha}, x : A \text{ (5)}$$

By definition of splitting, we have

$$\Theta_0, \bar{\alpha} \vdash A_0 \rightarrow B_0 \triangleright A_0 \rightarrow B_0 \dashv \Theta_0, \bar{\alpha} \text{ (11)}.$$

By (11), (14), (2), and I-ABSANNOCHECK, we have

$$\Theta_0 \vdash_{\forall \bar{\alpha}. A_0 \rightarrow B_0} \lambda x^A. M \Rightarrow \forall \bar{\alpha}. A \rightarrow B_1 \dashv \Theta_2 \text{ (6)}.$$

By (3) and metasubstitution weakening, we have $\theta_1 :: \Theta_2 \sqsubseteq \Theta$ (7). By (5), (16), and definition of metasubstitution, we have $\theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta$ (8). By (9) and definition of metasubstitution, we have $\Theta, \bar{\alpha} \vdash \theta_1 A \equiv A'$ (17). By (17) and (4), we have

$$\Theta \vdash \theta_1 (\forall \bar{\alpha}. A \rightarrow B_1) \equiv \forall \bar{\alpha}. A' \rightarrow B \text{ (18)}.$$

Our goal follows from (6), (7), (18), and (8).

Subcase $A_p = \forall \bar{\alpha}. \hat{a}$. By the original solution, we have $\Theta, \bar{\alpha} \vdash \theta \hat{a} \equiv A' \rightarrow B$ (12).

Assuming fresh $\hat{\beta}$ and $\hat{\gamma}$, we have

$$\Theta_0, \bar{\alpha} \vdash \hat{a} \triangleright \hat{\beta} \rightarrow \hat{\gamma} \dashv \Theta_1, \bar{\alpha} \text{ (13)}.$$

Since $\theta \hat{a}$ is equivalent to $A' \rightarrow B$, and by definition of splitting $\hat{\beta}$ and $\hat{\gamma}$ are introduced immediately before the assignment for $\hat{a}$, the components A' and B are well-scoped at the positions of $\hat{\beta}$ and $\hat{\gamma}$ in Θ_1 . By (12) and (13), we have a metasubstitution θ' extending θ with $\hat{\beta} \mapsto A'$ and $\hat{\gamma} \mapsto B$ such that

$$\theta' :: \Theta_1, \bar{\alpha} \sqsubseteq \Theta, \bar{\alpha} \text{ (16)}$$

$$\theta \equiv \theta' :: \Theta_0, \bar{\alpha} \sqsubseteq \Theta, \bar{\alpha} \text{ (17)}$$

By construction of θ' and ⑨, $(\theta' :: \Theta_1, \bar{\alpha} \sqsubseteq \Theta, \bar{\alpha}; A)$ is a solution to the consistency problem $(\Theta_1, \bar{\alpha}; A; \hat{\beta})$. Then by Lemma 26, and since precision on types is equivalence, we have

$$\Theta_1, \bar{\alpha} \vdash \hat{\beta} \sim A \dashv \Theta_2, \bar{\alpha} \text{ ⑮}$$

$$\theta'' :: \Theta_2, \bar{\alpha} \sqsubseteq \Theta, \bar{\alpha} \text{ ⑯}$$

$$\theta' \equiv \theta'' :: \Theta_1, \bar{\alpha} \sqsubseteq \Theta, \bar{\alpha} \text{ ⑰}$$

Thus $(\theta'' :: \Theta_2, \bar{\alpha}, x : A \sqsubseteq \Theta, \bar{\alpha}, x : A; B)$ is a solution to the type inference problem $(\Theta_2, \bar{\alpha}, x : A; C; \hat{\gamma}; M)$. IH on ① gives

$$\Theta_2, \bar{\alpha}, x : A \vdash [\hat{\gamma}]_c M \Rightarrow B' \dashv \Theta_3, \bar{\alpha}, x : A, \Xi \text{ ⑱}$$

$$\theta_1 :: \Theta_3, \bar{\alpha}, x : A, \Xi \sqsubseteq \Theta, \bar{\alpha}, x : A \text{ ⑲}$$

$$\Theta, \bar{\alpha}, x : A \vdash \theta_1 B' \equiv B \text{ ⑳}$$

$$\theta'' \equiv \theta_1 :: \Theta_2, \bar{\alpha}, x : A \sqsubseteq \Theta, \bar{\alpha}, x : A \text{ ㉑}$$

By ⑬, ⑮, ⑱, and I-ABSANNOCHECK, we have

$$\Theta_0 \vdash [\forall \bar{\alpha}. \hat{\alpha}]_c \lambda x^A. M \Rightarrow \forall \bar{\alpha}. A \rightarrow B' \dashv \Theta_3 \text{ ㉒}.$$

By ⑳ and metasubstitution weakening, we have $\theta_1 :: \Theta_3 \sqsubseteq \Theta$ ㉓. By ㉓, ㉑, ㉒, and definition of metasubstitution, we have $\theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta$ ㉔. By ㉓ and definition of metasubstitution, we have $\Theta, \bar{\alpha} \vdash \theta_1 A \equiv A'$ ㉕. By ㉕ and ㉒, we have

$$\Theta \vdash \theta_1 (\forall \bar{\alpha}. A \rightarrow B') \equiv \forall \bar{\alpha}. A' \rightarrow B \text{ ㉖}.$$

Our goal follows from ㉒, ㉓, ㉖, and ㉔.

Case

$$\frac{\text{SD-ABSANNO} \quad m \neq C \quad \Theta \vdash A \equiv A' \text{ ㉗} \quad \Theta, x : A \vdash_m M : B \text{ ㉘}}{\Theta \vdash \lambda x^A. M : A' \rightarrow B}$$

The type inference problem is $(\Theta_0; m; P_0; \lambda x^A. M)$ where $P_0 :^{ty} m$. The solution is $(\theta :: \Theta_0 \sqsubseteq \Theta; A' \rightarrow B)$. We know that $P_0 = \mathbb{Q}$ or P_0 is of function shape. Let

$$\Theta_0 \vdash P_0 \triangleright P \rightarrow Q \dashv \Theta_1 \text{ ㉙}$$

Since $(\theta; A' \rightarrow B)$ is a solution, we have $\Theta \vdash \theta P_0 \triangleright A' \rightarrow B$. Then by ㉙, definition of splitting, we have a metasubstitution θ' such that

$$\theta' :: \Theta_1 \sqsubseteq \Theta \text{ ㉚}$$

$$\theta \equiv \theta' :: \Theta_0 \sqsubseteq \Theta \text{ ㉛}$$

$$\Theta \vdash \theta'(P \rightarrow Q) \triangleright A' \rightarrow B$$

Thus $\Theta \vdash \theta' P \blacktriangleright A'$ and $\Theta \vdash \theta' Q \blacktriangleright B$ (15). Then by (9), $(\theta' :: \Theta_1 \sqsubseteq \Theta; A)$ is a solution to the consistency problem $(\Theta_1; P; A)$. Then by Lemma 26, and since precision on types is equivalence, we have

$$\Theta_1 \vdash A \sim P \dashv \Theta_2 \quad (16)$$

$$\theta'' :: \Theta_2 \sqsubseteq \Theta \quad (17)$$

$$\theta' \equiv \theta'' :: \Theta_1 \sqsubseteq \Theta \quad (18)$$

By (15) and (18), we have $\Theta, x : A \vdash \theta'' Q \blacktriangleright B$. Then by (1) and (17), $(\theta'' :: \Theta_2, x : A \sqsubseteq \Theta, x : A; B)$ is a solution to the type inference problem $(\Theta_2, x : A; m-; Q; M)$. IH on (1) gives

$$\Theta_2, x : A [Q]_{m-} M \Rightarrow B' \dashv \Theta_3, x : A, \Xi \quad (2)$$

$$\theta_1 :: \Theta_3, x : A, \Xi \sqsubseteq \Theta, x : A \quad (3)$$

$$\Theta, x : A \vdash \theta_1 B' \equiv B \quad (4)$$

$$\theta'' \equiv \theta_1 :: \Theta_2, x : A \sqsubseteq \Theta, x : A \quad (5)$$

By (12), (16), (2), and I-ABSANNO, we have

$$\Theta_0 [P_0]_{m-} \lambda x^A. M \Rightarrow A \rightarrow B' \dashv \Theta_3, \Xi \quad (6).$$

By (3) and metasubstitution weakening, we have $\theta_1 :: \Theta_3, \Xi \sqsubseteq \Theta$ (7). By (14), (18), (5), and definition of metasubstitution, we have $\theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta$ (8). By (9) and definition of metasubstitution, we have $\Theta \vdash \theta_1 A \equiv A'$ (9). By (9) and (4), we have $\Theta \vdash \theta_1 (A \rightarrow B') \equiv A' \rightarrow B$ (19). Our goal follows from (6), (7), (19), and (8).

Case

SD-APP

$$\frac{\begin{array}{c} \Theta \vdash_c N \Rightarrow P \quad (1) \\ \Theta \vdash P \blacktriangleright A \quad (2) \quad \Theta \vdash_{Fm} M : A \rightarrow B \quad (3) \quad \Theta \vdash_c N : A \quad (4) \end{array}}{\Theta \vdash_m M N : B}$$

The type inference problem is $(\Theta_0; m; Q; M N)$ and the solution is $(\theta :: \Theta_0 \sqsubseteq \Theta; B)$. Theorem 24 on (1) gives that

$$\Theta_0 [\Delta]_{c-} N \Rightarrow P' \dashv \Theta_1 \quad (5)$$

$$\theta_1 :: \Theta_1 \sqsubseteq \Theta$$

$$\Theta \vdash \theta_1 P' \leq P \quad (6)$$

$$\theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \quad (14)$$

Lemma 72 on (6) and (2) gives that $\Theta \vdash \theta_1 P' \blacktriangleright A$ (12). By the original solution $(\theta; B)$ and (14), we have $\Theta \vdash \theta_1 Q \blacktriangleright B$ (13). By (12) and (13), we have $\Theta \vdash \theta_1 (P' \rightarrow Q) \blacktriangleright$

$A \rightarrow B$. Then by ③ we have that $(\theta_1 :: \Theta_1 \sqsubseteq \Theta; A \rightarrow B)$ is a solution to the type inference problem $(\Theta_1; \text{F m}; P' \rightarrow Q; M)$. Then by IH on ③, we have

$$\Theta_1 [P' \rightarrow Q] \vdash_{\text{F m}} M \Rightarrow A_2 \rightarrow B_2 \dashv \Theta_2 \quad (7)$$

$$\theta_2 :: \Theta_2 \sqsubseteq \Theta \quad (15)$$

$$\Theta \vdash \theta_2(A_2 \rightarrow B_2) \equiv A \rightarrow B \quad (8)$$

$$\theta_2 \equiv \theta_1 :: \Theta_1 \sqsubseteq \Theta \quad (16)$$

⑧ gives $\Theta \vdash \theta_2 A_2 \equiv A$, which further gives $\Theta \vdash \theta_2 A_2 \blacktriangleright A$. Then by ④ we have $(\theta_2 :: \Theta_2 \sqsubseteq \Theta; A)$ is a solution to the type inference problem $(\Theta_2; \text{C}; A_2; N)$. Then by IH, we have

$$\Theta_2 [A_2] \vdash_{\text{C}} N \Rightarrow A_3 \dashv \Theta_3 \quad (9)$$

$$\theta_3 :: \Theta_3 \sqsubseteq \Theta \quad (17)$$

$$\Theta \vdash \theta_3 A_3 \equiv A$$

$$\theta_3 \equiv \theta_2 :: \Theta_2 \sqsubseteq \Theta \quad (18)$$

By ⑧, ①⑥, and ①④, we have $\theta_3 \equiv \theta :: \Theta_0 \sqsubseteq \Theta \quad (10)$. Moreover, ⑧ gives $\Theta \vdash \theta_2 B_2 \equiv B$, which together with ①⑧ gives $\Theta \vdash \theta_3 B_2 \equiv B \quad (19)$. By I-APP on ⑤, ⑦, and ⑨, we have

$$\Theta_0 [Q] \vdash_{\text{m}} M N \Rightarrow B_2 \dashv \Theta_3 \quad (11)$$

Our goal follows from ①①, ①⑦, ①⑩, and ①⑨.

□

D.3 Skeleton inference

In this section, we prove Theorems 23 and 24. To avoid repeating the skeleton-subtyping premises in P-UNIT, P-VAR, and P-ANNO, we use an equivalent presentation in this section. We write $O ::= () \mid x \mid M : A$ for base terms and use the following four rules:

$\frac{}{\Gamma \vdash_{\text{I}} () \Rightarrow 1} \quad \text{P-UNIT}$	$\frac{\Gamma \ni x : P}{\Gamma \vdash_{\text{I}} x \Rightarrow P} \quad \text{P-VAR}$	$\frac{}{\Gamma \vdash_{\text{I}} (M : A) \Rightarrow A} \quad \text{P-ANNO}$	$\frac{\Gamma \vdash_{\text{I}} O \Rightarrow P \quad \Gamma \vdash_{\text{m}}^{\text{sk}} P \leq Q}{\Gamma \vdash_{\text{m}} O \Rightarrow Q} \quad \text{P-SUB}$
--	--	---	--

This presentation is local to the proofs.

Theorem 23 (Soundness of skeleton inference). *Given a skeleton inference problem $(\Theta_0; \text{m}; P_0; M)$, if $\Theta_0 [P_0] \vdash_{\text{m}} M \Rightarrow P \dashv \Theta_1$, then $(\Theta_0 \sqsubseteq \Theta_1; P)$ is a solution.*

Proof. Our goal is to show that $\Theta_1 \vdash_{\text{m}} M \Rightarrow P$ where $\Theta_1 \vdash P_0 \blacktriangleright P$ and $\Theta_0 \sqsubseteq \Theta_1$ is a valid metasubstitution. By induction on the derivation of $\Theta_0 [P_0] \vdash_{\text{m}} M \Rightarrow P \dashv \Theta_1$.

Case

$$\frac{\text{PI-UNIT} \quad \Theta_0 [P_0]_{\mathfrak{m}}^{\text{sk}} 1 \leq P \dashv \Theta_1 \text{ (1)}}{\Theta_0 [P_0]_{\mathfrak{m}} () \Rightarrow P \dashv \Theta_1}$$

Theorem 27 on (1) gives that $(\Theta_0 \sqsubseteq \Theta_1; P)$ is a solution to the subtyping problem $(\Theta_0; \text{sk}; \mathfrak{m}; P_0; 1)$, which gives $\Theta_1 \vdash_{\mathfrak{m}}^{\text{sk}} 1 \leq P$ (2) where $\Theta_1 \vdash P_0 \blacktriangleright P$ (3). By P-UNIT, (2), and P-SUB, we have $\Theta_1 \vdash_{\mathfrak{m}} () \Rightarrow P$ (4). Our goal follows from (3) and (4).

Case

$$\frac{\text{PI-VAR} \quad \Theta_0 \ni x : P \quad \Theta_0 [P_0]_{\mathfrak{m}}^{\text{sk}} P \leq Q \dashv \Theta_1 \text{ (1)}}{\Theta_0 [P_0]_{\mathfrak{m}} x \Rightarrow Q \dashv \Theta_1}$$

Theorem 27 on (1) gives that $(\Theta_0 \sqsubseteq \Theta_1; Q)$ is a solution to the subtyping problem $(\Theta_0; \text{sk}; \mathfrak{m}; P_0; P)$, which gives $\Theta_1 \vdash_{\mathfrak{m}}^{\text{sk}} P \leq Q$ (2) where $\Theta_1 \vdash P_0 \blacktriangleright Q$ (3). By $\Theta_0 \ni x : P$ and $\Theta_0 \sqsubseteq \Theta_1$, we have $\Theta_1 \ni x : P$. Then by P-VAR, (2), and P-SUB, we have $\Theta_1 \vdash_{\mathfrak{m}} x \Rightarrow Q$ (4). Our goal follows from (3) and (4).

Case

$$\frac{\text{PI-ANNO} \quad \Theta_0 [P]_{\mathfrak{m}}^{\text{sk}} A \leq Q \dashv \Theta_1 \text{ (1)}}{\Theta_0 [P]_{\mathfrak{m}} (M : A) \Rightarrow Q \dashv \Theta_1}$$

Theorem 27 on (1) gives that $(\Theta_0 \sqsubseteq \Theta_1; Q)$ is a solution to the subtyping problem $(\Theta_0; \text{sk}; \mathfrak{m}; P; A)$, which gives $\Theta_1 \vdash_{\mathfrak{m}}^{\text{sk}} A \leq Q$ (2) where $\Theta_1 \vdash P \blacktriangleright Q$ (3). By P-ANNO, (2), and P-SUB, we have $\Theta_1 \vdash_{\mathfrak{m}} (M : A) \Rightarrow Q$ (4). Our goal follows from (3) and (4).

Case

$$\frac{\text{PI-FREEZE} \quad \Theta_0 [\mathbb{A}]_{\mathfrak{I}} M \Rightarrow P \dashv \Theta_1 \text{ (1)}}{\Theta_0 [\mathbb{A}]_{\mathfrak{C}} [M] \Rightarrow P \dashv \Theta_1}$$

IH on (1) gives that $(\Theta_0 \sqsubseteq \Theta_1; P)$ is a solution to the skeleton inference problem $(\Theta_0; \mathfrak{I}; \mathbb{A}; M)$, which gives $\Theta_1 \vdash_{\mathfrak{I}} M \Rightarrow P$ (2). By P-FREEZE, we have $\Theta_1 \vdash_{\mathfrak{C}} [M] \Rightarrow P$ (3). Our goal follows from (3) and the refinement $\Theta_1 \vdash \mathbb{A} \blacktriangleright P$.

Case

$$\frac{\text{PI-ABS} \quad \begin{array}{l} \mathfrak{m} \neq \mathfrak{C} \quad \Theta_0 \vdash P_0 \blacktriangleright P \rightarrow Q_0 \dashv \Theta_1 \text{ (1)} \\ \Theta_1, x : P [Q_0]_{\mathfrak{m}-} M \Rightarrow Q \dashv \Theta_2, x : P, \Xi \text{ (2)} \end{array}}{\Theta_0 [P_0]_{\mathfrak{m}} \lambda x. M \Rightarrow P \rightarrow Q \dashv \Theta_2, \Xi}$$

IH on ② gives that $(\Theta_1, x : P \sqsubseteq \Theta_2, x : P, \Xi \text{ ③}; Q)$ is a solution to the skeleton inference problem $(\Theta_1, x : P; m-; Q_0; M)$, which gives

$$\Theta_2, x : P, \Xi \vdash_{m-} M \Rightarrow Q \text{ ④}$$

$$\Theta_2, x : P, \Xi \vdash Q_0 \blacktriangleright Q \text{ ⑤}$$

By structural rules and P-Abs, we have $\Theta_2, \Xi \vdash_{\mathfrak{m}} \lambda x. M \Rightarrow P \rightarrow Q \text{ ⑥}$. By ①, Lemma 70, and stability, we have $\Theta_2, \Xi \vdash P_0 \blacktriangleright P \rightarrow Q_0$. Since refinement judgements are stable under dropping term variables, ⑤ gives $\Theta_2, \Xi \vdash Q_0 \blacktriangleright Q$. Thus, by transitivity of refinement, $\Theta_2, \Xi \vdash P_0 \blacktriangleright P \rightarrow Q \text{ ⑦}$. By ①, ③, and transitivity of information increase, we have $\Theta_0 \sqsubseteq \Theta_2, \Xi \text{ ⑧}$. Our goal follows from ⑥, ⑦, and ⑧.

Case

$$\begin{array}{c} \text{PI-AbsANNO} \\ m \neq C \quad \Theta_0 \vdash P_0 \blacktriangleright P \rightarrow Q_0 \vdash \Theta_1 \text{ ①} \quad \Theta_1 \vdash A \sim P \vdash \Theta_2 \text{ ②} \\ \hline \Theta_2, x : A [Q_0] \vdash_{m-} M \Rightarrow Q \vdash \Theta_3, x : A, \Xi \text{ ③} \\ \hline \Theta_0 [P_0] \vdash_{\mathfrak{m}} \lambda x^A. M \Rightarrow A \rightarrow Q \vdash \Theta_3, \Xi \end{array}$$

Lemma 25 on ② gives that, for the hidden consistency output A' ,

$$\Theta_1 \sqsubseteq \Theta_2 \quad \Theta_2 \vdash P \blacktriangleright A' \quad \Theta_2 \vdash A' \equiv A$$

Thus, by preservation of refinement under type equivalence, we have $\Theta_2 \vdash P \blacktriangleright A \text{ ④}$.

IH on ③ gives that $(\Theta_2, x : A \sqsubseteq \Theta_3, x : A, \Xi \text{ ⑤}; Q)$ is a solution to the skeleton inference problem $(\Theta_2, x : A; m-; Q_0; M)$, which gives

$$\Theta_3, x : A, \Xi \vdash_{m-} M \Rightarrow Q \text{ ⑥}$$

$$\Theta_3, x : A, \Xi \vdash Q_0 \blacktriangleright Q \text{ ⑦}$$

By structural rules and P-AbsANNO, we have $\Theta_3, \Xi \vdash_{\mathfrak{m}} \lambda x^A. M \Rightarrow A \rightarrow Q \text{ ⑧}$.

By ①, Lemma 70, and stability, we have $\Theta_3, \Xi \vdash P_0 \blacktriangleright P \rightarrow Q_0 \text{ ⑨}$. By stability on ④, we have $\Theta_3, \Xi \vdash P \blacktriangleright A$. Since refinement judgements are stable under dropping term variables, ⑦ gives $\Theta_3, \Xi \vdash Q_0 \blacktriangleright Q$. Thus, by ⑨, ⑦, and transitivity of refinement, we have $\Theta_3, \Xi \vdash P_0 \blacktriangleright A \rightarrow Q \text{ ⑩}$. By ①, $\Theta_1 \sqsubseteq \Theta_2$, ⑤, and transitivity of information increase, we have $\Theta_0 \sqsubseteq \Theta_3, \Xi \text{ ⑪}$. Our goal follows from ⑧, ⑩, and ⑪.

Case

$$\begin{array}{c} \text{PI-AbsCHECK} \\ \Theta_0, x : \mathbb{A} [\mathbb{A}] \vdash_c M \Rightarrow Q \vdash \Theta_1, x : \mathbb{A}, \Xi \text{ ①} \\ \hline \Theta_0 [\mathbb{A}] \vdash_c \lambda x. M \Rightarrow \mathbb{A}. \mathbb{A} \rightarrow Q \vdash \Theta_1, \Xi \end{array}$$

IH on ① gives

$$\Theta_1, x : \mathbb{A}, \Xi \vdash_c M \Rightarrow Q \text{ ②}$$

$$\Theta_0, x : \mathbb{A} \sqsubseteq \Theta_1, x : \mathbb{A}, \Xi \text{ ③}$$

By structural rules and P-AbsCHECK, we have $\Theta_1, \Xi \vdash_c \lambda x. M \Rightarrow \mathbb{A}. \mathbb{A} \rightarrow Q \text{ ④}$. By refinement, we have $\Theta_1, \Xi \vdash \mathbb{A} \blacktriangleright \mathbb{A}. \mathbb{A} \rightarrow Q \text{ ⑤}$. By ③ and weakening, we also have $\Theta_0 \sqsubseteq \Theta_1, \Xi \text{ ⑥}$. Our goal follows from ④, ⑤, and ⑥.

Case

PI-AbsANNOCHECK

$$\Theta_0, x : A \vdash_c M \Rightarrow Q \vdash \Theta_1, x : A, \Xi \text{ ①}$$

$$\Theta_0 \vdash_c \lambda x^A. M \Rightarrow \mathbb{A}. A \rightarrow Q \vdash \Theta_1, \Xi$$

IH on ① gives

$$\Theta_1, x : A, \Xi \vdash_c M \Rightarrow Q \text{ ②}$$

$$\Theta_0, x : A \sqsubseteq \Theta_1, x : A, \Xi \text{ ③}$$

By structural rules and P-AbsANNOCHECK, we have $\Theta_1, \Xi \vdash_c \lambda x^A. M \Rightarrow \mathbb{A}. A \rightarrow Q \text{ ④}$. By refinement, we have $\Theta_1, \Xi \vdash \mathbb{A} \blacktriangleright \mathbb{A}. A \rightarrow Q \text{ ⑤}$. By ③ and weakening, we also have $\Theta_0 \sqsubseteq \Theta_1, \Xi \text{ ⑥}$. Our goal follows from ④, ⑤, and ⑥.

Case

PI-APP

$$\Theta_0 \vdash_c N \Rightarrow P_0 \vdash \Theta_1 \text{ ①}$$

$$\Theta_1 [P_0 \rightarrow Q_0] \vdash_{\text{fm}} M \Rightarrow P \rightarrow Q \vdash \Theta_2 \text{ ②}$$

$$\Theta_0 [Q_0] \vdash_{\text{m}} M N \Rightarrow Q \vdash \Theta_2$$

IH on ① gives

$$\Theta_1 \vdash_c N \Rightarrow P_0 \text{ ③}$$

$$\Theta_0 \sqsubseteq \Theta_1 \text{ ④}$$

IH on ② gives that $(\Theta_1 \sqsubseteq \Theta_2 \text{ ⑤}; P \rightarrow Q)$ is a solution to the skeleton inference problem $(\Theta_1; \text{fm}; P_0 \rightarrow Q_0; M)$, which gives

$$\Theta_2 \vdash_{\text{fm}} M \Rightarrow P \rightarrow Q \text{ ⑥}$$

$$\Theta_2 \vdash P_0 \rightarrow Q_0 \blacktriangleright P \rightarrow Q \text{ ⑦}$$

By definition of skeleton refinement and ⑦, we have

$$\Theta_2 \vdash P_0 \blacktriangleright P \text{ ⑧} \quad \text{and} \quad \Theta_2 \vdash Q_0 \blacktriangleright Q \text{ ⑨}.$$

By stability on ③, we have $\Theta_2 \vdash_c N \Rightarrow P_0$. By P-APP on this judgement, ⑧, and ⑥, we have $\Theta_2 \vdash_{\text{m}} M N \Rightarrow Q \text{ ⑩}$. By ④, ⑤, and transitivity of information increase, we have $\Theta_0 \sqsubseteq \Theta_2 \text{ ⑪}$. Our goal follows from ⑩, ⑨, and ⑪.

□

Theorem 24 (Completeness and optimality of skeleton inference). *Given a skeleton inference problem $(\Theta_0; \mathfrak{m}; P_0; M)$, if there exists a solution $(\theta :: \Theta_0 \sqsubseteq \Theta; Q)$, then $\Theta_0 [P_0]_{\mathfrak{m}} M \Rightarrow P \dashv \Theta_1$ gives its optimal solution $(\Theta_0 \sqsubseteq \Theta_1; P)$.*

Proof. It is enough to prove that if $(\theta :: \Theta_0 \sqsubseteq \Theta; Q)$ is a solution to the skeleton inference problem $(\Theta_0; \mathfrak{m}; P_0; M)$, then we have $\Theta_0 [P_0]_{\mathfrak{m}} M \Rightarrow P \dashv \Theta_1$ and $\theta_1 :: \Theta_1 \sqsubseteq \Theta$ and $\Theta \vdash \theta_1 P \leq Q$ and $\theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta$. By lexicographic induction on the size of M and the height of the derivation of $\Theta \vdash M \Rightarrow Q$ where $\Theta \vdash \theta P_0 \blacktriangleright Q$ and $\theta :: \Theta_0 \sqsubseteq \Theta$.

Case

P-UNIT

$$\frac{}{\Theta \vdash_{\mathbf{I}} () \Rightarrow 1}$$

In this case $\mathfrak{m} = \mathbf{I}$ and $P_0 = \mathbb{I}$. By SI-INFER and PI-UNIT, we have $\Theta_0 [\mathbb{I}]_{\mathbf{I}} () \Rightarrow 1 \dashv \Theta_0$. Our goal follows from $\theta :: \Theta_0 \sqsubseteq \Theta$ and $\Theta \vdash \theta 1 \leq 1$.

Case

P-VAR

$$\frac{\Theta \ni x : P}{\Theta \vdash_{\mathbf{I}} x \Rightarrow P}$$

In this case $\mathfrak{m} = \mathbf{I}$ and $P_0 = \mathbb{I}$. By $\theta :: \Theta_0 \sqsubseteq \Theta$ and $\Theta \ni x : P$, we have $\Theta_0 \ni x : P'$ and $\theta P' = P$ syntactically, which gives $\Theta \vdash \theta P' \leq P$ ①. By SI-INFER and PI-VAR, we have $\Theta_0 [\mathbb{I}]_{\mathbf{I}} x \Rightarrow P' \dashv \Theta_0$. Our goal follows from $\theta :: \Theta_0 \sqsubseteq \Theta$ and ①.

Case

P-ANNO

$$\frac{}{\Theta \vdash_{\mathbf{I}} (M : A) \Rightarrow A}$$

In this case $\mathfrak{m} = \mathbf{I}$ and $P_0 = \mathbb{I}$. By SI-INFER and PI-ANNO, we have $\Theta_0 [\mathbb{I}]_{\mathbf{I}} (M : A) \Rightarrow A \dashv \Theta_0$. Our goal follows from $\theta :: \Theta_0 \sqsubseteq \Theta$ and $\Theta \vdash \theta A \leq A$.

Case

P-SUB

$$\frac{\Theta \vdash_{\mathbf{I}} O \Rightarrow P \text{ ①} \quad \Theta \vdash_{\mathfrak{m}}^{\text{sk}} P \leq Q \text{ ②}}{\Theta \vdash_{\mathfrak{m}} O \Rightarrow Q}$$

Since $(\theta :: \Theta_0 \sqsubseteq \Theta; Q)$ is a solution to the skeleton inference problem, we have

$$\Theta \vdash \theta P_0 \blacktriangleright Q \text{ ⑬} \quad \text{and} \quad \theta :: \Theta_0 \sqsubseteq \Theta \text{ ⑭}.$$

By ⑭ and the refinement of $\mathbb{L}$, IH on ① gives

$$\begin{aligned} \Theta_0 [\mathbb{L}]_{\mathbf{I}} O &\Rightarrow P' \dashv \Theta_1 \text{ ③} \\ \theta_1 :: \Theta_1 &\sqsubseteq \Theta \text{ ④} \\ \Theta &\vdash \theta_1 P' \leq P \text{ ⑤} \\ \theta &\equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \text{ ⑥} \end{aligned}$$

By Lemma 71 on ⑤ and ②, we have $\Theta \vdash_m^{\text{sk}} \theta_1 P' \leq Q$ ⑦. By ⑥ and ⑬, we have $\Theta \vdash \theta_1 P_0 \blacktriangleright Q$. Thus $(\theta_1 :: \Theta_1 \sqsubseteq \Theta; Q)$ is a solution to the subtyping problem $(\Theta_1; \text{sk}; m; P_0; P')$. By Theorem 28, we have

$$\begin{aligned} \Theta_1 [P_0]_{\mathbf{m}}^{\text{sk}} P' &\leq Q' \dashv \Theta_2 \text{ ⑧} \\ \theta_2 :: \Theta_2 &\sqsubseteq \Theta \text{ ⑨} \\ \Theta &\vdash \theta_2 Q' \leq Q \text{ ⑩} \\ \theta_1 &\equiv \theta_2 :: \Theta_1 \sqsubseteq \Theta \text{ ⑪} \end{aligned}$$

Case analysis on O .

Subcase $O = ()$. Inversion on ③ gives $\Theta_1 = \Theta_0$ and $P' = 1$. By PI-UNIT on ⑧, we have $\Theta_0 [P_0]_{\mathbf{m}} O \Rightarrow Q' \dashv \Theta_2$ ⑫.

Subcase $O = x$. Inversion on ③ gives $\Theta_0 \ni x : P'$ and $\Theta_1 = \Theta_0$. By PI-VAR on ⑧, we have $\Theta_0 [P_0]_{\mathbf{m}} O \Rightarrow Q' \dashv \Theta_2$ ⑫.

Subcase $O = (M : A)$. Inversion on ③ gives $P' = A$. By PI-ANNO on ⑧, we have $\Theta_0 [P_0]_{\mathbf{m}} O \Rightarrow Q' \dashv \Theta_2$ ⑫.

By ⑥, ⑪, and $\Theta_0 \sqsubseteq \Theta_1$, we have $\theta \equiv \theta_2 :: \Theta_0 \sqsubseteq \Theta$ ⑮. Our goal follows from ⑫, ⑨, ⑩, and ⑮.

Case

$$\begin{array}{c} \text{P-FREEZE} \\ \hline \Theta \vdash_{\mathbf{I}} M \Rightarrow P \text{ ①} \\ \hline \Theta \vdash_{\mathbf{c}} [M] \Rightarrow P \end{array}$$

In this case the skeleton inference problem is $(\Theta_0; \text{c}; \mathbb{L}; [M])$. IH on ① gives

$$\begin{aligned} \Theta_0 [\mathbb{L}]_{\mathbf{I}} M &\Rightarrow P' \dashv \Theta_1 \text{ ②} \\ \theta_1 :: \Theta_1 &\sqsubseteq \Theta \text{ ③} \\ \Theta &\vdash \theta_1 P' \leq P \text{ ④} \\ \theta &\equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \text{ ⑥} \end{aligned}$$

By PI-FREEZE on ②, we have $\Theta_0 [\mathbb{L}]_{\mathbf{c}} [M] \Rightarrow P' \dashv \Theta_1$ ⑤. Our goal follows from ⑤, ③, ④, and ⑥.

Case

$$\frac{\text{P-ABS} \quad \text{m} \neq \text{C} \quad \Theta, x : P \vdash_{\text{m}} M \Rightarrow Q \text{ (1)}}{\Theta \vdash_{\text{m}} \lambda x. M \Rightarrow P \rightarrow Q}$$

The solution $(\theta :: \Theta_0 \sqsubseteq \Theta; P \rightarrow Q)$ to the skeleton inference problem $(\Theta_0; \text{m}; P_0; \lambda x. M)$ gives

$$\Theta \vdash \theta P_0 \blacktriangleright P \rightarrow Q \text{ (19)} \quad \text{and} \quad \theta :: \Theta_0 \sqsubseteq \Theta \text{ (20)}.$$

Let

$$\Theta_0 \vdash P_0 \blacktriangleright P_1 \rightarrow Q_1 \vdash \Theta_1 \text{ (2)}.$$

By (19), (20), (2), and definition of splitting, there is a metasubstitution θ' such that

$$\theta' :: \Theta_1 \sqsubseteq \Theta \text{ (3)}$$

$$\theta \equiv \theta' :: \Theta_0 \sqsubseteq \Theta \text{ (4)}$$

$$\Theta \vdash \theta'(P_1 \rightarrow Q_1) \blacktriangleright P \rightarrow Q \text{ (5)}$$

By inversion on (5), we have

$$\Theta \vdash \theta' P_1 \blacktriangleright P \text{ (13)} \quad \text{and} \quad \Theta \vdash \theta' Q_1 \blacktriangleright Q \text{ (14)}.$$

By definition, (13) gives $\Theta \vdash \theta' P_1 \leq P$ (15). By reflexivity of context precision and (15), we have

$$\Theta, x : \theta' P_1 \leq_{\text{ctx}} \Theta, x : P \text{ (16)}.$$

By weakening (14), we have

$$\Theta, x : \theta' P_1 \vdash \theta' Q_1 \blacktriangleright Q \text{ (17)}.$$

Lemma 73 on (16) and (1) gives $\Theta, x : \theta' P_1 \vdash_{\text{m}} M \Rightarrow Q$ (25). By (3) and the definition of metasubstitution, we have

$$\theta' :: \Theta_1, x : P_1 \sqsubseteq \Theta, x : \theta' P_1 \text{ (27)}.$$

By (25), (17), and (27), $(\theta' :: \Theta_1, x : P_1 \sqsubseteq \Theta, x : \theta' P_1; Q)$ (28) is a solution to $(\Theta_1, x : P_1; \text{m}; Q_1; M)$. IH on (25) with solution (28) gives

$$\Theta_1, x : P_1 [Q_1] \vdash_{\text{m}} M \Rightarrow Q' \vdash \Theta_2, x : P_1, \Xi \text{ (6)}$$

$$\theta_1 :: \Theta_2, x : P_1, \Xi \sqsubseteq \Theta, x : \theta' P_1 \text{ (7)}$$

$$\Theta, x : \theta' P_1 \vdash \theta_1 Q' \leq Q \text{ (8)}$$

$$\theta' \equiv \theta_1 :: \Theta_1, x : P_1 \sqsubseteq \Theta, x : \theta' P_1 \text{ (9)}$$

By (2), (6), and PI-ABS, we have $\Theta_0 [P_0] \vdash_{\text{m}} \lambda x. M \Rightarrow P_1 \rightarrow Q' \vdash \Theta_2, \Xi$ (10). By (7) and metasubstitution weakening, we have $\theta_1 :: \Theta_2, \Xi \sqsubseteq \Theta$ (11). Since precision ignores term-variable entries, (8) also holds under Θ . By (9), (15), (8), and the arrow rule for refinement, we have $\Theta \vdash \theta_1(P_1 \rightarrow Q') \leq P \rightarrow Q$ (12). By (4), inversion on (9), and

transitivity of metasubstitution equivalence, we have $\theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta$ (18). Our goal follows from (10), (11), (12), and (18).

Case

$$\frac{\text{P-ABSANNO} \quad m \neq C \quad \Theta, x : A \vdash_{m-} M \Rightarrow Q \text{ (1)}}{\Theta \vdash_m \lambda x^A.M \Rightarrow A \rightarrow Q}$$

The solution $(\theta :: \Theta_0 \sqsubseteq \Theta; A \rightarrow Q)$ to the skeleton inference problem $(\Theta_0; m; P_0; \lambda x^A.M)$ gives

$$\Theta \vdash \theta P_0 \blacktriangleright A \rightarrow Q \text{ (19)} \quad \text{and} \quad \theta :: \Theta_0 \sqsubseteq \Theta \text{ (20)}.$$

Let

$$\Theta_0 \vdash P_0 \blacktriangleright P_1 \rightarrow Q_1 \dashv \Theta_1 \text{ (2)}.$$

By (19), (20), (2), and definition of splitting, there is a metasubstitution θ' such that

$$\begin{aligned} \theta' :: \Theta_1 \sqsubseteq \Theta & \text{ (3)} \\ \theta \equiv \theta' :: \Theta_0 \sqsubseteq \Theta & \text{ (4)} \\ \Theta \vdash \theta'(P_1 \rightarrow Q_1) \blacktriangleright A \rightarrow Q & \text{ (5)} \end{aligned}$$

By inversion on (5), we have

$$\Theta \vdash \theta' P_1 \blacktriangleright A \text{ (13)} \quad \text{and} \quad \Theta \vdash \theta' Q_1 \blacktriangleright Q \text{ (18)}.$$

By (13) and consistency of A with A , $(\theta' :: \Theta_1 \sqsubseteq \Theta; A)$ is a solution to the consistency problem $(\Theta_1; P_1; A)$. Then by Lemma 26, we have

$$\begin{aligned} \Theta_1 [P_1] \vdash A \sim A' \dashv \Theta_2 & \text{ (14)} \\ \theta'' :: \Theta_2 \sqsubseteq \Theta & \text{ (15)} \\ \Theta \vdash \theta'' A' \leq A & \\ \theta' \equiv \theta'' :: \Theta_1 \sqsubseteq \Theta & \text{ (17)} \end{aligned}$$

By (15), (17), and (18), $(\theta'' :: \Theta_2, x : A \sqsubseteq \Theta, x : A; Q)$ is a solution to the skeleton inference problem $(\Theta_2, x : A; m-; Q_1; M)$. IH on (1) with this solution gives

$$\begin{aligned} \Theta_2, x : A [Q_1] \vdash_{m-} M \Rightarrow Q' \dashv \Theta_3, x : A, \Xi & \text{ (6)} \\ \theta_1 :: \Theta_3, x : A, \Xi \sqsubseteq \Theta, x : A & \text{ (7)} \\ \Theta, x : A \vdash \theta_1 Q' \leq Q & \text{ (8)} \\ \theta'' \equiv \theta_1 :: \Theta_2, x : A \sqsubseteq \Theta, x : A & \text{ (9)} \end{aligned}$$

By (2), (14), (6), and PI-ABSANNO, we have $\Theta_0 [P_0] \vdash_m \lambda x^A.M \Rightarrow A \rightarrow Q' \dashv \Theta_3, \Xi$ (10).

By (7), we have $\theta_1 :: \Theta_3, \Xi \sqsubseteq \Theta$ (11). Since metasubstitutions do not change the annotation A , by (8), we have $\Theta \vdash \theta_1(A \rightarrow Q') \leq A \rightarrow Q$ (12). By (4), (17), (9), and transitivity of metasubstitution equivalence, we have

$$\theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \text{ (21)}.$$

Our goal follows from ⑩, ⑪, ⑫, and ⑮.

Case

$$\frac{\text{P-ABSCHECK} \quad \Theta, x : \mathbb{A} \vdash_c M \Rightarrow Q \text{ ①}}{\Theta \vdash_c \lambda x.M \Rightarrow \mathbb{A}.\mathbb{A} \rightarrow Q}$$

The skeleton inference problem is $(\Theta_0; \mathbb{C}; \mathbb{A}; \lambda x.M)$. ① gives a solution to the problem $(\Theta_0, x : \mathbb{A}; \mathbb{C}; \mathbb{A}; M)$. IH on ① with this solution gives

$$\begin{aligned} \Theta_0, x : \mathbb{A} \vdash_c M &\Rightarrow Q' \vdash \Theta_1, x : \mathbb{A}, \Xi \text{ ②} \\ \theta_1 :: \Theta_1, x : \mathbb{A}, \Xi &\sqsubseteq \Theta, x : \mathbb{A} \text{ ③} \\ \Theta, x : \mathbb{A} \vdash \theta_1 Q' &\leq Q \text{ ④} \\ \theta \equiv \theta_1 :: \Theta_0, x : \mathbb{A} &\sqsubseteq \Theta, x : \mathbb{A} \text{ ⑥} \end{aligned}$$

By PI-ABSCHECK on ②, we have $\Theta_0 \vdash_c \lambda x.M \Rightarrow \mathbb{A}.\mathbb{A} \rightarrow Q' \vdash \Theta_1, \Xi \text{ ⑤}$. By erasing the term variable from ③ and ⑥, we have

$$\theta_1 :: \Theta_1, \Xi \sqsubseteq \Theta \text{ ⑩} \quad \text{and} \quad \theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \text{ ⑪}.$$

Since precision ignores term-variable entries, ④ gives

$$\Theta \vdash \theta_1(\mathbb{A}.\mathbb{A} \rightarrow Q') \leq \mathbb{A}.\mathbb{A} \rightarrow Q \text{ ⑫}.$$

Our goal follows from ⑤, ⑩, ⑫, and ⑪.

Case

$$\frac{\text{P-ABSANNOCHECK} \quad \Theta, x : A \vdash_c M \Rightarrow Q \text{ ①}}{\Theta \vdash_c \lambda x^A.M \Rightarrow \mathbb{A}.A \rightarrow Q}$$

The skeleton inference problem is $(\Theta_0; \mathbb{C}; \mathbb{A}; \lambda x^A.M)$. ① gives a solution to the problem $(\Theta_0, x : A; \mathbb{C}; \mathbb{A}; M)$. IH on ① with this solution gives

$$\begin{aligned} \Theta_0, x : A \vdash_c M &\Rightarrow Q' \vdash \Theta_1, x : A, \Xi \text{ ②} \\ \theta_1 :: \Theta_1, x : A, \Xi &\sqsubseteq \Theta, x : A \text{ ③} \\ \Theta, x : A \vdash \theta_1 Q' &\leq Q \text{ ④} \\ \theta \equiv \theta_1 :: \Theta_0, x : A &\sqsubseteq \Theta, x : A \text{ ⑥} \end{aligned}$$

By PI-ABSANNOCHECK on ②, we have $\Theta_0 \vdash_c \lambda x^A.M \Rightarrow \mathbb{A}.A \rightarrow Q' \vdash \Theta_1, \Xi \text{ ⑤}$. By erasing the term variable from ③ and ⑥, we have

$$\theta_1 :: \Theta_1, \Xi \sqsubseteq \Theta \text{ ⑩} \quad \text{and} \quad \theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \text{ ⑪}.$$

Since precision ignores term-variable entries and annotations do not contain flexible variables, ④ gives

$$\Theta \vdash \theta_1(\mathbb{A}.A \rightarrow Q') \leq \mathbb{A}.A \rightarrow Q \text{ ⑫}.$$

Our goal follows from ⑤, ⑩, ⑫, and ⑪.

Case

$$\begin{array}{c}
 \text{P-APP} \\
 \frac{\Theta \vdash_c N \Rightarrow P \text{ ①} \quad \Theta \vdash P \blacktriangleright P' \text{ ②} \quad \Theta \vdash_{Fm} M \Rightarrow P' \rightarrow Q \text{ ③}}{\Theta \vdash_m M N \Rightarrow Q}
 \end{array}$$

The skeleton inference problem is $(\Theta_0; m; Q_0; M N)$ and the solution is $(\theta :: \Theta_0 \sqsubseteq \Theta; Q)$. The solution gives

$$\Theta \vdash \theta Q_0 \blacktriangleright Q \text{ ⑲} \quad \text{and} \quad \theta :: \Theta_0 \sqsubseteq \Theta \text{ ⑳}.$$

Since $\Theta \vdash \mathbb{A} \blacktriangleright P$, $(\textcircled{20}; P)$ is a solution to $(\Theta_0; c; \mathbb{A}; N)$. IH on ① with this solution gives

$$\begin{array}{l}
 \Theta_0 [\mathbb{A}] \vdash_c N \Rightarrow P_1 \dashv \Theta_1 \text{ ④} \\
 \theta_1 :: \Theta_1 \sqsubseteq \Theta \text{ ⑤} \\
 \Theta \vdash \theta_1 P_1 \leq P \text{ ⑥} \\
 \theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \text{ ⑦}
 \end{array}$$

Lemma 72 on ⑥ and ② gives $\Theta \vdash \theta_1 P_1 \blacktriangleright P' \text{ ⑧}$. By ⑲ and ⑦, we have $\Theta \vdash \theta_1 Q_0 \blacktriangleright Q \text{ ⑨}$. Thus $\Theta \vdash \theta_1 (P_1 \rightarrow Q_0) \blacktriangleright P' \rightarrow Q$. Then by ③, $(\theta_1 :: \Theta_1 \sqsubseteq \Theta; P' \rightarrow Q)$ is a solution to the skeleton inference problem $(\Theta_1; Fm; P_1 \rightarrow Q_0; M)$. IH on ③ with this solution gives

$$\begin{array}{l}
 \Theta_1 [P_1 \rightarrow Q_0] \vdash_{Fm} M \Rightarrow P_2 \rightarrow Q_2 \dashv \Theta_2 \text{ ⑩} \\
 \theta_2 :: \Theta_2 \sqsubseteq \Theta \text{ ⑪} \\
 \Theta \vdash \theta_2 (P_2 \rightarrow Q_2) \leq P' \rightarrow Q \text{ ⑫} \\
 \theta_1 \equiv \theta_2 :: \Theta_1 \sqsubseteq \Theta \text{ ⑬}
 \end{array}$$

By PI-APP on ④ and ⑩, we have $\Theta_0 [Q_0] \vdash_m M N \Rightarrow Q_2 \dashv \Theta_2 \text{ ⑭}$. By inversion on ⑫, we have $\Theta \vdash \theta_2 Q_2 \leq Q \text{ ⑮}$. By ⑦ and ⑬, we have $\theta \equiv \theta_2 :: \Theta_0 \sqsubseteq \Theta \text{ ⑯}$. Our goal follows from ⑭, ⑪, ⑮, and ⑯.

□

D.4 Subtyping

In this section, we prove soundness and completeness of algorithmic subtyping with respect to the declarative subtyping rules.

D.4.1 Decorated algorithmic judgements

We extend the syntax of decorated skeletons ψ , unresolved types T , and unresolved skeletons S with flexible variables $\hat{a}$ for decorated algorithmic judgements.

Figure D.1 defines the decorated algorithmic subtyping $\Psi \mid \Theta_0 [P_0] \vdash_m^n S \leqslant Q \dashv \Theta_1$. Figure D.2 defines the decorated algorithmic consistency $\Psi \mid \Theta_0 [P_0] \vdash S \dot{\sim} Q \dashv \Theta_1$. Like ordinary algorithmic consistency, decorated consistency may carry auxiliary contexts $\Delta_1 \mid \Delta_2$, where Δ_1 tracks bound variables from P_0 and Δ_2 tracks bound variables from S . The tag context Ψ contains precisely the tags solved by the judgement, following the declarative judgements. We omit the rules that remain unchanged from the ordinary algorithmic rules modulo adding the empty tag context $\cdot$ before each judgement.

We define an erasure operation that transforms unresolved skeletons into ordinary skeletons. It is defined homomorphically with the following non-trivial cases.

$$\llbracket \forall^\delta \alpha. S \rrbracket = \forall \alpha. \llbracket S \rrbracket \quad \llbracket \lambda \hat{a} \rrbracket = \lambda a \quad \llbracket \lambda \hat{a}. S \rrbracket = \lambda a. \llbracket S \rrbracket.$$

Erasure commutes with substitution naturally:

$$\llbracket S[\psi/\alpha] \rrbracket = \llbracket S \rrbracket[\llbracket \psi \rrbracket/\alpha].$$

We write

$$\boxed{C; \psi \rightsquigarrow \Psi}$$

for the tag-solving counterpart of constraint solving. The ordinary judgement $C; P \rightsquigarrow Q$ takes an input skeleton P and produces an output skeleton Q . In contrast, $C; \psi \rightsquigarrow \Psi$ takes a decorated candidate ψ and computes the set Ψ of tags in ψ solved by the constraint list C . The definition is contribution-only: constraints add tags only when their own structure matches the corresponding structure in ψ ; constructors already present on the input side of solving are traversed but do not contribute tags.

$$\boxed{\Psi \mid \Theta_0 [P_0]_m^n S \dot{\leq} Q \vdash \Theta_1}$$

$$\begin{array}{c}
\text{SI-INFER} \\
\hline
\cdot \mid \Theta_0 [\mathbb{A}]_1^n S \dot{\leq} [S] \vdash \Theta_0 \\
\\
\text{SI-FORALLR} \\
\frac{\Psi \mid \Theta_0, \alpha [B]_c^{\text{ty}} S \dot{\leq} B \vdash \Theta_1, \alpha, \Xi}{\Psi \mid \Theta_0 [\forall \alpha. B]_c^{\text{ty}} S \dot{\leq} \forall \alpha. B \vdash \Theta_1} \\
\\
\text{SI-ARROW} \\
\frac{\Psi_1 \mid \Theta_0 [P'_1]_1 S_1 \dot{\sim} Q_1 \vdash \Theta_1 \quad \Psi_2 \mid \Theta_1 [P'_2]_m^n S_2 \dot{\leq} Q_2 \vdash \Theta_2}{\Psi_1, \Psi_2 \mid \Theta_0 [P'_1 \rightarrow P'_2]_{f_m}^n S_1 \rightarrow S_2 \dot{\leq} Q_1 \rightarrow Q_2 \vdash \Theta_2} \\
\\
\text{SI-FORALLL} \\
\frac{\begin{array}{l} m \neq \mathbb{I} \quad P_0 \text{ guarded or } P_0 = \mathbb{A} \quad [S] = P \quad P \leq_m^n P_0 \not\vdash^\alpha \Xi; P_1 \quad \Theta_0, \Xi \vdash P_1 \\ \Psi_g \text{ fresh} \quad \Theta_0, \Xi \vdash^n \psi_g \# \Psi_g \quad [\psi_g] = P_1 \quad \Psi, \Psi_g \mid \Theta_0, \Xi [P_0]_m^n S[\psi_g/\alpha] \dot{\leq} Q \vdash \Theta_1 \end{array}}{\Psi \mid \Theta_0 [P_0]_m^n \forall^\delta \alpha. S \dot{\leq} Q \vdash \Theta_1} \\
\\
\text{SI-GHOSTG} \\
\hline
\cdot \mid \Theta_0 [\mathbb{A}]_m^{\text{sk}} \mathbb{A} \dot{\leq} \mathbb{A} \vdash \Theta_0 \\
\\
\text{SI-UNIVGHOSTL} \\
\frac{m \neq \mathbb{I} \quad \Psi \mid \Theta_0 [P_0]_m^{\text{sk}} S \dot{\leq} Q \vdash \Theta_1}{\Psi \mid \Theta_0 [P_0]_m^{\text{sk}} \mathbb{A}. S \dot{\leq} Q \vdash \Theta_1} \\
\\
\text{SI-UNIVGHOSTLG} \\
\frac{m \neq \mathbb{I} \quad \Psi \mid \Theta_0 [P_0]_m^{\text{sk}} S \dot{\leq} Q \vdash \Theta_1}{\Psi \mid \Theta_0 [P_0]_m^{\text{sk}} \mathbb{A}. S \dot{\leq} Q \vdash \Theta_1}
\end{array}$$

Figure D.1: Decorated algorithmic subtyping (only showing non-trivial rules).

$$\boxed{\Psi \mid \Theta_0 [P_0] \vdash_{\Delta_1|\Delta_2} S \dot{\sim} Q \dashv \Theta_1}$$

$$\begin{array}{c}
\text{U-GHOSTG} \\
\hline
\cdot \mid \Theta_0 [\mathbb{A}] \vdash_{\Delta_1|\Delta_2} \mathbb{A} \dot{\sim} \mathbb{A} \dashv \Theta_0 \\
\\
\text{U-ARROW} \\
\frac{\Psi_1 \mid \Theta_0 [P'_1] \vdash_{\Delta_1|\Delta_2} S_1 \dot{\sim} Q_1 \dashv \Theta_1 \quad \Psi_2 \mid \Theta_1 [P'_2] \vdash_{\Delta_1|\Delta_2} S_2 \dot{\sim} Q_2 \dashv \Theta_2}{\Psi_1, \Psi_2 \mid \Theta_0 [P'_1 \rightarrow P'_2] \vdash_{\Delta_1|\Delta_2} S_1 \rightarrow S_2 \dot{\sim} Q_1 \rightarrow Q_2 \dashv \Theta_2} \\
\\
\text{U-FORALLINH} \qquad \text{U-FORALLSYN} \\
\frac{\Psi \mid \Theta_0, \alpha [P_0] \vdash_{\Delta_1, \alpha|\Delta_2, \alpha} S \dot{\sim} Q \dashv \Theta_1, \alpha}{\Psi, \delta \mid \Theta_0 [\forall \alpha. P_0] \vdash_{\Delta_1|\Delta_2} \forall^\delta \alpha. S \dot{\sim} \forall \alpha. Q \dashv \Theta_1} \quad \frac{\Psi \mid \Theta_0, \alpha [\mathbb{A}.P_0] \vdash_{\Delta_1|\Delta_2, \alpha} S \dot{\sim} Q \dashv \Theta_1, \alpha}{\Psi \mid \Theta_0 [\mathbb{A}.P_0] \vdash_{\Delta_1|\Delta_2} \forall^\delta \alpha. S \dot{\sim} \forall \alpha. Q \dashv \Theta_1} \\
\\
\text{U-UNIVGHOST} \qquad \text{U-UNIVGHOSTG} \\
\frac{\Psi \mid \Theta_0 [P_0] \vdash_{\Delta_1|\Delta_2} S \dot{\sim} Q \dashv \Theta_1}{\Psi \mid \Theta_0 [\mathbb{A}.P_0] \vdash_{\Delta_1|\Delta_2} \mathbb{A}.S \dot{\sim} \mathbb{A}.Q \dashv \Theta_1} \quad \frac{\Psi \mid \Theta_0 [P_0] \vdash_{\Delta_1|\Delta_2} S \dot{\sim} Q \dashv \Theta_1}{\Psi \mid \Theta_0 [\mathbb{A}.P_0] \vdash_{\Delta_1|\Delta_2} \mathbb{A}.S \dot{\sim} \mathbb{A}.Q \dashv \Theta_1} \\
\\
\text{U-UNIVGHOST1} \\
\frac{\Psi \mid \Theta_0, \alpha [P_0] \vdash_{\Delta_1, \alpha|\Delta_2} \mathbb{A}.S \dot{\sim} Q \dashv \Theta_1, \alpha}{\Psi \mid \Theta_0 [\forall \alpha. P_0] \vdash_{\Delta_1|\Delta_2} \mathbb{A}.S \dot{\sim} \forall \alpha. Q \dashv \Theta_1} \\
\\
\text{U-UNIVGHOST2} \\
\frac{\Theta_0 \vdash P_0 \text{ guarded} \quad P_0 \text{ non-flex} \quad \Psi \mid \Theta_0 [P_0] \vdash_{\Delta_1|\Delta_2} S \dot{\sim} Q \dashv \Theta_1}{\Psi \mid \Theta_0 [P_0] \vdash_{\Delta_1|\Delta_2} \mathbb{A}.S \dot{\sim} Q \dashv \Theta_1} \\
\\
\text{U-GHOSTR} \\
\frac{\Delta_1 \cap \text{ftv}(\lfloor S \rfloor) = \emptyset}{\cdot \mid \Theta_0 [\mathbb{A}] \vdash_{\Delta_1|\Delta_2} S \dot{\sim} \lfloor S \rfloor \dashv \Theta_0} \\
\\
\text{U-UNIVGHOSTR} \\
\frac{\Theta_0 \vdash S \text{ guarded} \quad S \text{ non-flex} \quad \Psi \mid \Theta_0 [P_0] \vdash_{\Delta_1|\Delta_2} S \dot{\sim} Q \dashv \Theta_1}{\Psi \mid \Theta_0 [\mathbb{A}.P_0] \vdash_{\Delta_1|\Delta_2} S \dot{\sim} Q \dashv \Theta_1}
\end{array}$$

The flexible-variable rules of Figure 9.6 are inherited unchanged and have empty tag context in their conclusions.

Figure D.2: Decorated algorithmic consistency.

$C; \psi \rightsquigarrow \Psi$			
$\frac{}{.; \psi \rightsquigarrow .}$	$\frac{\psi \leq_m P \rightsquigarrow \Psi}{\text{sub}_m(P); \psi \rightsquigarrow \Psi}$	$\frac{\psi \sim P \rightsquigarrow \Psi_1 \quad C; \psi \rightsquigarrow \Psi_2}{\text{eq}(P), C; \psi \rightsquigarrow \Psi_1, \Psi_2}$	
$\psi \leq_m Q \rightsquigarrow \Psi$			
$\frac{\text{SS-VAR}}{\alpha \leq_m Q \rightsquigarrow .}$	$\frac{\text{SS-FORALLL} \quad \psi \leq_m Q \rightsquigarrow \Psi}{\forall^\epsilon \alpha. \psi \leq_m Q \rightsquigarrow \Psi}$	$\frac{\text{SS-UNIVGHOSTLG} \quad \psi \leq_m Q \rightsquigarrow \Psi}{\mathbb{A}. \psi \leq_m Q \rightsquigarrow \Psi}$	$\frac{\text{SS-VACUOUS}}{\psi \leq_m \mathbb{A} \rightsquigarrow .}$
$\frac{\text{SS-CHECK} \quad \psi, B \text{ guarded} \quad \psi \sim B \rightsquigarrow \Psi}{\psi \leq_c B \rightsquigarrow \Psi}$	$\frac{\text{SS-FORALLR} \quad \psi \leq_c B \rightsquigarrow \Psi}{\psi \leq_c \forall \beta. B \rightsquigarrow \Psi}$	$\frac{\text{SS-ARROW} \quad \psi_1 \sim Q_1 \rightsquigarrow \Psi_1 \quad \psi_2 \leq_m Q_2 \rightsquigarrow \Psi_2}{\psi_1 \rightarrow \psi_2 \leq_{fm} Q_1 \rightarrow Q_2 \rightsquigarrow \Psi_1, \Psi_2}$	
$\frac{\text{SS-FLEX}}{\hat{\alpha} \leq_m Q \rightsquigarrow .}$			
$\psi \sim Q \rightsquigarrow \Psi$			
$\frac{\text{SE-UNIT}}{1 \sim 1 \rightsquigarrow .}$	$\frac{\text{SE-VAR}}{\alpha \sim Q \rightsquigarrow .}$	$\frac{\text{SE-VACUOUS}}{\psi \sim \mathbb{A} \rightsquigarrow .}$	$\frac{\text{SE-ARROW} \quad \psi_1 \sim Q_1 \rightsquigarrow \Psi_1 \quad \psi_2 \sim Q_2 \rightsquigarrow \Psi_2}{\psi_1 \rightarrow \psi_2 \sim Q_1 \rightarrow Q_2 \rightsquigarrow \Psi_1, \Psi_2}$
$\frac{\text{SE-FORALLINH} \quad \psi \sim Q \rightsquigarrow \Psi}{\forall^\epsilon \alpha. \psi \sim \forall \alpha. Q \rightsquigarrow \Psi, \epsilon}$	$\frac{\text{SE-UNIVGHOSTG} \quad \psi \sim Q \rightsquigarrow \Psi}{\mathbb{A}. \psi \sim \mathbb{A}. Q \rightsquigarrow \Psi}$	$\frac{\text{SE-FORALLSYN} \quad \psi \sim \mathbb{A}. Q \rightsquigarrow \Psi}{\forall^\epsilon \alpha. \psi \sim \mathbb{A}. Q \rightsquigarrow \Psi}$	
$\frac{\text{SE-UNIVGHOSTR} \quad [\psi] \text{ guarded} \quad \psi \sim Q \rightsquigarrow \Psi}{\psi \sim \mathbb{A}. Q \rightsquigarrow \Psi}$	$\frac{\text{SE-FLEXL}}{\hat{\alpha} \sim Q \rightsquigarrow .}$	$\frac{\text{SE-FLEXR} \quad \mathbb{A} \notin \psi \quad \mathbb{A} \notin \psi}{\psi \sim \hat{\alpha} \rightsquigarrow .}$	

Figure D.3: Decorated tag solving.

D.4.2 Soundness of subtyping

In this section, we prove Theorem 27.

Theorem 27 (Soundness of subtyping). *Given a subtyping problem $(\Theta_0; n; m; P_0; P)$, if $\Theta_0 [P_0]_{\mathfrak{m}}^n P \leq Q \dashv \Theta_1$, then $(\Theta_0 \sqsubseteq \Theta_1; Q)$ is a solution.*

We use decorated algorithmic subtyping as a bridge between algorithmic subtyping and declarative subtyping. We first prove an important lemma that given an algorithmic subtyping derivation, we can construct a decorated algorithmic subtyping derivation under the empty tag context.

Lemma 74 (Decorating algorithmic subtyping). *If $\Theta_0 [P_0]_{\mathfrak{m}}^n P \leq Q \dashv \Theta_1$, then $\cdot \mid \Theta_0 [P_0]_{\mathfrak{m}}^n P \dot{\leq} Q \dashv \Theta_1$.*

Then, we prove another lemma on the soundness of decorated algorithmic subtyping. It also records the output invariants needed by the definition of solutions to subtyping problems.

Lemma 75 (Soundness of decorated subtyping). *If $\Psi \mid \Theta_0 [P_0]_{\mathfrak{m}}^n S \dot{\leq} Q \dashv \Theta_1$, then*

$$\Theta_0 \sqsubseteq \Theta_1 \quad \Theta_1 \vdash P_0 \blacktriangleright Q \quad \Psi \mid \Theta_1 \vdash_{\mathfrak{m}}^n S \leq Q.$$

Consequently, given a problem of subtyping $(\Theta_0; n; m; P_0; P)$, if $\cdot \mid \Theta_0 [P_0]_{\mathfrak{m}}^n P \dot{\leq} Q \dashv \Theta_1$, then $(\Theta_0 \sqsubseteq \Theta_1; Q)$ is a solution to it.

Combining Lemma 74 and Lemma 75, we prove Theorem 27.

The main challenge of proving Lemma 74 is to show that when we do instantiation, all tags introduced by the type used for instantiation are solved. To prove this lemma, we first prove a few auxiliary lemmas about decorated algorithmic rules.

Lemma 76 (Look decoration). *If $P \leq_{\mathfrak{m}}^n P_0 \not\leq^{\alpha} \Xi; Q$ and $\Theta_0 \vdash Q$, then there are fresh tag set Ψ and decorated skeleton ψ such that $\Theta_0 \vdash^n \psi \# \Psi$ and $\lfloor \psi \rfloor = Q$.*

Proof. Construct ψ by decorating every quantifier in Q with a fresh tag (taking Ψ to be the set of exactly these tags), transforming every ghost $\mathfrak{A}$ into a dotted ghost $\mathfrak{A}$, and transforming every universal ghost $\mathfrak{A}$ into a dotted universal ghost $\mathfrak{A}$. The judgement $\Theta_0 \vdash^n \psi \# \Psi$ follows from straightforward induction on well-formedness of Q . $\square$

For the proofs below, we write $\text{guess}^n(P)$ for the sort-indexed meta-function defined by

$$\text{guess}^{\text{ty}}(P) = \text{guess}(P) \quad \text{guess}^{\text{sk}}(P) = (\cdot; P).$$

Lemma 77 (Decorating constraint solving). *If*

$$P \leq_m^n P_0 \not\leq^\alpha \Xi; P_a \text{ (1)} \quad \Theta_0, \Xi \vdash^n \psi_a \# \Psi_a \quad \llbracket \psi_a \rrbracket = P_a,$$

then there are C and Ξ_c such that

$$P \leq_m P_0 \not\leq^\alpha \Xi_c; C \quad C; \psi_a \rightsquigarrow \Psi_a.$$

Proof. Inversion on (1) gives

$$P \leq_m P_0 \not\leq^\alpha \Xi_c; C \quad C; \mathbb{L} \rightsquigarrow \Xi_s; P_s \text{ (2)} \quad (\Xi'; P_a) = \text{guess}^n(P_s) \quad \Xi = \Xi_c, \Xi_s, \Xi'$$

The decorated constraint solving $C; \psi_a \rightsquigarrow \Psi_a$ can be constructed by a straightforward induction on constraint solving (2), observing that every tagged quantifier in ψ_a comes from a quantifier in P_s , and every dotted ghost in ψ_a (when $n = \text{sk}$) comes from a ghost in P_s . $\square$

Lemma 78 (Constraint solving concatenation). *If $C_1, C_2; \psi \rightsquigarrow \Psi$, then there are Ψ_1 and Ψ_2 such that*

$$C_1; \psi \rightsquigarrow \Psi_1 \quad C_2; \psi \rightsquigarrow \Psi_2 \quad \Psi = \Psi_1, \Psi_2.$$

Conversely, if

$$C_1; \psi \rightsquigarrow \Psi_1 \quad C_2; \psi \rightsquigarrow \Psi_2,$$

then $C_1, C_2; \psi \rightsquigarrow \Psi_1, \Psi_2$.

Proof. Both directions follow by straightforward induction on the constraint list C_1 . $\square$

We use the following convention in the proofs below. The constraint-collection and decorated-constraint-solving judgements are not syntax-directed: several rules may overlap, but they are congruent. When a proof says WLOG, we use the congruence of these judgements to reorder such exposing steps so that the rule being decorated is applied first. Non-congruent overlaps are excluded by side conditions; for instance, LS-CHECK does not apply when the source being inspected is the variable being looked for.

Lemma 79 (Subtyping decoration for variables). *Suppose*

$$\psi \leq_m P_0 \rightsquigarrow \Psi_a$$

For any substitution ρ from rigid type variables to decorated skeletons, if

$$\Psi \mid \Theta_0 \llbracket P_0 \rrbracket_m^n \rho(\llbracket \psi \rrbracket) \leq Q \vdash \Theta_1,$$

then

$$\Psi, \Psi_a \mid \Theta_0 [P_0] \vdash_m^n \rho(\psi) \dot{\leq} Q \dashv \Theta_1.$$

Proof. Induction on the decorated constraint solving derivation $\psi \leq_m P_0 \rightsquigarrow \Psi_a$, generalising over Θ_0, Ψ, ρ , and the decorated subtyping premise.

Case

SS-VAR

$$\frac{}{\alpha \leq_m P_0 \rightsquigarrow \cdot}$$

The subtyping premise is

$$\Psi \mid \Theta_0 [P_0] \vdash_m^n \rho(\alpha) \dot{\leq} Q \dashv \Theta_1 \text{ ①}.$$

Since $\Psi_a = \cdot$, the goal follows from ①.

Case

SS-FORALL

$$\frac{\psi \leq_m P_0 \rightsquigarrow \Psi_a \text{ ①}}{\forall^\epsilon \beta. \psi \leq_m P_0 \rightsquigarrow \Psi_a}$$

Assume WLOG that $m \neq \text{I}$ and P_0 guarded; otherwise the decorated constraint solving derivation is equivalent to applying other rules first. The subtyping premise is equivalently

$$\Psi \mid \Theta_0 [P_0] \vdash_m^n \forall \beta. \rho(\lfloor \psi \rfloor) \dot{\leq} Q \dashv \Theta_1 \text{ ②}.$$

Inversion on ② with decorated SI-FORALL gives

$$\begin{aligned} \rho(\lfloor \psi \rfloor) &\leq_m^n P_0 \not\vdash^\beta \Xi'; P_g \text{ ③} & \Psi_g \text{ fresh ⑩} \\ \Theta_0, \Xi' \vdash^n \psi_g \# \Psi_g \text{ ④} & & \lfloor \psi_g \rfloor = P_g \text{ ⑤} \end{aligned}$$

The same inversion also gives

$$\Psi, \Psi_g \mid \Theta_0, \Xi' [P_0] \vdash_m^n \rho(\lfloor \psi \rfloor)[\psi_g/\beta] \dot{\leq} Q \dashv \Theta_1 \text{ ⑥}.$$

Let $\rho' = \rho[\psi_g/\beta]$. We have

$$\rho'(\lfloor \psi \rfloor) = \rho(\lfloor \psi \rfloor)[\psi_g/\beta] \text{ ⑦} \quad \rho'(\psi) = \rho(\psi)[\psi_g/\beta] \text{ ⑧}.$$

By IH on ①, ρ' , ⑦, ⑧, and ⑥, we get

$$\Psi, \Psi_g, \Psi_a \mid \Theta_0, \Xi' [P_0] \vdash_m^n \rho(\psi)[\psi_g/\beta] \dot{\leq} Q \dashv \Theta_1 \text{ ⑨}.$$

By ④ and ⑤, we also have

$$\Theta_0, \Xi' \vdash P_g \text{ ⑪}.$$

Decorated SI-FORALL on ③, ⑪, ⑩, ④, ⑤, and ⑨ gives

$$\Psi, \Psi_a \mid \Theta_0 [P_0] \vdash_m^n \forall^\epsilon \beta. \rho(\psi) \dot{\leq} Q \dashv \Theta_1 \text{ ⑫}.$$

Our goal follows from ⑫ and $\forall^\epsilon \beta. \rho(\psi) = \rho(\forall^\epsilon \beta. \psi)$.

Case

$$\frac{\text{SS-UNIVGHOSTLG} \quad \psi \leq_m P_0 \rightsquigarrow \Psi_a \text{ ①}}{\mathbb{A}. \psi \leq_m P_0 \rightsquigarrow \Psi_a}$$

Assume WLOG that $m \neq 1$. The subtyping premise is equivalently

$$\Psi \mid \Theta_0 [P_0] \vdash_m^{\text{sk}} \mathbb{A}. \rho(\lfloor \psi \rfloor) \dot{\leq} Q \vdash \Theta_1 \text{ ②}.$$

Inversion on ② with decorated SI-UNIVGHOSTL gives

$$\Psi \mid \Theta_0 [P_0] \vdash_m^{\text{sk}} \rho(\lfloor \psi \rfloor) \dot{\leq} Q \vdash \Theta_1 \text{ ③}.$$

By IH on ①, ρ , and ③, we get

$$\Psi, \Psi_a \mid \Theta_0 [P_0] \vdash_m^{\text{sk}} \rho(\psi) \dot{\leq} Q \vdash \Theta_1 \text{ ④}.$$

Our goal follows from decorated SI-UNIVGHOSTLG on ④.

Case

SS-VACUOUS

$$\frac{}{\psi \leq_m \mathbb{A} \rightsquigarrow \cdot}$$

We have $P_0 = \mathbb{A}$. The subtyping premise is

$$\Psi \mid \Theta_0 [\mathbb{A}] \vdash_m^n \rho(\lfloor \psi \rfloor) \dot{\leq} Q \vdash \Theta_1 \text{ ①}.$$

By Lemma 80, with $S = \alpha$ and the same ρ , on ①, we get

$$\Psi \mid \Theta_0 [\mathbb{A}] \vdash_m^n \rho(\psi) \dot{\leq} Q \vdash \Theta_1 \text{ ②}.$$

Since $\Psi_a = \cdot$, our goal follows from ②.

Case

$$\frac{\text{SS-CHECK} \quad \psi, B \text{ guarded} \quad \psi \sim B \rightsquigarrow \Psi_a \text{ ①}}{\psi \leq_c B \rightsquigarrow \Psi_a}$$

The subtyping premise has the form

$$\Psi \mid \Theta_0 [B] \vdash_c^{\text{ty}} \rho(\lfloor \psi \rfloor) \dot{\leq} Q \vdash \Theta_1 \text{ ②}.$$

Case analysis on the shape of $\rho(\lfloor \psi \rfloor)$.

Subcase $\rho(\lfloor \psi \rfloor)$ is guarded. Inversion on ② with decorated SI-CHECK gives $Q = B$ and

$$\Psi \mid \Theta_0 [B] \vdash_c^{\text{ty}} \rho(\lfloor \psi \rfloor) \dot{\sim} Q_c \vdash \Theta_1 \text{ ③}.$$

By Lemma 95 on ①, ρ , and ③, we have

$$\Psi, \Psi_a \mid \Theta_0 [B] \vdash \rho(\psi) \dot{\sim} Q_c \dashv \Theta_1 \text{ ④}.$$

Our goal follows from decorated SI-CHECK on ④ and $Q = B$.

Subcase $\rho(\lfloor \psi \rfloor)$ is unguarded. In this case, ψ itself is a type variable that is mapped to an unguarded skeleton by ρ . Thus we have $\Psi_a = \cdot$ and $\rho(\psi) = \rho(\lfloor \psi \rfloor)$. Our goal directly follows from ②.

Case

$$\begin{array}{c} \text{SS-FORALLR} \\ \frac{\psi \leq_c B \rightsquigarrow \Psi_a \text{ ①}}{\psi \leq_c \forall \beta. B \rightsquigarrow \Psi_a} \end{array}$$

The subtyping premise is

$$\Psi \mid \Theta_0 [\forall \beta. B] \vdash_c^{\text{ty}} \rho(\lfloor \psi \rfloor) \dot{\leq} Q \dashv \Theta_1 \text{ ②}.$$

By inversion on ② with decorated SI-FORALLR, we have

$$\Psi \mid \Theta_0, \beta [B] \vdash_c^{\text{ty}} \rho(\lfloor \psi \rfloor) \dot{\leq} B \dashv \Theta_1, \beta, \Xi' \text{ ③}.$$

By IH on ①, ρ , and ③, we get

$$\Psi, \Psi_a \mid \Theta_0, \beta [B] \vdash_c^{\text{ty}} \rho(\psi) \dot{\leq} B \dashv \Theta_1, \beta, \Xi' \text{ ④}.$$

Our goal follows from decorated SI-FORALLR on ④.

Case

$$\begin{array}{c} \text{SS-ARROW} \\ \frac{\psi_1 \sim P_{0,1} \rightsquigarrow \Psi_{a,1} \text{ ①} \quad \psi_2 \leq_m P_{0,2} \rightsquigarrow \Psi_{a,2} \text{ ②}}{\psi_1 \rightarrow \psi_2 \leq_{\text{fm}} P_{0,1} \rightarrow P_{0,2} \rightsquigarrow \Psi_{a,1}, \Psi_{a,2}} \end{array}$$

The subtyping premise is

$$\Psi \mid \Theta_0 [P_{0,1} \rightarrow P_{0,2}] \vdash_{\text{fm}}^n \rho(\lfloor \psi_1 \rfloor) \rightarrow \rho(\lfloor \psi_2 \rfloor) \dot{\leq} Q \dashv \Theta_1 \text{ ③}.$$

By inversion on ③ with decorated SI-ARROW, there are Q_1 , Q_2 , Θ' , Ψ_1 , and Ψ_2 such that

$$\Psi_1 \mid \Theta_0 [P_{0,1}] \vdash \rho(\lfloor \psi_1 \rfloor) \dot{\sim} Q_1 \dashv \Theta' \text{ ④} \quad \Psi_2 \mid \Theta' [P_{0,2}] \vdash_m^n \rho(\lfloor \psi_2 \rfloor) \dot{\leq} Q_2 \dashv \Theta_1 \text{ ⑤}$$

and

$$Q = Q_1 \rightarrow Q_2 \text{ ⑥} \quad \Psi = \Psi_1, \Psi_2 \text{ ⑦}.$$

By Lemma 95 on ①, ρ , and ④, we have

$$\Psi_1, \Psi_{a,1} \mid \Theta_0 [P_{0,1}] \vdash \rho(\psi_1) \dot{\sim} Q_1 \dashv \Theta' \text{ ⑧}.$$

By IH on ②, ρ , and ⑤, we get

$$\Psi_2, \Psi_{a,2} \mid \Theta' [P_{0,2}] \vdash_m^n \rho(\psi_2) \dot{\leq} Q_2 \dashv \Theta_1 \text{ ⑨}.$$

Decorated SI-ARROW on ⑧ and ⑨ gives

$$\Psi_1, \Psi_{a,1}, \Psi_2, \Psi_{a,2} \mid \Theta_0 [P_{0,1} \rightarrow P_{0,2}] \vdash_{\text{Fm}}^n \rho(\psi_1) \rightarrow \rho(\psi_2) \dot{\leq} Q_1 \rightarrow Q_2 \dashv \Theta_1 \text{ ⑩}.$$

Our goal follows from ⑩, ⑥, and ⑦.

Case

SS-FLEX

$$\frac{}{\hat{a} \leq_m P_0 \rightsquigarrow \cdot}$$

The subtyping premise is

$$\Psi \mid \Theta_0 [P_0] \vdash_{\text{m}}^n \rho(\hat{a}) \dot{\leq} Q \dashv \Theta_1 \text{ ①}.$$

Since $\Psi_a = \cdot$, our goal follows from ①.

□

Lemma 80 (Vacuous subtyping decoration). *For any substitution ρ from rigid type variables to decorated skeletons, if*

$$\Psi \mid \Theta [\mathbb{A}] \vdash_{\text{m}}^n \rho(S[\llbracket \psi \rrbracket / \alpha]) \dot{\leq} Q \dashv \Theta',$$

then

$$\Psi \mid \Theta [\mathbb{A}] \vdash_{\text{m}}^n \rho(S[\psi / \alpha]) \dot{\leq} Q \dashv \Theta'.$$

Proof. By straightforward induction on the decorated subtyping derivation. Observe that since the input skeleton is $\mathbb{A}$, we can never solve any tags, which additionally gives $\Psi = \cdot$. □

Lemma 81 (Stability of constraint collection). *If $\beta \neq \alpha$ and $\alpha \notin \text{ftv}(Q)$, then*

$$P \leq_m P_0 \not\leq^\alpha \Xi; C \quad \text{iff} \quad P[Q/\beta] \leq_m P_0 \not\leq^\alpha \Xi; C.$$

Proof. By straightforward induction on the constraint-collection derivation in both directions. Observe that since $\alpha \notin \text{ftv}(Q)$, we can always use LS-ABSENT or LE-ABSENT for Q . □

Lemma 82 (Look freshness). *If $P \leq_m^n P_0 \not\leq^\beta \Xi; P_a$ and $\gamma \notin \text{ftv}(P_0)$ and $\gamma = \beta$ or $\gamma \notin \text{ftv}(P)$, then $\gamma \notin \text{ftv}(P_a)$.*

Proof. Trivial. □

The following is a key lemma. It shows that we can replace the instantiation with an ordinary skeleton $\llbracket \psi_a \rrbracket$ to use its decorated version ψ_a , so that subtyping solves the same set of tags Ψ_a of ψ_a as decorated constraint solving does.

Lemma 83 (Subtyping decoration). *Suppose*

$$P \leq_m P_0 \not\prec^\alpha \Xi_c; C \quad C; \psi_a \rightsquigarrow \Psi_a \quad [S] = P \quad \alpha \notin \text{ftv}([\psi_a]).$$

If

$$\Psi \mid \Theta_0 [P_0] \vdash_m^n S[\psi_a] / \alpha \dot{\leq} Q \dashv \Theta_1,$$

then

$$\Psi, \Psi_a \mid \Theta_0 [P_0] \vdash_m^n S[\psi_a] / \alpha \dot{\leq} Q \dashv \Theta_1.$$

Proof. Induction on the constraint-collection derivation $P \leq_m P_0 \not\prec^\alpha \Xi_c; C$, generalising over S, Ψ, Θ_0 , and the decorated subtyping premise. We allow arbitrary input context Θ_0 as long as the decorated subtyping premise holds. As a result, the well-formedness condition of the subtyping premise implicitly requires that flexible variables in Ξ_c must be present in Θ_0 if α appears in S .

Case

LS-VAR

$$\frac{}{\alpha \leq_m P_0 \not\prec^\alpha \text{sub}_m(P_0)}$$

The decorated constraint solving premise is

$$\text{sub}_m(P_0); \psi_a \rightsquigarrow \Psi_a \text{ (4)}$$

By inversion on (4), we have

$$\psi_a \leq_m P_0 \rightsquigarrow \Psi_a \text{ (5)}.$$

Since $[S] = \alpha$, we have $S = \alpha$. Thus the premise subtyping derivation is

$$\Psi \mid \Theta_0 [P_0] \vdash_m^n [\psi_a] \dot{\leq} Q \dashv \Theta_1 \text{ (2)}.$$

Our goal follows from Lemma 79 on (5) and (2):

$$\Psi, \Psi_a \mid \Theta_0 [P_0] \vdash_m^n \psi_a \dot{\leq} Q \dashv \Theta_1 \text{ (3)}.$$

Case

LS-ABSENT

$$\frac{\alpha \notin \text{ftv}(P)}{P \leq_m P_0 \not\prec^\alpha .}$$

The decorated constraint solving premise is

$$.; \psi_a \rightsquigarrow . \text{ (3)}$$

and the subtyping premise is

$$\Psi \mid \Theta_0 [P_0] \vdash_m^n S[\psi_a] / \alpha \dot{\leq} Q \dashv \Theta_1 \text{ (4)}.$$

Since $\lfloor S \rfloor = P$ and $\alpha \notin \text{ftv}(P)$, we have $\alpha \notin \text{ftv}(S)$. Hence

$$S[\lfloor \psi_a \rfloor / \alpha] = S[\psi_a / \alpha] \text{ (2)}.$$

Our goal follows from (4) by (2).

Case

LS-VACUOUS

$$\frac{}{P \leq_m \mathbb{A} \not\prec^\alpha}.$$

The decorated constraint solving premise is

$$\cdot; \psi_a \rightsquigarrow \cdot \text{ (3)}$$

and the subtyping premise is

$$\Psi \mid \Theta_0 \lfloor \mathbb{A} \rfloor_m^n S[\lfloor \psi_a \rfloor / \alpha] \dot{\leq} Q \dashv \Theta_1 \text{ (4)}.$$

By Lemma 80, with the identity substitution, on (4), we get

$$\Psi \mid \Theta_0 \lfloor \mathbb{A} \rfloor_m^n S[\psi_a / \alpha] \dot{\leq} Q \dashv \Theta_1 \text{ (2)}.$$

Our goal follows from (2).

Case

LS-CHECK

$$\frac{A \neq \alpha \quad A, B \text{ guarded} \quad A \sim B \not\prec^\alpha \Xi_c; C \text{ (1)}}{A \leq_c B \not\prec^\alpha \Xi_c; C}$$

The decorated constraint solving premise and the subtyping premise are

$$C; \psi_a \rightsquigarrow \Psi_a \text{ (2)} \quad \Psi \mid \Theta_0 [B]_c^{\text{ty}} S[\lfloor \psi_a \rfloor / \alpha] \dot{\leq} B \dashv \Theta_1 \text{ (3)}.$$

Since $\lfloor S \rfloor = A$, $A \neq \alpha$, and A is guarded, $S[\lfloor \psi_a \rfloor / \alpha]$ is guarded. Thus inversion on (3) gives that for a hidden Q_c ,

$$\Psi \mid \Theta_0 [B]_c^{\text{ty}} S[\lfloor \psi_a \rfloor / \alpha] \dot{\sim} Q_c \dashv \Theta_1 \text{ (4)}$$

By Lemma 94 on (1), (2), $\lfloor S \rfloor = A$, $\alpha \notin \text{ftv}(\lfloor \psi_a \rfloor)$, and (4), we have

$$\Psi, \Psi_a \mid \Theta_0 [B]_c^{\text{ty}} S[\psi_a / \alpha] \dot{\sim} Q_c \dashv \Theta_1 \text{ (6)}.$$

Our goal follows from decorated SI-CHECK on (6).

Case

LS-FORALLR

$$\frac{A \leq_c B \not\prec^\alpha \Xi_c; C \text{ (1)}}{A \leq_c \forall \beta. B \not\prec^\alpha \Xi_c; C}$$

The decorated constraint solving premise and the subtyping premise are

$$C; \psi_a \rightsquigarrow \Psi_a \text{ (2)} \quad \Psi \mid \Theta_0 [\forall \beta. B]_c^{\text{ty}} S[\lfloor \psi_a \rfloor / \alpha] \dot{\leq} \forall \beta. B \dashv \Theta_1 \text{ (3)}.$$

Inversion on ③ gives

$$\Psi \mid \Theta_0, \beta \vdash_c^{\text{ty}} S[\llbracket \psi_a \rrbracket / \alpha] \dot{\leq} B \vdash \Theta_1, \beta, \Xi \text{ ④}$$

By IH on ①, ②, $\llbracket S \rrbracket = A$, $\alpha \notin \text{ftv}(\llbracket \psi_a \rrbracket)$, and ④, we obtain

$$\Psi, \Psi_a \mid \Theta_0, \beta \vdash_c^{\text{ty}} S[\psi_a / \alpha] \dot{\leq} B \vdash \Theta_1, \beta, \Xi \text{ ⑧}.$$

Our goal follows from decorated SI-FORALLR on ⑧.

Case

$$\begin{array}{c} \text{LS-ARROW} \\ \frac{P_1 \sim P'_1 \not\leq^\alpha \Xi_1; C_1 \text{ ①} \quad P_2 \leq_m P'_2 \not\leq^\alpha \Xi_2; C_2 \text{ ②}}{P_1 \rightarrow P_2 \leq_{\text{fm}} P'_1 \rightarrow P'_2 \not\leq^\alpha \Xi_1, \Xi_2; C_1, C_2} \end{array}$$

The decorated constraint solving premise and the subtyping premise are

$$C_1, C_2; \psi_a \rightsquigarrow \Psi_a \text{ ⑤} \quad \Psi \mid \Theta_0 [P'_1 \rightarrow P'_2] \vdash_{\text{fm}}^n S[\llbracket \psi_a \rrbracket / \alpha] \dot{\leq} Q_1 \rightarrow Q_2 \vdash \Theta_1 \text{ ⑩}$$

Since $\llbracket S \rrbracket = P_1 \rightarrow P_2$, we have $S = S_1 \rightarrow S_2$ with

$$\llbracket S_1 \rrbracket = P_1 \text{ ③} \quad \llbracket S_2 \rrbracket = P_2 \text{ ④}.$$

By Lemma 78 on ⑤, there are $\Psi_{a,1}$ and $\Psi_{a,2}$ such that

$$C_1; \psi_a \rightsquigarrow \Psi_{a,1} \text{ ⑥} \quad C_2; \psi_a \rightsquigarrow \Psi_{a,2} \text{ ⑦} \quad \Psi_a = \Psi_{a,1}, \Psi_{a,2} \text{ ⑧}.$$

By inversion on ⑩, there exist Ψ_1, Ψ_2 , and Θ' such that

$$\begin{array}{l} \Psi_1 \mid \Theta_0 [P'_1] \vdash S_1[\llbracket \psi_a \rrbracket / \alpha] \dot{\sim} Q_1 \vdash \Theta' \text{ ⑨} \\ \Psi_2 \mid \Theta' [P'_2] \vdash_m^n S_2[\llbracket \psi_a \rrbracket / \alpha] \dot{\leq} Q_2 \vdash \Theta_1 \text{ ⑪} \\ \Psi = \Psi_1, \Psi_2 \text{ ⑫}. \end{array}$$

By Lemma 94 on ①, ⑥, ③, $\alpha \notin \text{ftv}(\llbracket \psi_a \rrbracket)$, and ⑨, we get

$$\Psi_1, \Psi_{a,1} \mid \Theta_0 [P'_1] \vdash S_1[\psi_a / \alpha] \dot{\sim} Q_1 \vdash \Theta' \text{ ⑮}.$$

By IH on ②, ⑦, ④, $\alpha \notin \text{ftv}(\llbracket \psi_a \rrbracket)$, and ⑪, we get

$$\Psi_2, \Psi_{a,2} \mid \Theta' [P'_2] \vdash_m^n S_2[\psi_a / \alpha] \dot{\leq} Q_2 \vdash \Theta_1 \text{ ⑯}.$$

Decorated SI-ARROW on ⑮ and ⑯ gives a derivation

$$\Psi_1, \Psi_{a,1}, \Psi_2, \Psi_{a,2} \mid \Theta_0 [P'_1 \rightarrow P'_2] \vdash_{\text{fm}}^n S_1[\psi_a / \alpha] \rightarrow S_2[\psi_a / \alpha] \dot{\leq} Q_1 \rightarrow Q_2 \vdash \Theta_1 \text{ ⑰}.$$

The tag context $\Psi_1, \Psi_{a,1}, \Psi_2, \Psi_{a,2}$ is equal to Ψ, Ψ_a by ⑧ and ⑫. Our goal follows from ⑰.

Case

$$\begin{array}{c} \text{LS-FORALL} \\ \frac{P \leq_m P_0 \not\leq^\alpha \Xi_c; C \text{ ①}}{\forall \beta. P \leq_m P_0 \not\leq^\alpha \Xi_c; C} \end{array}$$

Assume WLOG that $m \neq 1$ and P_0 is guarded; otherwise derivation is equivalent to applying LS-VACUOUS or LS-FORALLR first.

Since $\lfloor S \rfloor = \forall \beta. P$, $S = \forall \delta \beta. S'$ for some potential tag δ with $\lfloor S' \rfloor = P$ ②.

The decorated constraint solving premise and the subtyping premise are

$$C; \psi_a \rightsquigarrow \Psi_a \text{ ③} \quad \Psi \mid \Theta_0 [P_0]_m^n S[\lfloor \psi_a \rfloor / \alpha] \dot{\leq} Q \dashv \Theta_1 \text{ ⑮}$$

Inversion on ⑮ with SI-FORALLR gives a skeleton P_g , a decorated skeleton ψ_g , and tag set Ψ_g such that

$$P[\lfloor \psi_a \rfloor / \alpha] \leq_m^n P_0 \not\leq^\beta \Xi'; P_g \text{ ④} \quad \Psi_g \text{ fresh ⑯} \quad \Theta_0, \Xi' \vdash^n \psi_g \# \Psi_g \text{ ⑤} \quad \lfloor \psi_g \rfloor = P_g \text{ ⑥}$$

By ⑤ and ⑥, we also have

$$\Theta_0, \Xi' \vdash P_g \text{ ⑰}.$$

The same inversion also gives the subtyping judgement

$$\Psi, \Psi_g \mid \Theta_0, \Xi' [P_0]_m^n (S'[\lfloor \psi_a \rfloor / \alpha])[\psi_g / \beta] \dot{\leq} Q \dashv \Theta_1 \text{ ⑦}.$$

Since $\alpha \notin \text{ftv}(\lfloor \psi_a \rfloor)$, we have $\alpha \notin \text{ftv}(P[\lfloor \psi_a \rfloor / \alpha])$. Then by Lemma 82 on ④ and $\alpha \notin \text{ftv}(P_0)$, we have $\alpha \notin \text{ftv}(P_g)$ ⑧. By ⑥ and ⑧, we have $\alpha \notin \text{ftv}(\lfloor \psi_g \rfloor)$ ⑨. By erasure preserving free variables, we also have $\alpha \notin \text{ftv}(\psi_g)$.

By capture-avoiding substitution, the subtyping premise is equivalently

$$\Psi, \Psi_g \mid \Theta_0, \Xi' [P_0]_m^n (S'[\psi_g / \beta])[\lfloor \psi_a \rfloor / \alpha] \dot{\leq} Q \dashv \Theta_1 \text{ ⑩}.$$

By Lemma 81 on ① and ⑨, we have

$$P[\lfloor \psi_g \rfloor / \beta] \leq_m P_0 \not\leq^\alpha \Xi_c; C \text{ ⑪}.$$

By definition of erasure,

$$\lfloor S'[\psi_g / \beta] \rfloor = P[\lfloor \psi_g \rfloor / \beta] \text{ ⑫}.$$

By IH on ⑪, ③, ⑩, ⑫, and $\alpha \notin \text{ftv}(\lfloor \psi_a \rfloor)$, we get

$$\Psi, \Psi_g, \Psi_a \mid \Theta_0, \Xi' [P_0]_m^n (S'[\psi_g / \beta])[\psi_a / \alpha] \dot{\leq} Q \dashv \Theta_1 \text{ ⑬}.$$

Commuting substitutions back gives

$$\Psi, \Psi_g, \Psi_a \mid \Theta_0, \Xi' [P_0]_m^n (S'[\psi_a / \alpha])[\psi_g / \beta] \dot{\leq} Q \dashv \Theta_1 \text{ ⑭}.$$

Our goal follows from decorated SI-FORALLR on ④, ⑰, ⑯, ⑤, ⑥, and ⑭, using associativity and commutativity of set join.

Case

$$\begin{array}{c} \text{LS-UNIVGHOSTL} \\ P \leq_m P_0 \not\leq^\alpha \Xi_c; C \text{ ①} \\ \hline \mathbb{R}.P \leq_m P_0 \not\leq^\alpha \Xi_c; C \end{array}$$

Assume WLOG that $m \neq 1$. The decorated constraint solving premise and the subtyping premise are

$$C; \psi_a \rightsquigarrow \Psi_a \text{ (5)} \quad \Psi \mid \Theta_0 [P_0]_m^{sk} S[\llbracket \psi_a \rrbracket / \alpha] \dot{\leq} Q \dashv \Theta_1 \text{ (6)}$$

Since $\llbracket S \rrbracket = \mathbb{A}.P$, either $S = \mathbb{A}.S'$ or $S = \mathbb{A}.S'$ with $\llbracket S' \rrbracket = P$ (2).

Inverting (6) with decorated SI-UNIVGHOSTL or SI-UNIVGHOSTLG, respectively, gives

$$\Psi \mid \Theta_0 [P_0]_m^{sk} S'[\llbracket \psi_a \rrbracket / \alpha] \dot{\leq} Q \dashv \Theta_1 \text{ (3)}.$$

By IH on (1), (5), (2), $\alpha \notin \text{ftv}(\llbracket \psi_a \rrbracket)$, and (3), we have

$$\Psi, \Psi_a \mid \Theta_0 [P_0]_m^{sk} S'[\psi_a / \alpha] \dot{\leq} Q \dashv \Theta_1 \text{ (4)}.$$

Our goal follows from decorated SI-UNIVGHOSTL or SI-UNIVGHOSTLG, respectively, on (4).

□

Lemma 84 (Decorating instantiation). *Suppose*

$$P \leq_m^n P_0 \not\prec^\alpha \Xi; P_a \quad \llbracket S \rrbracket = P \quad \Theta_0, \Xi \vdash^n \psi_a \# \Psi_a \quad \llbracket \psi_a \rrbracket = P_a.$$

If

$$\Psi \mid \Theta_0, \Xi [P_0]_m^n S[P_a / \alpha] \dot{\leq} Q \dashv \Theta_1,$$

then

$$\Psi, \Psi_a \mid \Theta_0, \Xi [P_0]_m^n S[\psi_a / \alpha] \dot{\leq} Q \dashv \Theta_1.$$

Proof. By Lemma 77 on some premises, there are C and Ξ_c such that

$$P \leq_m P_0 \not\prec^\alpha \Xi_c; C \text{ (1)} \quad C; \psi_a \rightsquigarrow \Psi_a \text{ (2)}.$$

By Lemma 82 on the look premise and $\alpha \notin \text{ftv}(P_0)$, we have $\alpha \notin \text{ftv}(P_a)$ (4). By $\llbracket \psi_a \rrbracket = P_a$ and (4), we have $\alpha \notin \text{ftv}(\llbracket \psi_a \rrbracket)$ (5). By $\llbracket \psi_a \rrbracket = P_a$, the premise subtyping derivation is

$$\Psi \mid \Theta_0, \Xi [P_0]_m^n S[\llbracket \psi_a \rrbracket / \alpha] \dot{\leq} Q \dashv \Theta_1 \text{ (6)}.$$

By Lemma 83 on (1), (2), $\llbracket S \rrbracket = P$, (5), and (6), we have

$$\Psi, \Psi_a \mid \Theta_0, \Xi [P_0]_m^n S[\psi_a / \alpha] \dot{\leq} Q \dashv \Theta_1 \text{ (7)}$$

which is our goal.

□

Now we prove our main lemmas.

Lemma 74 (Decorating algorithmic subtyping). *If $\Theta_0 [P_0]_m^n P \leq Q \dashv \Theta_1$, then $\cdot \mid \Theta_0 [P_0]_m^n P \dot{\leq} Q \dashv \Theta_1$.*

Proof. By induction on the derivation of $\Theta_0 \vdash_m^n P \leq Q \dashv \Theta_1$. All cases except SI-FORALL are direct. The corresponding decorated rule has the same premises as the ordinary rule. For subtyping premises, we apply IH; for consistency premises in SI-CHECK and SI-ARROW, we apply Lemma 93. All tag contexts are empty. We show the only interesting case here.

Case

$$\begin{array}{c}
 \text{SI-FORALL} \\
 m \neq \text{I} \quad P_0 \text{ guarded or } P_0 = \text{!} \quad P \leq_m^n P_0 \not\leq^\alpha \Xi; P_a \text{ (1)} \quad \Theta_0, \Xi \vdash P_a \text{ (2)} \\
 \hline
 \Theta_0, \Xi \vdash_m^n P[P_a/\alpha] \leq Q \dashv \Theta_1 \text{ (3)} \\
 \hline
 \Theta_0 \vdash_m^n \forall \alpha. P \leq Q \dashv \Theta_1
 \end{array}$$

By IH on (3), we have

$$\cdot \mid \Theta_0, \Xi \vdash_m^n P[P_a/\alpha] \leq Q \dashv \Theta_1 \text{ (5)}.$$

By Lemma 76 on (1) and (2), there exist Ψ_a and ψ_a such that

$$\Psi_a \text{ fresh (4)} \quad \Theta_0, \Xi \vdash^n \psi_a \# \Psi_a \text{ (6)} \quad \lfloor \psi_a \rfloor = P_a \text{ (7)}.$$

By Lemma 84 on (1), (6), (7), and (5), with $S = P$ and $\Psi = \cdot$, we have

$$\Psi_a \mid \Theta_0, \Xi \vdash_m^n P[\psi_a/\alpha] \leq Q \dashv \Theta_1 \text{ (8)}.$$

Decorated SI-FORALL on (1), (2), (4), (6), (7), (8), and side-condition premises gives our goal

$$\cdot \mid \Theta_0 \vdash_m^n \forall \alpha. P \leq Q \dashv \Theta_1$$

□

Now we prove soundness of decorated algorithmic subtyping with respect to declarative subtyping. We prove a lemma for the case of flexible variables first.

Lemma 85 (Soundness of decorated algorithmic subtyping of flexible variables). *If $\cdot \mid \Theta_0 \mid \Xi \vdash_{\text{Fm}}^n \hat{\alpha} \leq Q \dashv \Theta_1$, then*

$$\Theta_0, \Xi \sqsubseteq \Theta_1 \quad \Theta_1 \vdash P_0 \blacktriangleright Q \quad \cdot \mid \Theta_1 \vdash_{\text{Fm}}^n \hat{\alpha} \leq Q.$$

Proof. By induction on the derivation of $\cdot \mid \Theta_0 \mid \Xi \vdash_{\text{Fm}}^n \hat{\alpha} \leq Q \dashv \Theta_1$, mutually with Lemma 75, which we use for SI-FLEX-SOLVE and SI-FLEX-ASSIGN. We only show these two interesting cases; other cases follow from straightforward application of the IH and weakening.

Case

$$\frac{\text{SI-FLEX-SOLVE} \quad \cdot \mid \Theta_0, \hat{\alpha}, \hat{\beta}, \hat{\alpha}_0 := \hat{\alpha} \rightarrow \hat{\beta}, \Xi [p_0] \vdash_{\text{Fm}}^n \hat{\alpha} \rightarrow \hat{\beta} \leq Q \dashv \Theta_1 \text{ ①}}{\cdot \mid \Theta_0, \hat{\alpha}_0 \mid \Xi [p_0] \vdash_{\text{Fm}}^n \hat{\alpha}_0 \leq Q \dashv \Theta_1}$$

By IH of Lemma 75 on ①, we have

$$\Theta_0, \hat{\alpha}, \hat{\beta}, \hat{\alpha}_0 := \hat{\alpha} \rightarrow \hat{\beta}, \Xi \sqsubseteq \Theta_1 \text{ ②} \quad \Theta_1 \vdash P_0 \blacktriangleright Q \text{ ③} \quad \cdot \mid \Theta_1 \vdash_{\text{Fm}}^n \hat{\alpha} \rightarrow \hat{\beta} \leq Q \text{ ④}.$$

By information increase from $\Theta_0, \hat{\alpha}, \hat{\beta}, \hat{\alpha}_0 := \hat{\alpha} \rightarrow \hat{\beta}, \Xi$ to Θ_1 , there exists τ such that

$$\Theta_1 \ni \hat{\alpha}_0 := \tau \text{ ⑤} \quad \Theta_1 \vdash \tau \equiv \hat{\alpha} \rightarrow \hat{\beta} \text{ ⑥}.$$

It is obvious that monotype equivalence preserves subtyping. Then by ⑥ and ④, we have

$$\cdot \mid \Theta_1 \vdash_{\text{Fm}}^n \tau \leq Q \text{ ⑦}.$$

By definition of information increase and ②, we have

$$\Theta_0, \hat{\alpha}_0, \Xi \sqsubseteq \Theta_1 \text{ ⑧}.$$

Our goal follows from ⑧, ③, and declarative S-FLEXMONO on ⑤ and ⑦.

Case

$$\frac{\text{SI-FLEX-ASSIGN} \quad \cdot \mid \Theta_0, \Xi [p_0[\tau/\hat{\beta}]] \vdash_{\text{Fm}}^n \hat{\alpha}[\tau/\hat{\beta}] \leq Q \dashv \Theta_1 \text{ ①}}{\cdot \mid \Theta_0, \hat{\beta} := \tau \mid \Xi [p_0] \vdash_{\text{Fm}}^n \hat{\alpha} \leq Q \dashv \Theta_1, \hat{\beta} := \tau}$$

By IH of Lemma 75 on ①, we have

$$\Theta_0, \Xi \sqsubseteq \Theta_1 \text{ ②} \quad \Theta_1 \vdash P_0[\tau/\hat{\beta}] \blacktriangleright Q \text{ ③} \quad \cdot \mid \Theta_1 \vdash_{\text{Fm}}^n \hat{\alpha}[\tau/\hat{\beta}] \leq Q \text{ ④}.$$

By definition of information increase and ②, we have

$$\Theta_0, \hat{\beta} := \tau, \Xi \sqsubseteq \Theta_1, \hat{\beta} := \tau \text{ ⑤}.$$

By definition of refinement and weakening on ③, we have

$$\Theta_1, \hat{\beta} := \tau \vdash P_0 \blacktriangleright Q \text{ ⑥}.$$

Case analysis on $\hat{\alpha}$.

Subcase $\hat{\alpha} = \hat{\beta}$. By weakening on ④ and declarative S-FLEXMONO on the lookup $\Theta_1, \hat{\beta} := \tau \ni \hat{\beta} := \tau$, we have

$$\cdot \mid \Theta_1, \hat{\beta} := \tau \vdash_{\text{Fm}}^n \hat{\beta} \leq Q \text{ ⑦}.$$

Our goal follows from ⑤, ⑥, and ⑦.

Subcase $\hat{\alpha} \neq \hat{\beta}$. By context weakening on ④, we have

$$\cdot \mid \Theta_1, \hat{\beta} := \tau \vdash_{\text{Fm}}^n \hat{\alpha} \leq Q \text{ ⑦}.$$

Our goal follows from ⑤, ⑥, and ⑦.

□

Lemma 75 (Soundness of decorated subtyping). *If $\Psi \mid \Theta_0 [P_0]_{\mathfrak{m}}^n S \dot{\leq} Q \dashv \Theta_1$, then*

$$\Theta_0 \sqsubseteq \Theta_1 \quad \Theta_1 \vdash P_0 \blacktriangleright Q \quad \Psi \mid \Theta_1 \vdash_{\mathfrak{m}}^n S \leq Q.$$

Consequently, given a problem of subtyping $(\Theta_0; n; \mathfrak{m}; P_0; P)$, if $\cdot \mid \Theta_0 [P_0]_{\mathfrak{m}}^n P \dot{\leq} Q \dashv \Theta_1$, then $(\Theta_0 \sqsubseteq \Theta_1; Q)$ is a solution to it.

Proof. By induction on the derivation of $\Psi \mid \Theta_0 [P_0]_{\mathfrak{m}}^n S \dot{\leq} Q \dashv \Theta_1$, mutually with Lemma 85, which we use for the SI-FLEXMONO case. The two invariants $\Theta_0 \sqsubseteq \Theta_1$ and $\Theta_1 \vdash P_0 \blacktriangleright Q$ are obvious in most cases following definitions, IH, or other lemmas. We focus on showing $\Psi \mid \Theta_1 \vdash_{\mathfrak{m}}^n S \leq Q$ in each case, except for SI-ARROW whose refinement invariant is non-trivial.

Case

SI-INFER

$$\frac{}{\cdot \mid \Theta_0 [\mathbb{A}]_{\mathfrak{I}}^n S \dot{\leq} [S] \dashv \Theta_0}$$

By decorated consistency U-GHOSTR, we have

$$\cdot \mid \Theta_0 [\mathbb{A}]_{\mathfrak{I}}^n S \dot{\sim} [S] \dashv \Theta_0 \text{ ①}.$$

By Lemma 92 on ①, we have

$$\cdot \mid \Theta_0 \vdash S \sim [S] \text{ ②}.$$

Our goal follows from declarative S-INFER on ②.

Case

SI-CHECK

$$\frac{T, B \text{ guarded} \quad \Psi \mid \Theta_0 [B]_{\mathfrak{I}}^n T \dot{\sim} B \dashv \Theta_1 \text{ ①}}{\Psi \mid \Theta_0 [B]_{\mathfrak{I}}^n T \dot{\leq} B \dashv \Theta_1}$$

By Lemma 92 on ①, we have

$$\Psi \mid \Theta_1 \vdash T \sim B \text{ ②}.$$

Our goal follows from declarative S-CHECK on the guardedness premise and ②.

Case

SI-FORALLR

$$\frac{\Psi \mid \Theta_0, \alpha [B]_{\mathfrak{I}}^n S \dot{\leq} B \dashv \Theta_1, \alpha, \Xi \text{ ①}}{\Psi \mid \Theta_0 [\forall \alpha. B]_{\mathfrak{I}}^n S \dot{\leq} \forall \alpha. B \dashv \Theta_1}$$

By IH on ①, we have

$$\Psi \mid \Theta_1, \alpha, \Xi \vdash_c^{\text{ty}} S \leq B \text{ ②}.$$

Since Ξ is fresh for S and B , by context strengthening for unused variables in Ξ , we have

$$\Psi \mid \Theta_1, \alpha \vdash_c^{\text{ty}} S \leq B \text{ ③}.$$

Our goal follows from declarative S-FORALLR on ③.

Case

$$\frac{\text{SI-UNIVGHOSTR} \quad S \text{ guarded}}{\cdot \mid \Theta_0 [\mathbb{A}] \vdash_c^{\text{sk}} S \leq \mathbb{A}.[S] \vdash \Theta_0}$$

By decorated consistency U-GHOSTR, we have

$$\cdot \mid \Theta_0 [\mathbb{A}] \vdash S \sim [S] \vdash \Theta_0 \text{ ①}.$$

By Lemma 92 on ①, we have

$$\cdot \mid \Theta_0 \vdash S \sim [S] \text{ ②}.$$

By the guardedness premise and preservation of guardedness by erasure, our goal follows from declarative S-UNIVGHOSTR on ②.

Case

$$\frac{\text{SI-GHOST}}{\cdot \mid \Theta_0 [P_0] \vdash_m^{\text{sk}} \mathbb{A} \leq P_0 \vdash \Theta_0}$$

Our goal follows from declarative S-GHOST.

Case

$$\frac{\text{SI-FLEXMONO} \quad \cdot \mid \Theta_0 \mid \cdot [P_0] \vdash_{\text{Fm}}^n \hat{a} \leq Q \vdash \Theta_1 \text{ ①}}{\cdot \mid \Theta_0 [P_0] \vdash_{\text{Fm}}^n \hat{a} \leq Q \vdash \Theta_1}$$

By Lemma 85 on ①, we have

$$\cdot \mid \Theta_1 \vdash_{\text{Fm}}^n \hat{a} \leq Q \text{ ②}.$$

Our goal follows from ②.

Case

$$\frac{\text{SI-ARROW} \quad \Psi_1 \mid \Theta_0 [P'_1] \vdash S_1 \sim Q_1 \vdash \Theta_1 \text{ ①} \quad \Psi_2 \mid \Theta_1 [P'_2] \vdash_m^n S_2 \leq Q_2 \vdash \Theta_2 \text{ ②}}{\Psi_1, \Psi_2 \mid \Theta_0 [P'_1 \rightarrow P'_2] \vdash_{\text{Fm}}^n S_1 \rightarrow S_2 \leq Q_1 \rightarrow Q_2 \vdash \Theta_2}$$

By Lemma 92 on ①, we have

$$\Theta_0 \sqsubseteq \Theta_1 \text{ ③} \quad \Theta_1 \vdash P'_1 \blacktriangleright Q_1 \text{ ④} \quad \Psi_1 \mid \Theta_1 \vdash S_1 \sim Q_1 \text{ ⑤}.$$

By IH on ②, we have

$$\Theta_1 \sqsubseteq \Theta_2 \text{ ⑥} \quad \Theta_2 \vdash P'_2 \blacktriangleright Q_2 \text{ ⑦} \quad \Psi_2 \mid \Theta_2 \vdash_m^n S_2 \leq Q_2 \text{ ⑧}.$$

By transitivity of information increase on ③ and ⑥, we have

$$\Theta_0 \sqsubseteq \Theta_2 \text{ ⑨}.$$

By stability under information increase on ④ and ⑥, we have

$$\Theta_2 \vdash P'_1 \blacktriangleright Q_1 \text{ ⑩}.$$

By ⑩ and ⑦, we have

$$\Theta_2 \vdash P'_1 \rightarrow P'_2 \blacktriangleright Q_1 \rightarrow Q_2 \text{ ⑪}.$$

By stability under information increase on ⑤ and ⑥, we have

$$\Psi_1 \mid \Theta_2 \vdash S_1 \sim Q_1 \text{ ⑫}.$$

Our goal follows from ⑨, ⑪, and declarative S-ARROW on ⑫ and ⑧.

Case

SI-FORALL

$$\frac{\begin{array}{c} m \neq I \quad P_0 \text{ guarded or } P_0 = \mathbb{A} \quad [S] = P \\ P \leq_m^n P_0 \not\vdash^\alpha \Xi; P_1 \quad \Theta_0, \Xi \vdash P_1 \quad \Psi_g \text{ fresh} \quad \Theta_0, \Xi \vdash^n \psi_g \# \Psi_g \text{ ①} \\ [\psi_g] = P_1 \quad \Psi, \Psi_g \mid \Theta_0, \Xi [P_0]_m^n S[\psi_g/\alpha] \leq Q \dashv \Theta_1 \text{ ②} \end{array}}{\Psi \mid \Theta_0 [P_0]_m^n \forall^\delta \alpha. S \leq Q \dashv \Theta_1}$$

By IH on ②, we have

$$\Psi, \Psi_g \mid \Theta_1 \vdash_m^n S[\psi_g/\alpha] \leq Q \text{ ③}.$$

By the information-increase part of the IH on ② and stability under information increase on ①, we have

$$\Theta_1 \vdash^n \psi_g \# \Psi_g \text{ ④}.$$

Our goal follows from declarative S-FORALL on the freshness premise, ④, and ③.

Case

SI-GHOSTG

$$\frac{}{\cdot \mid \Theta_0 [\mathbb{A}]_m^{\text{sk}} \mathbb{A} \leq \mathbb{A} \dashv \Theta_0}$$

Our goal follows from declarative S-GHOSTG.

Case

$$\frac{\text{SI-UNIVGHOSTL} \quad m \neq I \quad \Psi \mid \Theta_0 \mid P_0 \vdash_m^{\text{sk}} S \dot{\leq} Q \dashv \Theta_1 \text{ (1)}}{\Psi \mid \Theta_0 \mid P_0 \vdash_m^{\text{sk}} \mathbb{A}.S \dot{\leq} Q \dashv \Theta_1}$$

By IH on (1), we have

$$\Psi \mid \Theta_1 \vdash_m^{\text{sk}} S \leq Q \text{ (2)}.$$

Our goal follows from declarative S-UNIVGHOSTL on (2).

Case

$$\frac{\text{SI-UNIVGHOSTLG} \quad m \neq I \quad \Psi \mid \Theta_0 \mid P_0 \vdash_m^{\text{sk}} S \dot{\leq} Q \dashv \Theta_1 \text{ (1)}}{\Psi \mid \Theta_0 \mid P_0 \vdash_m^{\text{sk}} \mathbb{A}.S \dot{\leq} Q \dashv \Theta_1}$$

By IH on (1), we have

$$\Psi \mid \Theta_1 \vdash_m^{\text{sk}} S \leq Q \text{ (2)}.$$

Our goal follows from declarative S-UNIVGHOSTLG on (2).

□

Finally we prove soundness of algorithmic subtyping with respect to declarative subtyping.

Theorem 27 (Soundness of subtyping). *Given a subtyping problem $(\Theta_0; n; m; P_0; P)$, if $\Theta_0 \mid P_0 \vdash_m^n P \leq Q \dashv \Theta_1$, then $(\Theta_0 \sqsubseteq \Theta_1; Q)$ is a solution.*

Proof. By Lemma 74 on $\Theta_0 \mid P_0 \vdash_m^n P \leq Q \dashv \Theta_1$, we have $\cdot \mid \Theta_0 \mid P_0 \vdash_m^n P \dot{\leq} Q \dashv \Theta_1 \text{ (1)}$. Our goal follows from Lemma 75 on (1). □

D.4.3 Completeness of subtyping

In this section, we prove Theorem 28.

Theorem 28 (Completeness and optimality of subtyping). *Given a subtyping problem $(\Theta_0; n; m; P_0; P)$, if there exists a solution, $\Theta_0 \mid P_0 \vdash_m^n P \leq P_1 \dashv \Theta_1$ gives its optimal solution $(\Theta_0 \sqsubseteq \Theta_1; P_1)$.*

It is enough to show that if $(\theta :: \Theta_0 \sqsubseteq \Theta; Q)$ is a solution to the subtyping problem $(\Theta_0; n; m; P_0; P)$, then we have

$$\begin{aligned} \Theta_0 \mid P_0 \vdash_m^n P &\leq P_1 \dashv \Theta_1 \\ \theta_1 :: \Theta_1 &\sqsubseteq \Theta \\ \Theta &\vdash \theta_1 P_1 \leq Q \\ \theta &\equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \end{aligned}$$

We write $\Theta \vdash P \leq_{\Psi} \psi$ for partial precision, omitting the empty Δ context. Its full form is $\Theta \mid \Delta \vdash P \leq_{\Psi} \psi$. It reads as under Θ and Δ , skeleton P is partially more precise than ψ on tags Ψ . It is a partial form of $\Theta \vdash P \leq \lfloor \psi \rfloor$ in the sense that quantifiers in ψ whose tags are not in Ψ may be matched by ghosts in P . As a result, its definition corresponds one to one to the definition of refinement in algorithmic contexts. The tag context Ψ records the tags of ψ that have corresponding quantifiers in P . We define partial precision syntactically as follows.

$$\begin{array}{c}
\frac{}{\Theta \mid \Delta \vdash 1 \leq 1} \quad \frac{}{\Theta \mid \Delta \vdash \alpha \leq \alpha} \quad \frac{\Theta \mid \Delta \vdash P \leq_{\Psi_1} \psi_1 \quad \Theta \mid \Delta \vdash Q \leq_{\Psi_2} \psi_2}{\Theta \mid \Delta \vdash P \rightarrow Q \leq_{\Psi_1, \Psi_2} \psi_1 \rightarrow \psi_2} \\
\\
\frac{\Theta, \alpha \mid \Delta, \alpha \vdash P \leq_{\Psi} \psi}{\Theta \mid \Delta \vdash \forall \alpha. P \leq_{\Psi, \epsilon} \forall^{\epsilon} \alpha. \psi} \quad \frac{\Delta \cap \text{ftv}(\psi) = \emptyset}{\Theta \mid \Delta \vdash \mathbb{A} \leq \psi} \quad \frac{\Theta \mid \Delta \vdash P \leq_{\Psi} \psi}{\Theta \mid \Delta \vdash \mathbb{A}. P \leq_{\Psi} \psi} \\
\\
\frac{\Theta, \alpha \mid \Delta \vdash \mathbb{A}. P \leq_{\Psi} \psi}{\Theta \mid \Delta \vdash \mathbb{A}. P \leq_{\Psi} \forall^{\epsilon} \alpha. \psi} \quad \frac{\Theta \mid \Delta \vdash P \leq_{\Psi} \psi}{\Theta \mid \Delta \vdash \mathbb{A}. P \leq_{\Psi} \mathbb{A}. \psi} \\
\\
\frac{\Theta \ni \hat{\alpha} := \tau \quad \Theta \mid \Delta \vdash \tau \leq_{\Psi} \psi}{\Theta \mid \Delta \vdash \hat{\alpha} \leq_{\Psi} \psi} \quad \frac{\Theta \ni \hat{\alpha} := \tau \quad \Theta \mid \Delta \vdash P \leq_{\Psi} \tau}{\Theta \mid \Delta \vdash P \leq_{\Psi} \hat{\alpha}}
\end{array}$$

Lemma 86 (Complete partial precision). *If*

$$\Theta \vdash^n \psi \# \Psi \quad \Theta \vdash P \leq_{\Psi} \psi,$$

then

$$\Theta \vdash P \leq \lfloor \psi \rfloor.$$

Proof. By well-formedness of ψ , every tag in it is unique and is in Ψ . Then our goal follows from a straightforward induction on the derivation of partial precision $\Theta \vdash P \leq_{\Psi} \psi$, observing that every tagged quantifier in ψ must correspond to a quantifier in P in order to be solved. $\square$

Lemma 87 (Stability of subtyping under more precise substitution). *If*

$$\Theta \vdash P_a \leq \lfloor \psi_a \rfloor \quad \Theta \vdash^n \psi_a \# \Psi' \quad \Psi' \text{ fresh} \quad \Psi_a \subseteq \Psi' \quad \Psi, \Psi_a \mid \Theta \vdash_m^n S[\psi_a/\alpha] \leq Q$$

and P_a is a type when $n = \text{ty}$, then

$$\Psi \mid \Theta \vdash_m^n S[P_a/\alpha] \leq Q.$$

Proof. By a straightforward induction on the subtyping derivation $\Psi, \Psi_a \mid \Theta \vdash_m^n S[\psi_a/\alpha] \leq Q$, observing that by $\Theta \vdash P_a \leq \lfloor \psi_a \rfloor$, replacing ψ_a with P_a would not result in any type mismatch. $\square$

Lemma 88 (Completeness of constraint collection and solving for variables). *Suppose*

$$\Theta_0 \vdash P_0 \quad \Theta_0 \vdash P_i \quad P_0 :^n m \quad \theta :: \Theta_0 \sqsubseteq \Theta.$$

Let ρ be a capture-avoiding substitution from rigid type variables to decorated skeletons, and Ψ be the solved tags introduced by ρ . If ψ_a is a well-formed instantiation and the premise solves Ψ_a :

$$\Theta \vdash^n \psi_a \# \Psi' \quad \Psi_i, \Psi_a = \Psi' \quad \Psi' \text{ fresh}$$

and

$$\Theta \mid \Delta \vdash \theta P_i \leq_{\Psi_i} \psi_a \quad \Psi, \Psi_a \mid \Theta \vdash_m^n \rho(\psi_a) \leq Q \quad \Theta \vdash \theta P_0 \blacktriangleright Q.$$

Moreover, for every rigid variable in ψ_a substituted by ρ , P_i has the same rigid variable in the same position. Then there exist Ξ_s, P_a , and θ_a such that

$$P_i \leq_m P_0 \rightsquigarrow \Xi_s; P_a \quad \Theta_0, \Xi_s \vdash P_a$$

and

$$\theta_a :: \Theta_0, \Xi_s \sqsubseteq \Theta \quad \theta \equiv \theta_a :: \Theta_0 \sqsubseteq \Theta \quad \Theta \mid \Delta \vdash \theta_a P_a \leq_{\Psi_i, \Psi_a} \psi_a.$$

Proof. By induction on the derivation of $\Psi, \Psi_a \mid \Theta \vdash_m^n \rho(\psi_a) \leq Q$, generalising over $\Theta_0, P_0, P_i, \theta, \psi_a, \Psi', \Psi_i, \Psi_a, \Psi, \rho$, and Δ . The cases S-INFER, S-UNIVGHOSTR, and S-GHOSTG, whose conclusion has empty solved tag context, are similar to the corresponding cases in Lemma 89. The S-GHOST case cannot occur because ψ_a cannot contain $\mathcal{L}$. We only show non-trivial cases below.

Case

$$\frac{\text{S-CHECK} \quad \rho(\psi_a), Q \text{ guarded} \quad \Psi, \Psi_a \mid \Theta \vdash \rho(\psi_a) \sim Q \text{ ①}}{\Psi, \Psi_a \mid \Theta \vdash_c^{\text{ty}} \rho(\psi_a) \leq Q}$$

By Lemma 97 on ①, the precision premise, the refinement premise, the substitution ρ , and the tag context premises derived from $\Psi_i, \Psi_a = \Psi'$, we have

$$\begin{aligned} P_i \sim P_0 \rightsquigarrow \Xi_s; P_a \text{ ②} \quad & \Theta_0, \Xi_s \vdash P_a \text{ ③} \\ \theta_a :: \Theta_0, \Xi_s \sqsubseteq \Theta \text{ ④} \quad & \theta \equiv \theta_a :: \Theta_0 \sqsubseteq \Theta \text{ ⑤} \quad \Theta \mid \Delta \vdash \theta_a P_a \leq_{\Psi_i, \Psi_a} \psi_a \text{ ⑥.} \end{aligned}$$

Also, $P_0 :^{\text{ty}} c$ gives that P_0 is a concrete type. The refinement premise then gives θP_0 is a guarded type. Hence P_0 is guarded. We proceed by case analysis on P_i .

Subcase P_i guarded. By SOLS-CHECK on ②, we have

$$P_i \leq_c P_0 \rightsquigarrow \Xi_s; P_a \text{ ⑧}.$$

Our goal follows from ③–⑧.

Subcase $P_i = \mathbb{A}$. By inversion on ②, we have $\Xi_s = \cdot$ and $P_a = P_0$. By SOLS-GHOSTL, we have

$$\mathbb{A} \leq_c P_0 \rightsquigarrow \mathbb{A}.P_0 \text{ ⑧}.$$

By ③ and guardedness of P_0 , we have

$$\Theta_0 \vdash \mathbb{A}.P_0 \text{ ⑨}.$$

By ⑥, $P_a = P_0$, and definition of partial precision, we have

$$\Theta \mid \Delta \vdash \theta_a(\mathbb{A}.P_0) \leq_{\Psi_i, \Psi_a} \psi_a \text{ ⑩}.$$

Our goal follows by choosing $\mathbb{A}.P_0$ as the output skeleton, and using ④, ⑤, ⑧, ⑨, and ⑩.

Subcase $P_i = \forall \beta. P'_i$. Impossible. The precision premise $\Theta \mid \Delta \vdash \theta P_i \leq_{\Psi_i} \psi_a$ and guardedness of $\rho(\psi_a)$, together with the fact that ρ preserves top-level quantifiers, imply that P_i cannot have a top-level quantifier.

Subcase $P_i = \mathbb{A}.P'_i$. Since P_i is well-formed, P'_i is guarded. By congruence of constraint solving, we assume that ② is derived by SOLE-GHOSTGUARDED. Inversion on ② gives

$$P'_i \sim P_0 \rightsquigarrow \Xi_s; P_a \text{ ⑧}.$$

By SOLS-CHECK on ⑧, we have

$$P'_i \leq_c P_0 \rightsquigarrow \Xi_s; P_a \text{ ⑨}.$$

Then by SOLS-UNIVGHOSTL on ⑨, we have

$$\mathbb{A}.P'_i \leq_c P_0 \rightsquigarrow \Xi_s; \mathbb{A}.P_a \text{ ⑩}.$$

Since P'_i and P_0 are guarded, ⑧ gives that P_a is guarded. Hence $\Theta_0, \Xi_s \vdash \mathbb{A}.P_a$ ⑬. By ⑥ and definition of partial precision, we have

$$\Theta \mid \Delta \vdash \theta_a(\mathbb{A}.P_a) \leq_{\Psi_i, \Psi_a} \psi_a \text{ ⑫}.$$

Our goal follows by choosing $\mathbb{A}.P_a$ as the output skeleton and using ④, ⑤, ⑩, ⑫, and ⑬.

Case

$$\frac{\text{S-FORALLR} \quad \Psi, \Psi_a \mid \Theta, \beta \vdash_c^{\text{ty}} \rho(\psi_a) \leq Q \text{ ①}}{\Psi, \Psi_a \mid \Theta \vdash_c^{\text{ty}} \rho(\psi_a) \leq \forall \beta. Q}$$

By the refinement premise, we have $P_0 = \forall\beta.P'_0$ and $\Theta, \beta \vdash \theta P'_0 \triangleright Q$ ②. From $P_0 :^{ty} c$ and $P_0 = \forall\beta.P'_0$, we have $P'_0 :^{ty} c$ ③. By IH on ① with ②, ③, and context weakening on the other premises, there are Ξ_s, P_a , and θ_a such that

$$\begin{aligned} P_i \leq_c P'_0 \rightsquigarrow \Xi_s; P_a & \text{ ⑥} & \Theta_0, \beta, \Xi_s \vdash P_a & \text{ ⑦} \\ \theta_a :: \Theta_0, \beta, \Xi_s \sqsubseteq \Theta, \beta & \text{ ⑧} & \theta \equiv \theta_a :: \Theta_0, \beta \sqsubseteq \Theta, \beta & \text{ ⑨} \\ \Theta, \beta \mid \Delta \vdash \theta_a P_a \leq_{\Psi_i, \Psi_a} \psi_a & \text{ ⑩}. \end{aligned}$$

By SOLS-FORALLR on ⑥, we have

$$P_i \leq_c \forall\beta.P'_0 \rightsquigarrow \Xi_s; P_a \text{ ⑪}.$$

By the well-formedness premise of ψ_a , we have $\beta \notin \text{ftv}(\psi_a)$. Then by ⑩, we have $\beta \notin \text{ftv}(\theta_a P_a)$, and thus $\beta \notin \text{ftv}(P_a)$. Therefore ⑦ holds in the context without β . By definition of metasubstitutions, ⑧ and ⑨ also hold in the context without β . ⑩ holds in the context without β as well. Our goal follows from these judgements after removing β from the contexts and ⑪.

Case

S-ARROW

$$\frac{\Psi_1, \Psi_{a,1} \mid \Theta \vdash \rho_1(\psi_1) \sim Q_1 \text{ ①} \quad \Psi_2, \Psi_{a,2} \mid \Theta \vdash_m^n \rho_2(\psi_2) \leq Q_2 \text{ ②}}{\Psi_1, \Psi_{a,1}, \Psi_2, \Psi_{a,2} \mid \Theta \vdash_{Fm}^n \rho_1(\psi_1) \rightarrow \rho_2(\psi_2) \leq Q_1 \rightarrow Q_2}$$

In this case $\psi_a = \psi_1 \rightarrow \psi_2$, and ρ_1 and ρ_2 are the restrictions of ρ to the two premises, whose introduced tag contexts are Ψ_1 and Ψ_2 . By $P_0 :^n Fm$, we have $P_0 = P_{0,1} \rightarrow P_{0,2}$. Together with the refinement premise, we have

$$\Theta \vdash \theta P_{0,1} \triangleright Q_1 \text{ ③} \quad \Theta \vdash \theta P_{0,2} \triangleright Q_2 \text{ ④} \quad P_{0,2} :^n m \text{ ⑤}.$$

The conclusion S-ARROW gives

$$\Psi = \Psi_1, \Psi_2 \text{ ⑧} \quad \Psi_a = \Psi_{a,1}, \Psi_{a,2} \text{ ⑨}$$

We proceed by case analysis on P_i .

Subcase $P_i = P_{i,1} \rightarrow P_{i,2}$. By inversion on the precision premise, there are $\Psi_{i,1}$ and $\Psi_{i,2}$ such that

$$\Psi_i = \Psi_{i,1}, \Psi_{i,2} \text{ ⑩} \quad \Theta \mid \Delta \vdash \theta P_{i,1} \leq_{\Psi_{i,1}} \psi_1 \text{ ⑬} \quad \Theta \mid \Delta \vdash \theta P_{i,2} \leq_{\Psi_{i,2}} \psi_2 \text{ ⑭}.$$

Combining ⑨, ⑩, and $\Psi_i, \Psi_a = \Psi'$, by well-formedness of ψ_a , we have

$$\Theta \vdash^n \psi_1 \# \Psi_{i,1}, \Psi_{a,1} \text{ ⑪} \quad \Theta \vdash^n \psi_2 \# \Psi_{i,2}, \Psi_{a,2} \text{ ⑫}.$$

By Lemma 97 on (1), (3), (11), (13), taking ρ_1 as the substitution, Ψ_1 as the tags introduced by the substitution, $\Psi_{i,1}$ as the already solved tag context, and $\Psi_{a,1}$ as the newly solved tag context, there are Ξ_1 , P_1 , and θ_1 such that

$$\begin{aligned} P_{i,1} \sim P_{0,1} \rightsquigarrow \Xi_1; P_1 \quad (15) \quad & \Theta_0, \Xi_1 \vdash P_1 \quad (16) \\ \theta_1 :: \Theta_0, \Xi_1 \sqsubseteq \Theta \quad (17) \quad & \theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \quad (18) \quad & \Theta \mid \Delta \vdash \theta_1 P_1 \leq_{\Psi_{i,1}, \Psi_{a,1}} \psi_1 \quad (19). \end{aligned}$$

By (18) and equivalence of metasubstitutions, (4) and (14) can be rewritten with θ_1 . By IH on (2), (4), (14), (5), (12), (17), (18), taking $\Psi_{i,2}$ as the already solved tag context, $\Psi_{a,2}$ as the newly solved tag context, and ρ_2 as the substitution with introduced tags Ψ_2 , and Θ_0, Ξ_1 as the input context, there are Ξ_2 , P_2 , and θ_a such that

$$\begin{aligned} P_{i,2} \leq_m P_{0,2} \rightsquigarrow \Xi_2; P_2 \quad (21) \quad & \Theta_0, \Xi_1, \Xi_2 \vdash P_2 \quad (22) \\ \theta_a :: \Theta_0, \Xi_1, \Xi_2 \sqsubseteq \Theta \quad (23) \quad & \theta_1 \equiv \theta_a :: \Theta_0, \Xi_1 \sqsubseteq \Theta \quad (24) \\ \Theta \mid \Delta \vdash \theta_a P_2 \leq_{\Psi_{i,2}, \Psi_{a,2}} \psi_2 \quad (25). \end{aligned}$$

By (24) and equivalence of metasubstitutions, (19) can be rewritten with θ_a . By SOLS-ARROW on (15) and (21), we have

$$P_{i,1} \rightarrow P_{i,2} \leq_{\text{fm}} P_{0,1} \rightarrow P_{0,2} \rightsquigarrow \Xi_1, \Xi_2; P_1 \rightarrow P_2 \quad (26).$$

By (18), (24), and transitivity of equivalence of metasubstitutions, we have

$$\theta \equiv \theta_a :: \Theta_0 \sqsubseteq \Theta \quad (27).$$

By weakening (16) with Ξ_2 and using (22), we have $\Theta_0, \Xi_1, \Xi_2 \vdash P_1 \rightarrow P_2$ (28).

By (9), (10), (24), the rewritten form of (19), and (25), we have

$$\Theta \mid \Delta \vdash \theta_a (P_1 \rightarrow P_2) \leq_{\Psi_i, \Psi_a} \psi_1 \rightarrow \psi_2 \quad (29).$$

Our goal follows by choosing $\Xi_s = \Xi_1, \Xi_2$ and $P_a = P_1 \rightarrow P_2$, and using (23), (26)–(29).

Subcase $P_i = \mathbb{L}$. Similar to the above subcase. Use SOLS-GHOSTLF instead of SOLS-ARROW to combine the two constraint solving judgements. Concretely, inversion on the precision premise gives $\Psi_i = \cdot$. We apply Lemma 97 to (1) with input skeleton $P_i = \mathbb{L}$, and then apply the IH to (2) with input skeleton $P_i = \mathbb{L}$. This gives

$$\begin{aligned} \mathbb{L} \sim P_{0,1} \rightsquigarrow \Xi_1; P_1 \quad (30) \quad & \Theta_0, \Xi_1 \vdash P_1 \quad (31) \\ \mathbb{L} \leq_m P_{0,2} \rightsquigarrow \Xi_2; P_2 \quad (32) \quad & \Theta_0, \Xi_1, \Xi_2 \vdash P_2 \quad (33) \end{aligned}$$

together with the corresponding metasubstitution and precision premises.
By SOLS-GHOSTLF on ③① and ③②, we have

$$\hat{\Delta} \leq_{\text{Fm}} P_{0,1} \rightarrow P_{0,2} \rightsquigarrow \Xi_1, \Xi_2; \hat{\Delta}.(P_1 \rightarrow P_2) \text{ ③④}.$$

The output skeleton is well-formed by ③① and ③③. The precision conclusion $\Theta \mid \Delta \vdash \theta_a(\hat{\Delta}.(P_1 \rightarrow P_2)) \leq_{\Psi_i, \Psi_a} \psi_1 \rightarrow \psi_2$ is obviously true. The goal uses $\Xi_s = \Xi_1, \Xi_2$.

Subcase $P_i = 1, P_i = \alpha$, or $P_i = \forall\beta.P'_i$. Impossible by the precision premise.

Subcase $P_i = \hat{\alpha}$. By SOLS-FLEX, we have

$$\hat{\alpha} \leq_{\text{Fm}} P_0 \rightsquigarrow \hat{\alpha} \text{ ⑩}.$$

Our goal follows by choosing $P_a = \hat{\alpha}$, ⑩, and some other obvious judgments.

Subcase $P_i = \hat{\Delta}.P'_i$. By well-formedness, P'_i is guarded, and inversion on the precision premise gives $\Theta \mid \Delta \vdash \theta P'_i \leq_{\Psi_i} \psi_1 \rightarrow \psi_2$ ⑩. We reuse the previous subcases on ⑩ to obtain an output skeleton P_a , and then use SOLS-UNIVGHOSTL to lift it to $\hat{\Delta}.P_a$.

Case

S-FORALL

$$\frac{\Psi_b \text{ fresh} \quad \Theta \vdash^n \psi_b \# \Psi_b \text{ ①} \quad \begin{array}{c} m \neq I \\ \Psi, \Psi_a, \Psi_b \mid \Theta \vdash_m^n (\rho(\psi'))[\psi_b/\beta] \leq Q \text{ ②} \end{array}}{\Psi, \Psi_a \mid \Theta \vdash_m^n \rho(\forall^\epsilon \beta. \psi') \leq Q}$$

In this case $\psi_a = \forall^\epsilon \beta. \psi'$. Let $\rho_b = \rho \circ [\psi_b/\beta]$. It introduces the tag context Ψ, Ψ_b , and by capture-avoiding substitution ② is equivalent to

$$\Psi, \Psi_b, \Psi_a \mid \Theta \vdash_m^n \rho_b(\psi') \leq Q \text{ ⑨}.$$

The precision premise is

$$\Theta \mid \Delta \vdash \theta P_i \leq_{\Psi_i} \forall^\epsilon \beta. \psi' \text{ ⑩}.$$

By well-formedness of ψ_a , the tag ϵ uniquely appears on the top-level quantifier. By S-FORALL, we have $\epsilon \notin \Psi_a$. Then by $\Psi_i, \Psi_a = \Psi'$, we must have $\epsilon \in \Psi_i$. Let $\Psi_i = \Psi'_i \uplus \{\epsilon\}$. Then by ⑩, we know that P_i must have a top-level quantifier. Let $P_i = \forall\beta.P_1$. By well-formedness of P_i , we have

$$\Theta_0, \beta \vdash P_1 \text{ ⑭}.$$

By inversion on ⑩, we have

$$\Theta, \beta \mid \Delta, \beta \vdash \theta P_1 \leq_{\Psi'_i} \psi' \text{ ⑪}.$$

By well-formedness of ψ_a and $\Psi_i, \Psi_a = \Psi'$, we have

$$\Theta, \beta \vdash^n \psi' \# \Psi'_i, \Psi_a \text{ (12)}.$$

By weakening (9) to Θ, β and applying the IH with ρ_b , taking Ψ, Ψ_b as the tag context introduced by the substitution, Ψ'_i as the already solved tag context, and Ψ_a as the newly solved tag context, there are Ξ_s, P_b , and θ_a such that

$$\begin{aligned} P_1 \leq_m P_0 \rightsquigarrow \Xi_s; P_b \text{ (3)} \quad & \Theta_0, \beta, \Xi_s \vdash P_b \text{ (4)} \\ \theta_a :: \Theta_0, \beta, \Xi_s \sqsubseteq \Theta, \beta \text{ (5)} \quad & \theta \equiv \theta_a :: \Theta_0, \beta \sqsubseteq \Theta, \beta \text{ (6)} \quad & \Theta, \beta \mid \Delta, \beta \vdash \theta_a P_b \leq_{\Psi'_i, \Psi_a} \psi' \text{ (7)}. \end{aligned}$$

By SOLS-FORALL on (3), we have

$$\forall \beta. P_1 \leq_m P_0 \rightsquigarrow \Xi_s; \forall \beta. P_b \text{ (8)}.$$

By (4), we have $\Theta_0, \Xi_s \vdash \forall \beta. P_b$ (9). By the partial precision rule for quantifiers and $\Psi_i = \Psi'_i \uplus \{\epsilon\}$, (7) gives

$$\Theta \mid \Delta \vdash \theta_a (\forall \beta. P_b) \leq_{\Psi_i, \Psi_a} \forall^\epsilon \beta. \psi' \text{ (13)}.$$

The restrictions of (5) and (6) to the contexts without the fresh β give the required metasubstitution and equivalence premises. Our goal follows by choosing $P_a = \forall \beta. P_b$.

Case

$$\frac{\text{S-UNIVGHOSTLG} \quad m \neq I \quad \Psi, \Psi_a \mid \Theta \vdash_m^{\text{sk}} \rho(\psi') \leq Q \text{ (1)}}{\Psi, \Psi_a \mid \Theta \vdash_m^{\text{sk}} \mathbb{A}.\rho(\psi') \leq Q}$$

Similar to the previous case of S-FORALL. We know that P_i must have a top-level $\mathbb{A}$, then apply IH on (1) with the body of P_i , and finally reconstruct the goals we want for P_i . It is simpler than the previous case since we do not need to do instantiation and re-abstraction of type variables.

□

Lemma 89 (Completeness of constraint collection and solving). *Suppose*

$$\Theta_0 \vdash P_0 \quad \Theta_0 \vdash P_i \quad P_0 :^n m \quad [S] = P \quad \theta :: \Theta_0 \sqsubseteq \Theta \quad \alpha \notin \Theta.$$

If

$$\Theta \vdash^n \psi_a \# \Psi' \quad \Psi \text{ is disjoint from } \Psi' \quad \Psi_i, \Psi_a = \Psi'$$

and

$$\Theta \mid \Delta \vdash \theta P_i \leq_{\Psi_i} \psi_a \quad \Psi, \Psi_a \mid \Theta \vdash_m^n (\theta S)[\psi_a/\alpha] \leq Q \quad \Theta \vdash \theta P_0 \blacktriangleright Q,$$

then there exist C, Ξ_c, Ξ_s, P_a , and θ_a such that constraint collection and constraint solving succeed:

$$P \leq_m P_0 \not\leq^\alpha \Xi_c; C \quad C; P_i \rightsquigarrow \Xi_s; P_a \quad \Theta_0, \Xi_c, \Xi_s \vdash P_a$$

and P_a is partially more precise than ψ_a for the solved tags:

$$\theta_a :: \Theta_0, \Xi_c, \Xi_s \sqsubseteq \Theta \quad \theta \equiv \theta_a :: \Theta_0 \sqsubseteq \Theta \quad \Theta \mid \Delta \vdash \theta_a P_a \leq_{\Psi_i, \Psi_a} \psi_a.$$

Proof. By induction on the derivation of $\Psi, \Psi_a \mid \Theta \vdash_m^n (\theta S)[\psi_a/\alpha] \leq Q$, generalising over $\Theta_0, P_0, P_i, P, S, \theta, \psi_a, \Psi', \Psi_i, \Psi_a, \Psi$, and Δ . We discuss two special cases before the induction.

If $P = \alpha$, then $S = \alpha$. By well-formedness of ψ_a and disjointness of Ψ from Ψ' , we have $\Psi = \cdot$. By Lemma 88 with the identity substitution, which introduces the empty tag context, we have

$$P_i \leq_m P_0 \rightsquigarrow \Xi_s; P_a \quad \Theta_0, \Xi_s \vdash P_a \\ \theta_a :: \Theta_0, \Xi_s \sqsubseteq \Theta \quad \theta \equiv \theta_a :: \Theta_0 \sqsubseteq \Theta \quad \Theta \mid \Delta \vdash \theta_a P_a \leq_{\Psi_i, \Psi_a} \psi_a.$$

By LS-VAR, we have

$$\alpha \leq_m P_0 \not\leq^\alpha \text{sub}_m(P_0).$$

By definition of constraint solving, we have

$$\text{sub}_m(P_0); P_i \rightsquigarrow \Xi_s; P_a.$$

Our goal follows by choosing $C = \text{sub}_m(P_0)$ and $\Xi_c = \cdot$.

If $P = \hat{\beta}$, then $\alpha \notin \text{ftv}(P)$, which gives $\Psi_a = \cdot$. By LS-ABSENT and constraint solving, we have

$$\hat{\beta} \leq_m P_0 \not\leq^\alpha \cdot \quad \cdot; P_i \rightsquigarrow P_i.$$

Our goal follows by choosing $C = \cdot$, $\Xi_c = \cdot$, $P_a = P_i$, and $\theta_a = \theta$.

In the following induction we assume $P \neq \alpha$ and that P is not a flexible variable.

Case

$$\frac{\text{S-INFER} \quad \cdot \mid \Theta \vdash (\theta S)[\psi_a/\alpha] \sim Q}{\cdot \mid \Theta \vdash_1^n (\theta S)[\psi_a/\alpha] \leq Q}$$

The solved tag context in the conclusion is empty, so

$$\Psi = \cdot \text{①} \quad \Psi_a = \cdot \text{②}.$$

By $P_0 :^n \text{I}$, we have $P_0 = \hat{\sqcup}$. Thus by LS-VACUOUS and constraint solving, we have

$$P \leq_1 \hat{\sqcup} \not\leq^\alpha \cdot \text{③} \quad \cdot; P_i \rightsquigarrow P_i \text{④}.$$

Our goal follows from ②, ③, and ④ by choosing $\Xi_c = \cdot$, $P_a = P_i$, and $\theta_a = \theta$.

Case

$$\frac{\text{S-CHECK} \quad (\theta S)[\psi_a/\alpha], Q \text{ guarded} \quad \Psi, \Psi_a \mid \Theta \vdash (\theta S)[\psi_a/\alpha] \sim Q \text{ ①}}{\Psi, \Psi_a \mid \Theta \vdash_c^{\text{ty}} (\theta S)[\psi_a/\alpha] \leq Q}$$

By Lemma 96 on the well-formedness premises for P_0 and P_i , the tag context premises derived from $\Psi_i, \Psi_a = \Psi'$, the input precision premise, $\lfloor S \rfloor = P$, $\theta :: \Theta_0 \sqsubseteq \Theta$, ①, and the refinement premise, there are C, Ξ_c, Ξ_s, P_a , and θ_a such that

$$\begin{aligned} P \sim P_0 \not\prec^\alpha \Xi_c; C \text{ ②} \quad & C; P_i \rightsquigarrow \Xi_s; P_a \text{ ③} \quad & \Theta_0, \Xi_c, \Xi_s \vdash P_a \text{ ④} \\ \theta_a :: \Theta_0, \Xi_c, \Xi_s \sqsubseteq \Theta \text{ ⑤} \quad & \theta \equiv \theta_a :: \Theta_0 \sqsubseteq \Theta \text{ ⑥} \quad & \Theta \mid \Delta \vdash \theta_a P_a \leq_{\Psi_i, \Psi_a} \psi_a \text{ ⑦}. \end{aligned}$$

S-CHECK, the guardedness premise and refinement premise give that P, P_0 are guarded types. Then by LS-CHECK on $P \neq \alpha$ and ②, we have

$$P \leq_c P_0 \not\prec^\alpha \Xi_c; C \text{ ⑧}.$$

Our goal follows from ③–⑧.

Case

$$\frac{\text{S-FORALLR} \quad \Psi, \Psi_a \mid \Theta, \beta \vdash_c^{\text{ty}} (\theta S)[\psi_a/\alpha] \leq Q \text{ ①}}{\Psi, \Psi_a \mid \Theta \vdash_c^{\text{ty}} (\theta S)[\psi_a/\alpha] \leq \forall \beta. Q}$$

By the refinement premise, we have $P_0 = \forall \beta. P'_0$ and $\Theta, \beta \vdash \theta P'_0 \triangleright Q \text{ ②}$. From $P_0 :^{\text{ty}} c$ and $P_0 = \forall \beta. P'_0$, we have $P'_0 :^{\text{ty}} c \text{ ⑩}$. By IH on ①, ② and ⑩, context weakening on the other premises, and the input precision premise, there are C, Ξ_c, Ξ_s, P_a , and θ_a such that

$$\begin{aligned} P \leq_c P'_0 \not\prec^\alpha \Xi_c; C \text{ ③} \quad & C; P_i \rightsquigarrow \Xi_s; P_a \text{ ④} \quad & \Theta_0, \beta, \Xi_c, \Xi_s \vdash P_a \text{ ⑤} \\ \theta_a :: \Theta_0, \beta, \Xi_c, \Xi_s \sqsubseteq \Theta, \beta \text{ ⑥} \quad & \theta \equiv \theta_a :: \Theta_0, \beta \sqsubseteq \Theta, \beta \text{ ⑦} \quad & \Theta, \beta \mid \Delta \vdash \theta_a P_a \leq_{\Psi_i, \Psi_a} \psi_a \text{ ⑧}. \end{aligned}$$

By LS-FORALLR on ③, we have

$$P \leq_c \forall \beta. P'_0 \not\prec^\alpha \Xi_c; C \text{ ⑨}.$$

By the well-formedness premise of ψ_a , we have $\beta \notin \text{ftv}(\psi_a)$. Then by ⑧, we have $\beta \notin \text{ftv}(\theta_a P_a)$, and thus $\beta \notin \text{ftv}(P_a)$. Therefore ⑤ holds in the context without β . By definition of metasubstitutions, ⑥ and ⑦ also hold in the context without β . ⑧ holds in the context without β as well. Our goal follows from these judgements after removing β from the contexts, ④, and ⑨.

Case

$$\frac{\text{S-UNIVGHOSTR} \quad (\theta S)[\psi_a/\alpha], Q \text{ guarded} \quad \cdot \mid \Theta \vdash (\theta S)[\psi_a/\alpha] \sim Q}{\cdot \mid \Theta \vdash_c^{\text{sk}} (\theta S)[\psi_a/\alpha] \leq \mathbb{Q}.Q}$$

The solved tag context in the conclusion is empty, so

$$\Psi = \cdot \text{ (1)} \quad \Psi_a = \cdot \text{ (2)}.$$

By $P_0 :^{\text{sk}} \mathbb{C}$, we have $P_0 = \mathbb{C}$. Then by LS-VACUOUS and constraint solving, we have

$$P \leq_c \mathbb{C} \not\leq^\alpha \cdot \text{ (3)}. \quad \cdot; P_i \rightsquigarrow P_i \text{ (4)}.$$

Our goal follows from (2), (3), and (4) by choosing $\Xi_c = \cdot$, $P_a = P_i$, and $\theta_a = \theta$.

Case

S-ARROW

$$\frac{\Psi_1, \Psi_{a,1} \mid \Theta \vdash (\theta S_1)[\psi_a/\alpha] \sim Q_1 \text{ (1)} \quad \Psi_2, \Psi_{a,2} \mid \Theta \vdash_m^n (\theta S_2)[\psi_a/\alpha] \leq Q_2 \text{ (2)}}{\Psi_1, \Psi_{a,1}, \Psi_2, \Psi_{a,2} \mid \Theta \vdash_m^n (\theta S_1)[\psi_a/\alpha] \rightarrow (\theta S_2)[\psi_a/\alpha] \leq Q_1 \rightarrow Q_2}$$

In this case $S = S_1 \rightarrow S_2$. From $\lfloor S \rfloor = P$ we have

$$P = P_1 \rightarrow P_2 \text{ (3)} \quad \lfloor S_1 \rfloor = P_1 \text{ (4)} \quad \lfloor S_2 \rfloor = P_2 \text{ (5)}.$$

By $P_0 :^n \mathbb{F}_m$, we have $P_0 = P_{0,1} \rightarrow P_{0,2}$. Together with the refinement premise, we have

$$\Theta \vdash \theta P_{0,1} \blacktriangleright Q_1 \text{ (6)} \quad \Theta \vdash \theta P_{0,2} \blacktriangleright Q_2 \text{ (7)} \quad P_{0,2} :^n \mathbb{M} \text{ (10)}.$$

S-ARROW gives

$$\Psi = \Psi_1, \Psi_2 \quad \Psi_a = \Psi_{a,1}, \Psi_{a,2} \text{ (8)}.$$

By Lemma 96 on (1), (6), (4), (8), the input precision premise, and the well-formedness premises for $P_{0,1}$ and P_i and the tag context premises derived from $\Psi_i, \Psi_a = \Psi'$, there are $C_1, \Xi_{c,1}, \Xi_{s,1}, P_m$, and θ_1 such that

$$P_1 \sim P_{0,1} \not\leq^\alpha \Xi_{c,1}; C_1 \text{ (9)} \quad C_1; P_i \rightsquigarrow \Xi_{s,1}; P_m \text{ (11)} \quad \Theta_0, \Xi_{c,1}, \Xi_{s,1} \vdash P_m \text{ (12)}$$

$$\theta_1 :: \Theta_0, \Xi_{c,1}, \Xi_{s,1} \sqsubseteq \Theta \text{ (13)} \quad \theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \text{ (14)} \quad \Theta \mid \Delta \vdash \theta_1 P_m \leq_{\Psi_i, \Psi_{a,1}} \psi_a \text{ (15)}.$$

By (14) and equivalence of metasubstitutions, (2) and (7) can be rewritten as

$$\Psi_2, \Psi_{a,2} \mid \Theta \vdash_m^n (\theta_1 S_2)[\psi_a/\alpha] \leq Q_2 \text{ (16)} \quad \Theta \vdash \theta_1 P_{0,2} \blacktriangleright Q_2 \text{ (17)}.$$

By IH on (16) with Ψ_2 as the ambient tag context and $\Psi_i, \Psi_{a,1}$ as the already solved tag context, using (10), (5), (17), (8), (12)–(15), the well-formedness premise for $P_{0,2}$, and the tag context premises ($\Theta \vdash^n \psi_a \# \Psi'$, Ψ_2 is disjoint from Ψ' , and $\Psi_i, \Psi_{a,1}, \Psi_{a,2} = \Psi'$), using $\Theta_0, \Xi_{c,1}, \Xi_{s,1}$ as the input context, there are $C_2, \Xi_{c,2}, \Xi_{s,2}, P_a$, and θ_a such that, after exchanging fresh declarations, we have

$$P_2 \leq_m P_{0,2} \not\leq^\alpha \Xi_{c,2}; C_2 \text{ (18)} \quad C_2; P_m \rightsquigarrow \Xi_{s,2}; P_a \text{ (19)}$$

$$\Theta_0, \Xi_{c,1}, \Xi_{c,2}, \Xi_{s,1}, \Xi_{s,2} \vdash P_a \text{ (20)} \quad \theta_a :: \Theta_0, \Xi_{c,1}, \Xi_{c,2}, \Xi_{s,1}, \Xi_{s,2} \sqsubseteq \Theta \text{ (21)}$$

$$\theta_1 \equiv \theta_a :: \Theta_0, \Xi_{c,1}, \Xi_{s,1} \sqsubseteq \Theta \text{ (22)} \quad \Theta \mid \Delta \vdash \theta_a P_a \leq_{\Psi_i, \Psi_{a,1}, \Psi_{a,2}} \psi_a \text{ (23)}.$$

By LS-ARROW on ⑨ and ⑱, we have

$$P_1 \rightarrow P_2 \leq_{\text{fm}} P_{0,1} \rightarrow P_{0,2} \not\leq^\alpha \Xi_{c,1}, \Xi_{c,2}; C_1, C_2 \text{ ⑳}.$$

After weakening ⑪ with $\Xi_{c,2}$, constraint solving on ⑪ and ⑲, observing that ⑪ does not contain subtyping constraints, gives

$$C_1, C_2; P_i \rightsquigarrow \Xi_{s,1}, \Xi_{s,2}; P_a \text{ ㉑}.$$

By ⑭, ㉒, and transitivity of equivalence of metasubstitutions, we have

$$\theta \equiv \theta_a :: \Theta_0 \sqsubseteq \Theta \text{ ㉓}.$$

Our goal follows by choosing $C = (C_1, C_2)$ and $\Xi_c = (\Xi_{c,1}, \Xi_{c,2})$ and $\Xi_s = (\Xi_{s,1}, \Xi_{s,2})$, and using ③, ⑧, ㉒, ㉑, and ㉓–㉓.

Case

S-FORALL

$$\frac{\begin{array}{c} m \neq I \quad \Psi_b \text{ fresh} \\ \Theta \vdash^n \psi_b \# \Psi_b \text{ ①} \quad \Psi, \Psi_a, \Psi_b \mid \Theta \vdash_m^n ((\theta S')[\psi_a/\alpha])[\psi_b/\beta] \leq Q \text{ ②} \end{array}}{\Psi, \Psi_a \mid \Theta \vdash_m^n (\theta(\forall\beta.S'))[\psi_a/\alpha] \leq Q}$$

In this case $S = \forall\beta.S'$. From $\lfloor S \rfloor = P$ we have $P = \forall\beta.P'$ and

$$\lfloor S' \rfloor = P' \text{ ③}.$$

Since ψ_b is well formed in Θ and α is not in Θ , we have

$$\alpha \notin \text{ftv}(\lfloor \psi_b \rfloor) \text{ ④}.$$

By capture-avoiding substitution and freshness conditions, ② is equivalently

$$\Psi, \Psi_b, \Psi_a \mid \Theta \vdash_m^n ((\theta S')[\psi_b/\beta])[\psi_a/\alpha] \leq Q \text{ ⑤}.$$

By definition of erasure, we have

$$\lfloor S'[\psi_b/\beta] \rfloor = P'[\lfloor \psi_b \rfloor/\beta] \text{ ⑥}.$$

By IH on ⑤ with ⑥, the input precision premise, the well-formedness premises for P_0 and P_i , the tag context premises, freshness of Ψ_b , and refinement premises, there are C, Ξ_c, Ξ_s, P_a , and θ_a such that

$$\begin{array}{lll} P'[\lfloor \psi_b \rfloor/\beta] \leq_m P_0 \not\leq^\alpha \Xi_c; C \text{ ⑦} & C; P_i \rightsquigarrow \Xi_s; P_a \text{ ⑧} & \Theta_0, \Xi_c, \Xi_s \vdash P_a \text{ ⑨} \\ \theta_a :: \Theta_0, \Xi_c, \Xi_s \sqsubseteq \Theta \text{ ⑩} & \theta \equiv \theta_a :: \Theta_0 \sqsubseteq \Theta \text{ ⑪} & \Theta \mid \Delta \vdash \theta_a P_a \leq_{\Psi_i, \Psi_a} \psi_a \text{ ⑫}. \end{array}$$

By the reverse direction of Lemma 81 on ④ and ⑦, we have

$$P' \leq_m P_0 \not\leq^\alpha \Xi_c; C \text{ ⑬}.$$

By LS-FORALL on ⑬, we have

$$\forall\beta.P' \leq_m P_0 \not\leq^\alpha \Xi_c; C \text{ ⑭}.$$

Our goal follows from ⑧–⑭.

Case

S-GHOST

$$\frac{}{\cdot \mid \Theta \vdash_m^{\text{sk}} \hat{\mathbb{A}} \leq Q}$$

The solved tag context in the conclusion is empty, so

$$\Psi = \cdot \text{ ①} \quad \Psi_a = \cdot \text{ ②}.$$

In this case $P = \hat{\mathbb{A}}$ and thus $\alpha \notin \text{ftv}(P)$. By LS-ABSENT and constraint solving, we have

$$\hat{\mathbb{A}} \leq_m P_0 \not\leq^\alpha \cdot \text{ ③}. \quad \cdot; P_i \rightsquigarrow P_i \text{ ④}.$$

Our goal follows from $P = \hat{\mathbb{A}}$, ②, ③, and ④ by choosing $\Xi_c = \cdot$, $P_a = P_i$, and $\theta_a = \theta$.

Case

S-GHOSTG

$$\frac{}{\cdot \mid \Theta \vdash_m^{\text{sk}} \hat{\mathbb{A}} \leq \hat{\mathbb{A}}}$$

Similar to the case of S-GHOST.

Cases

S-UNIVGHOSTL

$$m \neq \text{I} \quad \Psi, \Psi_a \mid \Theta \vdash_m^{\text{sk}} (\theta S')[\psi_a/\alpha] \leq Q \text{ ①}$$

$$\Psi, \Psi_a \mid \Theta \vdash_m^{\text{sk}} \hat{\mathbb{A}}.(\theta S')[\psi_a/\alpha] \leq Q$$

S-UNIVGHOSTLG

$$m \neq \text{I} \quad \Psi, \Psi_a \mid \Theta \vdash_m^{\text{sk}} (\theta S')[\psi_a/\alpha] \leq Q \text{ ①}$$

$$\Psi, \Psi_a \mid \Theta \vdash_m^{\text{sk}} \hat{\mathbb{A}}.(\theta S')[\psi_a/\alpha] \leq Q$$

In these cases $S = \hat{\mathbb{A}}.S'$ or $S = \hat{\mathbb{A}}.S'$, respectively. From $\llbracket S \rrbracket = P$ we have $P = \hat{\mathbb{A}}.P'$ and

$$\llbracket S' \rrbracket = P' \text{ ②}.$$

By IH on ① with ②, $P_0 :^{\text{sk}} m$, the input precision premise, and the refinement premise, there are C, Ξ_c, Ξ_s, P_a , and θ_a such that

$$P' \leq_m P_0 \not\leq^\alpha \Xi_c; C \text{ ③} \quad C; P_i \rightsquigarrow \Xi_s; P_a \text{ ④} \quad \Theta_0, \Xi_c, \Xi_s \vdash P_a \text{ ⑤}$$

$$\theta_a :: \Theta_0, \Xi_c, \Xi_s \sqsubseteq \Theta \text{ ⑥} \quad \theta \equiv \theta_a :: \Theta_0 \sqsubseteq \Theta \text{ ⑦} \quad \Theta \mid \Delta \vdash \theta_a P_a \leq_{\Psi_i, \Psi_a} \psi_a \text{ ⑧}.$$

By LS-UNIVGHOSTL on ③, we have

$$\hat{\mathbb{A}}.P' \leq_m P_0 \not\leq^\alpha \Xi_c; C \text{ ⑨}.$$

Our goal follows from ④–⑨.

□

Lemma 90 (Look completeness). *Suppose*

$$\Theta_0 \vdash P_0 \quad P_0 :^n m \quad \lfloor S \rfloor = P \quad \theta :: \Theta_0 \sqsubseteq \Theta.$$

If ψ_a is a valid instantiation for declarative S-FORALL with

$$\begin{aligned} \Theta \vdash^n \psi_a \# \Psi_a \quad & \Psi \text{ is disjoint from } \Psi_a \\ \Psi, \Psi_a \mid \Theta \vdash_m^n (\theta S)[\psi_a/\alpha] \leq Q \quad & \Theta \vdash \theta P_0 \blacktriangleright Q, \end{aligned}$$

then there exist P_a, Ξ , and θ_a such that look succeeds with

$$P \leq_m^n P_0 \not\leq^\alpha \Xi; P_a \quad \Theta_0, \Xi \vdash P_a$$

and look gives a more precise skeleton:

$$\theta_a :: \Theta_0, \Xi \sqsubseteq \Theta \quad \theta \equiv \theta_a :: \Theta_0 \sqsubseteq \Theta \quad \Theta \vdash \theta_a P_a \leq \lfloor \psi_a \rfloor$$

Proof. By Lemma 89 on the subtyping premise, refinement premise, and $\Theta \vdash \theta \sqsubseteq \leq \psi_a$ with $\Psi' = \Psi_a$, $\Psi_i = \cdot, \cdot, \Psi_a = \Psi_a$, and $P_i = \sqsubseteq$, there are C, Ξ_c, Ξ_s, P_s , and θ_s such that

$$\begin{aligned} P \leq_m P_0 \not\leq^\alpha \Xi_c; C \quad & C; \sqsubseteq \rightsquigarrow \Xi_s; P_s \quad & \Theta_0, \Xi_c, \Xi_s \vdash P_s \quad (3) \\ \theta_s :: \Theta_0, \Xi_c, \Xi_s \sqsubseteq \Theta \quad & \theta \equiv \theta_s :: \Theta_0 \sqsubseteq \Theta \quad (5) \quad & \Theta \vdash \theta_s P_s \leq_{\Psi_a} \psi_a \quad (6). \end{aligned}$$

By Lemma 86 on the well-formedness premise for ψ_a and (6), we have

$$\Theta \vdash \theta_s P_s \leq \lfloor \psi_a \rfloor \quad (7).$$

Let $(\Xi'; P_a) = \text{guess}^n(P_s)$ (8). By induction on the definition of $\text{guess}^n(P_s)$, using (3), (4), (6), and (7), there is θ_a such that

$$\begin{aligned} \Theta_0, \Xi_c, \Xi_s, \Xi' \vdash P_a \quad & \theta_a :: \Theta_0, \Xi_c, \Xi_s, \Xi' \sqsubseteq \Theta \quad (9) \quad (10) \\ \theta_s \equiv \theta_a :: \Theta_0, \Xi_c, \Xi_s \sqsubseteq \Theta \quad & \Theta \vdash \theta_a P_a \leq \lfloor \psi_a \rfloor \quad (11) \quad (12). \end{aligned}$$

When $n = \text{ty}$, we construct θ_a by extending θ_s to map each flexible variable in Ξ' to the corresponding type in $\lfloor \psi_a \rfloor$ based on its position in P_a in order to satisfy (12); when $n = \text{sk}$, θ_a is simply θ_s . For each quantifier in $\lfloor \psi_a \rfloor$, (6) ensures that it does not correspond to a universal ghost in P_s , since all tags are in Ψ_a .

By the definition of look judgement on (1), (2), and (8), we have

$$P \leq_m^n P_0 \not\leq^\alpha \Xi_c, \Xi_s, \Xi'; P_a \quad (13).$$

By (5), (11), and transitivity of equivalence of metasubstitutions, we have

$$\theta \equiv \theta_a :: \Theta_0 \sqsubseteq \Theta \quad (14).$$

Our goal follows by choosing $\Xi = \Xi_c, \Xi_s, \Xi'$ and using (9), (10), and (12)–(14). $\square$

Lemma 91 (Completeness of flexible-variable subtyping). *Given a subtyping problem $(\Theta_0; n; F_m; P_0; \hat{\alpha})$. If Ξ only contains declarations of flexible variables and*

$$\theta :: \Theta_0, \Xi \sqsubseteq \Theta \quad \Psi \mid \Theta \vdash_{F_m}^n \theta \hat{\alpha} \leq Q \quad \Theta \vdash \theta P_0 \blacktriangleright Q$$

then there exist P_1, Θ_1 , and θ_1 such that

$$\Theta_0 \mid \Xi [P_0]_{F_m}^n \hat{\alpha} \leq P_1 \dashv \Theta_1 \quad \theta_1 :: \Theta_1 \sqsubseteq \Theta$$

and

$$\Theta \vdash \theta_1 P_1 \leq Q \quad \theta \equiv \theta_1 :: \Theta_0, \Xi \sqsubseteq \Theta.$$

Proof. By induction on the contexts $\Theta_0 \mid \Xi$ of the subtyping judgement, mutually with the generalised version of Theorem 28 below. We perform a case analysis on which algorithmic subtyping rule is used first for the given contexts. The only two interesting cases are SI-FLEX-SOLVE and SI-FLEX-ASSIGN; we omit other cases which follow from routine applications of the IH.

Case

$$\begin{array}{c} \text{SI-FLEX-SOLVE} \\ \Theta_0, \hat{\alpha}, \hat{\beta}, \hat{\alpha}_0 := \hat{\alpha} \rightarrow \hat{\beta}, \Xi [P_0]_{F_m}^n \hat{\alpha} \rightarrow \hat{\beta} \leq P_1 \dashv \Theta_1 \\ \hline \Theta_0, \hat{\alpha}_0 \mid \Xi [P_0]_{F_m}^n \hat{\alpha}_0 \leq P_1 \dashv \Theta_1 \end{array}$$

The context has the form $\Theta_0, \hat{\alpha}_0 \mid \Xi$. The metasubstitution premise is

$$\theta :: \Theta_0, \hat{\alpha}_0, \Xi \sqsubseteq \Theta \text{ ①}.$$

By definition of metasubstitution, there is a monotype $\Theta \vdash \tau$ such that

$$\Theta \vdash \theta \hat{\alpha}_0 = \tau \text{ ②}.$$

The declarative subtyping premise gives

$$\Psi \mid \Theta \vdash_{F_m}^n \tau \leq Q \text{ ③}$$

either directly (when θ substitutes $\hat{\alpha}_0$ to τ) or from S-FLEXMONO (when $\Theta \ni \hat{\alpha}_0 := \tau$). Inversion on ③ with S-ARROW gives monotypes τ_1 and τ_2 such that

$$\Theta \vdash \tau \equiv \tau_1 \rightarrow \tau_2 \text{ ④} \quad \Psi \mid \Theta \vdash_{F_m}^n \tau_1 \rightarrow \tau_2 \leq Q \text{ ⑤}.$$

Let $\hat{\alpha}$ and $\hat{\beta}$ be fresh flexible variables. By ① and ④, we can construct a metasubstitution θ_s which maps $\hat{\alpha}$ to τ_1 and $\hat{\beta}$ to τ_2 , such that

$$\theta_s :: \Theta_0, \hat{\alpha}, \hat{\beta}, \hat{\alpha}_0 := \hat{\alpha} \rightarrow \hat{\beta}, \Xi \sqsubseteq \Theta \text{ ⑥}$$

$$\theta \equiv \theta_s :: \Theta_0, \hat{\alpha}_0, \Xi \sqsubseteq \Theta \text{ ⑦}.$$

By ⑦ and the refinement premise, we have

$$\Theta \vdash \theta_s P_0 \blacktriangleright Q \text{ ⑩}.$$

Thus $(\theta_s :: \Theta_0, \hat{\alpha}, \hat{\beta}, \hat{\alpha}_0 := \hat{\alpha} \rightarrow \hat{\beta}, \Xi \sqsubseteq \Theta; Q)$ is a solution to the subtyping problem $(\Theta_0, \hat{\alpha}, \hat{\beta}, \hat{\alpha}_0 := \hat{\alpha} \rightarrow \hat{\beta}, \Xi; n; \text{Fm}; P_0; \hat{\alpha} \rightarrow \hat{\beta})$. By the IH of generalised Theorem 28 on this solution, we have

$$\Theta_0, \hat{\alpha}, \hat{\beta}, \hat{\alpha}_0 := \hat{\alpha} \rightarrow \hat{\beta}, \Xi \vdash_{\text{Fm}}^n \hat{\alpha} \rightarrow \hat{\beta} \leq P_1 \dashv \Theta_1 \quad (8)$$

$$\theta_1 :: \Theta_1 \sqsubseteq \Theta \quad (9)$$

$$\Theta \vdash \theta_1 P_1 \leq Q \quad (11)$$

$$\theta_s \equiv \theta_1 :: \Theta_0, \hat{\alpha}, \hat{\beta}, \hat{\alpha}_0 := \hat{\alpha} \rightarrow \hat{\beta}, \Xi \sqsubseteq \Theta \quad (12).$$

By algorithmic SI-FLEX-SOLVE on (8), we have

$$\Theta_0, \hat{\alpha}_0 \mid \Xi \vdash_{\text{Fm}}^n \hat{\alpha}_0 \leq P_1 \dashv \Theta_1 \quad (13).$$

By (7) and (12), we have

$$\theta \equiv \theta_1 :: \Theta_0, \hat{\alpha}_0, \Xi \sqsubseteq \Theta \quad (14).$$

Our goal follows from (13), (9), (11), and (14).

Case

SI-FLEX-ASSIGN

$$\frac{\Theta_0, \Xi \vdash_{\text{Fm}}^n [P_0[\tau/\hat{\beta}]] \hat{\alpha}[\tau/\hat{\beta}] \leq P_1 \dashv \Theta_1}{\Theta_0, \hat{\beta} := \tau \mid \Xi \vdash_{\text{Fm}}^n \hat{\alpha} \leq P_1 \dashv \Theta_1, \hat{\beta} := \tau}$$

The context has the form $\Theta_0, \hat{\beta} := \tau \mid \Xi$. The metasubstitution premise is

$$\theta :: \Theta_0, \hat{\beta} := \tau, \Xi \sqsubseteq \Theta \quad (1).$$

By definition of metasubstitution, since Ξ only contains declarations, we have

$$\theta :: \Theta_0, \Xi \sqsubseteq \Theta \quad (2)$$

Therefore the declarative subtyping and refinement premises can be rewritten as

$$\Psi \mid \Theta \vdash_{\text{Fm}}^n \theta(\hat{\alpha}[\tau/\hat{\beta}]) \leq Q \quad (5) \quad \Theta \vdash \theta(P_0[\tau/\hat{\beta}]) \blacktriangleright Q \quad (6).$$

By the IH of generalised Theorem 28 on (2), (5), and (6), we have

$$\Theta_0, \Xi \vdash_{\text{Fm}}^n [P_0[\tau/\hat{\beta}]] \hat{\alpha}[\tau/\hat{\beta}] \leq P_1 \dashv \Theta_1 \quad (7)$$

$$\theta_1 :: \Theta_1 \sqsubseteq \Theta \quad (8)$$

$$\Theta \vdash \theta_1 P_1 \leq Q \quad (9)$$

$$\theta \equiv \theta_1 :: \Theta_0, \Xi \sqsubseteq \Theta \quad (10).$$

By algorithmic SI-FLEX-ASSIGN on (7), we have

$$\Theta_0, \hat{\beta} := \tau \mid \Xi \vdash_{\text{Fm}}^n \hat{\alpha} \leq P_1 \dashv \Theta_1, \hat{\beta} := \tau \quad (11).$$

By definition of metasubstitution, we can extend the source context of θ_1 with $\hat{\beta} := \tau$ to obtain

$$\theta_1 :: \Theta_1, \hat{\beta} := \tau \sqsubseteq \Theta \quad (12) \quad \theta \equiv \theta_1 :: \Theta_0, \hat{\beta} := \tau, \Xi \sqsubseteq \Theta \quad (13).$$

Our goal follows from ⑪, ⑫, ⑨, and ⑬.

□

Theorem 28 (Completeness and optimality of subtyping). *Given a subtyping problem $(\Theta_0; n; m; P_0; P)$, if there exists a solution, $\Theta_0 [P_0]_{\mathfrak{m}}^n P \leq P_1 \dashv \Theta_1$ gives its optimal solution $(\Theta_0 \sqsubseteq \Theta_1; P_1)$.*

Proof. We first generalise the theorem to allow non-empty tag contexts Ψ . It is enough to prove that given a subtyping problem $(\Theta_0; n; m; P_0; P)$, a tag context Ψ , an unresolved skeleton S , and a pair $(\theta :: \Theta_0 \sqsubseteq \Theta; Q)$, if

$$\llbracket S \rrbracket = P \quad \Psi \mid \Theta \vdash_{\mathfrak{m}}^n \theta S \leq Q \quad \Theta \vdash \theta P_0 \triangleright Q$$

then there exist P_1 , Θ_1 , and θ_1 such that

$$\Theta_0 [P_0]_{\mathfrak{m}}^n P \leq P_1 \dashv \Theta_1 \quad \theta_1 :: \Theta_1 \sqsubseteq \Theta \quad \Theta \vdash \theta_1 P_1 \leq Q \quad \theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta.$$

The theorem follows from this generalised version by taking $S = P$ and $\Psi = \cdot$. In the following proof, we first perform a case analysis on whether P is a flexible variable in function mode. In the remaining cases, we perform lexicographic induction on the derivation height of declarative subtyping $\Psi \mid \Theta \vdash_{\mathfrak{m}}^n \theta S \leq Q$ and the size of tag context Ψ .

Case $P = \hat{\alpha}$ and $\mathfrak{m} = \mathfrak{Fm}'$. By Lemma 91 with $\Xi = \cdot$, we have

$$\Theta_0 \mid \cdot [P_0]_{\mathfrak{Fm}'}^n \hat{\alpha} \leq P_1 \dashv \Theta_1 \text{ ①} \quad \theta_1 :: \Theta_1 \sqsubseteq \Theta \text{ ②}$$

and

$$\Theta \vdash \theta_1 P_1 \leq Q \text{ ③} \quad \theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \text{ ④}.$$

By algorithmic SI-FLEXMONO on ①, we have

$$\Theta_0 [P_0]_{\mathfrak{Fm}'}^n \hat{\alpha} \leq P_1 \dashv \Theta_1 \text{ ⑤}.$$

Our goal follows from ⑤, ②, ③, and ④.

Case

$$\begin{array}{c} \text{S-INFER} \\ \cdot \mid \Theta \vdash \theta S \sim Q \text{ ①} \\ \hline \cdot \mid \Theta \vdash_{\mathfrak{I}}^n \theta S \leq Q \end{array}$$

In this case $\mathfrak{m} = \mathfrak{I}$, $\Psi = \cdot$, and by $P_0 :^n \mathfrak{I}$ we have $P_0 = \hat{\lambda}$. By SI-INFER, we have

$$\Theta_0 [\hat{\lambda}]_{\mathfrak{I}}^n P \leq P \dashv \Theta_0 \text{ ②}.$$

The erasure premise is $\llbracket S \rrbracket = P$ ⑩. The consistency premise ①, together with ⑩ and Lemma 26, gives

$$\Theta \vdash \theta P \leq Q \text{ ③}.$$

Our goal follows from ②, $\theta :: \Theta_0 \sqsubseteq \Theta$, ③, and $\theta \equiv \theta :: \Theta_0 \sqsubseteq \Theta$.

Case

$$\frac{\text{S-CHECK} \quad \theta S, Q \text{ guarded} \quad \Psi \mid \Theta \vdash \theta S \sim Q \text{ ①}}{\Psi \mid \Theta \vdash_c^{\text{ty}} \theta S \leq Q}$$

We have that S and Q are concrete types. The erasure and refinement premises are

$$\lfloor S \rfloor = P \text{ ⑩} \quad \Theta \vdash \theta P_0 \blacktriangleright Q \text{ ⑪}.$$

By ① and ⑪, $(\theta :: \Theta_0 \sqsubseteq \Theta; Q)$ is a solution to the consistency problem $(\Theta_0; P_0; P)$. Then by Lemma 26, we have

$$\begin{aligned} \Theta_0 \lfloor P_0 \rfloor \vdash P &\sim P'_0 \dashv \Theta_1 \text{ ②} \\ \theta_1 :: \Theta_1 &\sqsubseteq \Theta \text{ ③} \\ \Theta \vdash \theta_1 P'_0 &\leq Q \text{ ④} \\ \theta \equiv \theta_1 :: \Theta_0 &\sqsubseteq \Theta \text{ ⑤} \end{aligned}$$

By ⑪ and ⑩, we have P, P_0 guarded and P, P_0 are concrete types. Since P_0 is a concrete type, ② gives that P'_0 is equivalent to P_0 . Together with ③ and ④, we have

$$\Theta \vdash \theta_1 P_0 \leq Q \text{ ⑥}.$$

Then algorithmic SI-CHECK on ② gives

$$\Theta_0 \lfloor P_0 \rfloor \vdash_c^{\text{ty}} P \leq P_0 \dashv \Theta_1 \text{ ⑦}.$$

Our goal follows from ⑦, ③, ⑥, and ⑤.

Case

$$\frac{\text{S-FORALLR} \quad \Psi \mid \Theta, \alpha \vdash_c^{\text{ty}} \theta S \leq Q \text{ ①}}{\Psi \mid \Theta \vdash_c^{\text{ty}} \theta S \leq \forall \alpha. Q}$$

We have that S and Q are concrete types. The refinement premise is $\Theta \vdash \theta P_0 \blacktriangleright \forall \alpha. Q$, by which we have $P_0 = \forall \alpha. P'_0$ and $\Theta, \alpha \vdash \theta P'_0 \blacktriangleright Q$ ②. By definition of metasubstitution, we have $\theta :: \Theta_0, \alpha \sqsubseteq \Theta, \alpha$ ③.

IH on ① with ② and ③ gives

$$\begin{aligned} \Theta_0, \alpha \lfloor P'_0 \rfloor \vdash_c^{\text{ty}} P &\leq P_1 \dashv \Theta_1, \alpha, \Xi \text{ ④} \\ \theta_1 :: \Theta_1, \alpha, \Xi &\sqsubseteq \Theta, \alpha \text{ ⑤} \\ \Theta, \alpha \vdash \theta_1 P_1 &\leq Q \text{ ⑥} \\ \theta \equiv \theta_1 :: \Theta_0, \alpha &\sqsubseteq \Theta, \alpha \text{ ⑦} \end{aligned}$$

We have P'_0, P, P_1 are concrete types. Thus ②, ⑥, and ⑦ together give that P'_0 and P_1 are equivalent through Q . Then ④ gives

$$\Theta_0, \alpha \vdash_c^{ty} P \leq P'_0 \dashv \Theta_1, \alpha, \Xi \text{ ⑫}$$

By algorithmic SI-FORALLR on ⑫, we have

$$\Theta_0 \vdash_c^{ty} P \leq \forall \alpha. P'_0 \dashv \Theta_1 \text{ ⑧}.$$

By ⑤, we have $\theta_1 :: \Theta_1 \sqsubseteq \Theta$ ⑨. By ⑥ and type equivalence, we have $\Theta \vdash \theta_1(\forall \alpha. P'_0) \leq \forall \alpha. Q$ ⑩. By ⑦, we have $\theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta$ ⑪. Our goal follows from ⑧, ⑨, ⑩, and ⑪.

Case

$$\frac{\text{S-UNIVGHOSTR} \quad \theta S, Q \text{ guarded} \quad \cdot \mid \Theta \vdash \theta S \sim Q \text{ ①}}{\cdot \mid \Theta \vdash_c^{sk} \theta S \leq \hat{\mathbb{A}}.Q}$$

In this case $m = c$, $n = sk$, and by $P_0 :^{sk} c$ we have $P_0 = \hat{\mathbb{A}}$. By SI-UNIVGHOSTR, we have

$$\Theta_0 \vdash_c^{sk} P \leq \hat{\mathbb{A}}.P \dashv \Theta_0 \text{ ②}.$$

The erasure premise is $[S] = P$ ⑩. The consistency premise ①, together with ⑩ and Lemma 26, gives

$$\Theta \vdash \theta P \leq Q \text{ ③}.$$

Thus

$$\Theta \vdash \theta(\hat{\mathbb{A}}.P) \leq \hat{\mathbb{A}}.Q \text{ ④}.$$

Our goal follows from ②, $\theta :: \Theta_0 \sqsubseteq \Theta$, ④, and $\theta \equiv \theta :: \Theta_0 \sqsubseteq \Theta$.

Case

$$\frac{\text{S-ARROW} \quad \Psi_1 \mid \Theta \vdash \theta S_1 \sim Q_1 \text{ ①} \quad \Psi_2 \mid \Theta \vdash_m^n \theta S_2 \leq Q_2 \text{ ②}}{\Psi_1, \Psi_2 \mid \Theta \vdash_m^n \theta S_1 \rightarrow \theta S_2 \leq Q_1 \rightarrow Q_2}$$

We have $S = S_1 \rightarrow S_2$. By mode Fm , we have $P_0 = P'_1 \rightarrow P'_2$. The erasure and refinement premises are

$$[S_1 \rightarrow S_2] = P \text{ ⑩} \quad \Theta \vdash \theta(P'_1 \rightarrow P'_2) \blacktriangleright Q_1 \rightarrow Q_2 \text{ ⑪}.$$

By ⑩, we have $P = P_1 \rightarrow P_2$. By inversion on ⑪, we have

$$\Theta \vdash \theta P'_1 \blacktriangleright Q_1 \text{ ③} \quad \Theta \vdash \theta P'_2 \blacktriangleright Q_2 \text{ ④}.$$

By ① and ③ and erasure, $(\theta :: \Theta_0 \sqsubseteq \Theta; Q_1)$ is a solution to the consistency problem $(\Theta_0; P'_1; P_1)$. Then by Lemma 26, we have

$$\begin{aligned} \Theta_0 [P'_1] \vdash P_1 \sim Q'_1 \dashv \Theta_1 & \text{ ⑤} \\ \theta_1 :: \Theta_1 \sqsubseteq \Theta & \text{ ⑥} \\ \Theta \vdash \theta_1 Q'_1 \leq Q_1 & \text{ ⑦} \\ \theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta & \text{ ⑧} \end{aligned}$$

By ⑧ and ④, we have $\Theta \vdash \theta_1 P'_2 \blacktriangleright Q_2$ ⑨. IH on ② with ⑥, ⑧, and ⑨ gives

$$\begin{aligned} \Theta_1 [P'_2] \vdash_m P_2 \leq Q'_2 \dashv \Theta_2 & \text{ ⑫} \\ \theta_2 :: \Theta_2 \sqsubseteq \Theta & \text{ ⑬} \\ \Theta \vdash \theta_2 Q'_2 \leq Q_2 & \text{ ⑭} \\ \theta_1 \equiv \theta_2 :: \Theta_1 \sqsubseteq \Theta & \text{ ⑮} \end{aligned}$$

By algorithmic SI-ARROW on ⑤ and ⑫, we have

$$\Theta_0 [P'_1 \rightarrow P'_2] \vdash_{\text{Fm}} P_1 \rightarrow P_2 \leq Q'_1 \rightarrow Q'_2 \dashv \Theta_2 \text{ ⑯}.$$

By ⑦, ⑮, and ⑭, we have $\Theta \vdash \theta_2(Q'_1 \rightarrow Q'_2) \leq Q_1 \rightarrow Q_2$ ⑰. By ⑧ and ⑮, we have $\theta \equiv \theta_2 :: \Theta_0 \sqsubseteq \Theta$ ⑱. Our goal follows from ⑯, ⑬, ⑰, and ⑱.

Case

S-FORALL

$$\frac{\begin{array}{c} m \neq I \quad \Psi_a \text{ fresh} \quad \Theta \vdash^n \psi_a \# \Psi_a \text{ ①} \quad \Psi, \Psi_a \mid \Theta \vdash_m^n (\theta S)[\psi_a/\alpha] \leq Q \text{ ②} \end{array}}{\Psi \mid \Theta \vdash_m^n \theta(\forall^\delta \alpha. S) \leq Q}$$

Since declarative subtyping is congruent, we assume WLOG that P_0 guarded or $P_0 = \mathbb{Q}$; otherwise we apply S-FORALLR first. Since $\lfloor \forall^\delta \alpha. S \rfloor = \forall \alpha. P$, we have $\lfloor S \rfloor = P$. The refinement premise is $\Theta \vdash \theta P_0 \blacktriangleright Q$ ⑩. By Lemma 90 on $P_0 :^n m$, $\lfloor S \rfloor = P$, ①, the freshness side condition, ②, and ⑩, there are P_a , Ξ , and θ_a such that

$$\begin{aligned} P & \leq_m^n P_0 \not\leq^\alpha \Xi; P_a \text{ ③} \\ \Theta_0, \Xi \vdash P_a & \text{ ④} \\ \theta_a :: \Theta_0, \Xi \sqsubseteq \Theta & \text{ ⑤} \\ \theta \equiv \theta_a :: \Theta_0 \sqsubseteq \Theta & \text{ ⑥} \\ \Theta \vdash \theta_a P_a \leq \lfloor \psi_a \rfloor & \text{ ⑦} \end{aligned}$$

By ⑥, ② is equivalently

$$\Psi, \Psi_a \mid \Theta \vdash_m^n (\theta_a S)[\psi_a/\alpha] \leq Q \text{ ⑧}.$$

By ③, we know that P_a is a concrete type when $n = \text{ty}$. Thus by Lemma 87 on ⑦, ①, the freshness side condition, and ⑧, we have

$$\Psi \mid \Theta \vdash_m^n (\theta_a S)[\theta_a P_a/\alpha] \leq Q \text{ ⑨}.$$

By ⑥ and ⑩, we have

$$\Theta \vdash \theta_a P_0 \blacktriangleright Q \text{ ⑪}.$$

IH on ⑨ with ⑤ and ⑪ gives

$$\Theta_0, \Xi [P_0]_m^n P[P_a/\alpha] \leq Q' + \Theta_1 \text{ ⑫}$$

$$\theta_1 :: \Theta_1 \sqsubseteq \Theta \text{ ⑬}$$

$$\Theta \vdash \theta_1 Q' \leq Q \text{ ⑭}$$

$$\theta_a \equiv \theta_1 :: \Theta_0, \Xi \sqsubseteq \Theta \text{ ⑮}$$

By algorithmic SI-FORALL on the side conditions, ③, ④, and ⑫, we have

$$\Theta_0 [P_0]_m^n \forall \alpha. P \leq Q' + \Theta_1 \text{ ⑯}.$$

By ⑥ and ⑮, we have

$$\theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \text{ ⑰}.$$

Our goal follows from ⑯, ⑬, ⑭, and ⑰.

Case

S-GHOST

$$\frac{}{\cdot \mid \Theta \vdash_m^{\text{sk}} \mathbb{A} \leq Q}$$

In this case $P = \mathbb{A}$. By algorithmic SI-GHOST, we have

$$\Theta_0 [P_0]_m^{\text{sk}} \mathbb{A} \leq P_0 + \Theta_0 \text{ ①}.$$

The refinement premise is $\Theta \vdash \theta P_0 \blacktriangleright Q$ ③, which by definition gives

$$\Theta \vdash \theta P_0 \leq Q \text{ ②}.$$

Our goal follows from ①, $\theta :: \Theta_0 \sqsubseteq \Theta$, ②, and $\theta \equiv \theta :: \Theta_0 \sqsubseteq \Theta$.

Case

S-GHOSTG

$$\frac{}{\cdot \mid \Theta \vdash_m^{\text{sk}} \mathbb{A} \leq \mathbb{A}}$$

The refinement premise is $\Theta \vdash \theta P_0 \blacktriangleright \mathbb{A}$, by which and $P_0 :^{\text{sk}}_m$ we know $P_0 = \mathbb{A}$.

We also have $P = \mathbb{A}$. By algorithmic SI-GHOST, we have

$$\Theta_0 [\mathbb{A}]_m^{\text{sk}} \mathbb{A} \leq \mathbb{A} + \Theta_0 \text{ ①}.$$

Our goal follows from ①, $\theta :: \Theta_0 \sqsubseteq \Theta$, $\Theta \vdash \theta \mathbb{A} \leq \mathbb{A}$, and $\theta \equiv \theta :: \Theta_0 \sqsubseteq \Theta$.

Cases

S-UNIVGHOSTL

$$\frac{m \neq I \quad \Psi \mid \Theta \vdash_m^{\text{sk}} \theta S \leq Q \text{ ①}}{\Psi \mid \Theta \vdash_m^{\text{sk}} \mathbb{A}.\theta S \leq Q}$$

S-UNIVGHOSTLG

$$\frac{m \neq I \quad \Psi \mid \Theta \vdash_m^{\text{sk}} \theta S \leq Q \text{ ①}}{\Psi \mid \Theta \vdash_m^{\text{sk}} \mathbb{A}.\theta S \leq Q}$$

Since both $\lfloor \mathbb{A}.S \rfloor$ and $\lfloor \mathbb{A}.S \rfloor$ equal $\mathbb{A}.P$, IH on ① gives

$$\Theta_0 [P_0] \vdash_m^{sk} P \leq Q' \dashv \Theta_1 \text{ ②}$$

$$\theta_1 :: \Theta_1 \sqsubseteq \Theta \text{ ③}$$

$$\Theta \vdash \theta_1 Q' \leq Q \text{ ④}$$

$$\theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \text{ ⑤}$$

By algorithmic SI-UNIVGHOSTL on ② and $m \neq I$, we have

$$\Theta_0 [P_0] \vdash_m^{sk} \mathbb{A}.P \leq Q' \dashv \Theta_1 \text{ ⑥}.$$

Our goal follows from ⑥, ③, ④, and ⑤.

□

D.5 Consistency

D.5.1 Lemmas for subtyping

We prove lemmas about consistency that are only used to prove soundness and completeness of subtyping in Appendix D.4. They are not directly needed for proving the soundness and completeness of consistency itself.

Lemma 92 (Soundness of decorated consistency). *If $\Psi \mid \Theta_0 [P_0] \vdash S \sim Q \dashv \Theta_1$, then*

$$\Theta_0 \sqsubseteq \Theta_1 \quad \Theta_1 \vdash P_0 \blacktriangleright Q \quad \Psi \mid \Theta_1 \vdash S \sim Q.$$

Proof. The proof is very similar to the proof of Lemma 25 in Appendix D.5.2. We first augment the definition of a consistency problem to $(\Delta_1; \Delta_2; \Psi; \Theta_0; P_0; S)$ where Δ_1 and Δ_2 are contexts of rigid type variables present in Θ_0 that no flexible variables depend on. It is enough to prove that, if $\Psi \mid \Theta_0 [P_0] \vdash_{\Delta_1 \mid \Delta_2} S \sim Q \dashv \Theta_1$, then $\Theta_0 \sqsubseteq \Theta_1$ and $\Psi \mid \Theta_1 \vdash S \sim Q$ and $\Theta_1 \mid \Delta_1 \vdash P_0 \blacktriangleright Q$ and $\Theta_1 \mid \Delta_2 \vdash \lfloor S \rfloor \blacktriangleright Q$. The main addition is to show that the tag context used by the algorithmic consistency judgement is the same as the tag context used by the declarative consistency judgement. This follows from IHs and the corresponding declarative rule in most cases. In U-FORALLINH, where we add a potential tag δ to the tag context, we use the corresponding declarative rule C-FORALLINH, which adds the same potential tag. There are three other non-trivial cases: U-GHOSTG, U-GHOSTR, and U-UNIVGHOSTR.

Case U-GHOSTG. We use C-GHOSTG.

Case U-GHOSTR. By a subsidiary induction on S , using C-FORALLSYN for quantifiers and C-UNIVGHOST or C-UNIVGHOSTG for universal ghosts, we construct $\cdot \mid \Theta_1 \vdash S \sim \lfloor S \rfloor$.

Case U-UNIVGHOSTR. There is no corresponding declarative rule for this case. We directly use IH. $\square$

Lemma 93 (Decorating algorithmic consistency). *If $\Theta_0 [P_0] \vdash P \sim Q \dashv \Theta_1$, then $\cdot \mid \Theta_0 [P_0] \vdash P \dot{\sim} Q \dashv \Theta_1$. Consequently, if $\Theta_0 \vdash P \sim P_0 \dashv \Theta_1$, then $\cdot \mid \Theta_0 \vdash P \dot{\sim} P_0 \dashv \Theta_1$.*

Proof. By straightforward induction on the ordinary algorithmic consistency derivation. $\square$

Lemma 94 (Consistency decoration). *Suppose*

$$P \sim P_0 \not\stackrel{\alpha}{\sim} \Xi_c; C \quad C; \psi_a \rightsquigarrow \Psi_a \quad [S] = P \quad \alpha \notin \text{ftv}([\psi_a]).$$

If

$$\Psi \mid \Theta_0 [P_0] \vdash S[[\psi_a] / \alpha] \dot{\sim} Q \dashv \Theta_1,$$

then

$$\Psi, \Psi_a \mid \Theta_0 [P_0] \vdash S[\psi_a / \alpha] \dot{\sim} Q \dashv \Theta_1.$$

Proof. Induction on the constraint-collection derivation $P \sim P_0 \not\stackrel{\alpha}{\sim} \Xi_c; C$, generalising over S, Ψ, Θ_0 , and the decorated consistency premise. We allow arbitrary input context Θ_0 as long as the decorated consistency premise holds. We omit the auxiliary $\Delta_1 \mid \Delta_2$ contexts in decorated consistency judgements throughout this proof. As a result, the well-formedness condition of the consistency premise implicitly requires that flexible variables in Ξ_c must be present in Θ_0 if α appears in S .

Case

LE-VAR

$$\frac{}{\alpha \sim P_0 \not\stackrel{\alpha}{\sim} \text{eq}(P_0)}$$

The decorated constraint solving premise is

$$\text{eq}(P_0); \psi_a \rightsquigarrow \Psi_a \text{ (4)}$$

By inversion on (4), we have

$$\psi_a \sim P_0 \rightsquigarrow \Psi_a \text{ (5)}.$$

Since $[S] = \alpha$, we have $S = \alpha$. Thus the premise consistency derivation is

$$\Psi \mid \Theta_0 [P_0] \vdash [\psi_a] \dot{\sim} Q \dashv \Theta_1 \text{ (2)}.$$

Our goal follows from Lemma 95 on (5) and (2):

$$\Psi, \Psi_a \mid \Theta_0 [P_0] \vdash \psi_a \dot{\sim} Q \dashv \Theta_1 \text{ (3)}.$$

Case

$$\frac{\text{LE-ABSENT} \quad \alpha \notin \text{ftv}(P)}{P \sim P_0 \not\sim^\alpha .}$$

The decorated constraint solving premise is

$$\cdot; \psi_a \rightsquigarrow \cdot \text{ (3)}$$

and the consistency premise is

$$\Psi \mid \Theta_0 [P_0] \vdash S[\llbracket \psi_a \rrbracket / \alpha] \dot{\sim} Q \vdash \Theta_1 \text{ (4)}.$$

Since $\llbracket S \rrbracket = P$ and $\alpha \notin \text{ftv}(P)$, we have $\alpha \notin \text{ftv}(S)$. Hence

$$S[\llbracket \psi_a \rrbracket / \alpha] = S[\psi_a / \alpha] \text{ (2)}.$$

Our goal follows from (4) by (2).

Case

$$\frac{\text{LE-VACUOUS}}{P \sim \mathbb{A} \not\sim^\alpha .}$$

The decorated constraint solving premise is

$$\cdot; \psi_a \rightsquigarrow \cdot \text{ (3)}$$

and the consistency premise is

$$\Psi \mid \Theta_0 [\mathbb{A}] \vdash S[\llbracket \psi_a \rrbracket / \alpha] \dot{\sim} Q \vdash \Theta_1 \text{ (4)}.$$

By a straightforward induction on (4), we get

$$\Psi \mid \Theta_0 [\mathbb{A}] \vdash S[\psi_a / \alpha] \dot{\sim} Q \vdash \Theta_1 \text{ (2)}.$$

Since $\Psi_a = \cdot$, our goal follows from (2).

Case

$$\frac{\text{LE-FLEXR} \quad P \neq \alpha \quad \alpha \in \text{ftv}(P) \quad \hat{\gamma} \text{ fresh}}{P \sim \hat{\beta} \not\sim^\alpha \hat{\gamma}; \text{eq}(\hat{\gamma})}$$

The decorated constraint solving premise is

$$\text{eq}(\hat{\gamma}); \psi_a \rightsquigarrow \Psi_a \text{ (3)}$$

and the consistency premise is

$$\Psi \mid \Theta_0 [\hat{\beta}] \vdash S[\llbracket \psi_a \rrbracket / \alpha] \dot{\sim} Q \vdash \Theta_1 \text{ (4)}.$$

By inversion on (3), we have

$$\psi_a \sim \hat{\gamma} \rightsquigarrow \Psi_a \text{ (5)}.$$

By inversion on ⑤, we have $\Psi_a = \cdot$ ⑥ and ψ_a does not contain dotted ghosts. Since α corresponds to a monotype which cannot determine any tags, by ④, we have

$$\Psi \mid \Theta_0 [\beta] \vdash S[\psi_a/\alpha] \dot{\sim} Q \dashv \Theta_1 \text{ ②}.$$

Our goal follows from ② and ⑥.

Case

LE-ARROW

$$\frac{P_1 \sim P'_1 \not\prec^\alpha \Xi_1; C_1 \text{ ①} \quad P_2 \sim P'_2 \not\prec^\alpha \Xi_2; C_2 \text{ ②}}{P_1 \rightarrow P_2 \sim P'_1 \rightarrow P'_2 \not\prec^\alpha \Xi_1, \Xi_2; C_1, C_2}$$

The decorated constraint solving premise and the consistency premise are

$$C_1, C_2; \psi_a \rightsquigarrow \Psi_a \text{ ⑤} \quad \Psi \mid \Theta_0 [P'_1 \rightarrow P'_2] \vdash S[\llbracket \psi_a \rrbracket / \alpha] \dot{\sim} Q \dashv \Theta_1 \text{ ⑩}.$$

Since $\llbracket S \rrbracket = P_1 \rightarrow P_2$, we have $S = S_1 \rightarrow S_2$ with

$$\llbracket S_1 \rrbracket = P_1 \text{ ③} \quad \llbracket S_2 \rrbracket = P_2 \text{ ④}.$$

By Lemma 78 on ⑤, there are $\Psi_{a,1}$ and $\Psi_{a,2}$ such that

$$C_1; \psi_a \rightsquigarrow \Psi_{a,1} \text{ ⑥} \quad C_2; \psi_a \rightsquigarrow \Psi_{a,2} \text{ ⑦} \quad \Psi_a = \Psi_{a,1}, \Psi_{a,2} \text{ ⑧}.$$

By inversion on ⑩, there exist Q_1, Q_2, Ψ_1, Ψ_2 , and Θ' such that

$$\begin{aligned} \Psi_1 \mid \Theta_0 [P'_1] \vdash S_1[\llbracket \psi_a \rrbracket / \alpha] \dot{\sim} Q_1 \dashv \Theta' \text{ ⑨} \\ \Psi_2 \mid \Theta' [P'_2] \vdash S_2[\llbracket \psi_a \rrbracket / \alpha] \dot{\sim} Q_2 \dashv \Theta_1 \text{ ⑪} \\ Q = Q_1 \rightarrow Q_2 \text{ ⑫} \quad \Psi = \Psi_1, \Psi_2 \text{ ⑬}. \end{aligned}$$

By IH on ①, ⑥, ③, $\alpha \notin \text{ftv}(\llbracket \psi_a \rrbracket)$, and ⑨, we get

$$\Psi_1, \Psi_{a,1} \mid \Theta_0 [P'_1] \vdash S_1[\psi_a/\alpha] \dot{\sim} Q_1 \dashv \Theta' \text{ ⑮}.$$

By IH on ②, ⑦, ④, $\alpha \notin \text{ftv}(\llbracket \psi_a \rrbracket)$, and ⑪, we get

$$\Psi_2, \Psi_{a,2} \mid \Theta' [P'_2] \vdash S_2[\psi_a/\alpha] \dot{\sim} Q_2 \dashv \Theta_1 \text{ ⑯}.$$

Decorated U-ARROW on ⑮ and ⑯ gives

$$\Psi_1, \Psi_{a,1}, \Psi_2, \Psi_{a,2} \mid \Theta_0 [P'_1 \rightarrow P'_2] \vdash S_1[\psi_a/\alpha] \rightarrow S_2[\psi_a/\alpha] \dot{\sim} Q_1 \rightarrow Q_2 \dashv \Theta_1 \text{ ⑰}.$$

The tag context $\Psi_1, \Psi_{a,1}, \Psi_2, \Psi_{a,2}$ is equal to Ψ, Ψ_a by ⑧ and ⑬. Our goal follows from ⑰ and ⑫.

Case

LE-FORALLINH

$$\frac{P \sim P' \not\prec^\alpha \Xi; C \text{ ①}}{\forall \beta. P \sim \forall \beta. P' \not\prec^\alpha \Xi; C}$$

Since $\lfloor S \rfloor = \forall \beta. P$, $S = \forall^\delta \beta. S'$ with $\lfloor S' \rfloor = P$ ②. The decorated constraint solving premise is

$$C; \psi_a \rightsquigarrow \Psi_a \text{ ③.}$$

The consistency premise has the following form for some Q' :

$$\Psi, \delta \mid \Theta_0 [\forall \beta. P'] \vdash S[\lfloor \psi_a \rfloor / \alpha] \dot{\sim} \forall \beta. Q' \dashv \Theta_1 \text{ ⑩.}$$

By inversion on ⑩, we have

$$\Psi \mid \Theta_0, \beta [P'] \vdash S'[\lfloor \psi_a \rfloor / \alpha] \dot{\sim} Q' \dashv \Theta_1, \beta \text{ ④}$$

By IH on ①, ③, ②, $\alpha \notin \text{ftv}(\lfloor \psi_a \rfloor)$, and ④, we get

$$\Psi, \Psi_a \mid \Theta_0, \beta [P'] \vdash S'[\psi_a / \alpha] \dot{\sim} Q' \dashv \Theta_1, \beta \text{ ⑧.}$$

Decorated U-FORALLINH on ⑧ gives

$$\Psi, \Psi_a, \delta \mid \Theta_0 [\forall \beta. P'] \vdash \forall^\delta \beta. S'[\psi_a / \alpha] \dot{\sim} \forall \beta. Q' \dashv \Theta_1 \text{ ⑨.}$$

Our goal follows from ⑨, using associativity and commutativity of set join.

Case

$$\begin{array}{c} \text{LE-FORALLSYN} \\ P \sim \mathbb{A}.P' \not\prec^\alpha \Xi; C \text{ ①} \\ \hline \forall \beta. P \sim \mathbb{A}.P' \not\prec^\alpha \Xi; C \end{array}$$

Since $\lfloor S \rfloor = \forall \beta. P$, $S = \forall^\delta \beta. S'$ with $\lfloor S' \rfloor = P$ ②. The decorated constraint solving premise is

$$C; \psi_a \rightsquigarrow \Psi_a \text{ ③.}$$

The consistency premise has the following form for some Q' :

$$\Psi \mid \Theta_0 [\mathbb{A}.P'] \vdash S[\lfloor \psi_a \rfloor / \alpha] \dot{\sim} \forall \beta. Q' \dashv \Theta_1 \text{ ⑩.}$$

By inversion on ⑩, we have

$$\Psi \mid \Theta_0, \beta [\mathbb{A}.P'] \vdash S'[\lfloor \psi_a \rfloor / \alpha] \dot{\sim} Q' \dashv \Theta_1, \beta \text{ ④}$$

By IH on ①, ③, ②, $\alpha \notin \text{ftv}(\lfloor \psi_a \rfloor)$, and ④, we get

$$\Psi, \Psi_a \mid \Theta_0, \beta [\mathbb{A}.P'] \vdash S'[\psi_a / \alpha] \dot{\sim} Q' \dashv \Theta_1, \beta \text{ ⑦.}$$

Decorated U-FORALLSYN on ⑦ gives

$$\Psi, \Psi_a \mid \Theta_0 [\mathbb{A}.P'] \vdash \forall^\delta \beta. S'[\psi_a / \alpha] \dot{\sim} \forall \beta. Q' \dashv \Theta_1 \text{ ⑧.}$$

Our goal follows from ⑧.

Case

$$\begin{array}{c} \text{LE-UNIVGHOST} \\ P \sim P' \not\prec^\alpha \Xi; C \text{ ①} \\ \hline \mathbb{A}.P \sim \mathbb{A}.P' \not\prec^\alpha \Xi; C \end{array}$$

The decorated constraint solving premise is

$$C; \psi_a \rightsquigarrow \Psi_a \text{ (3)}.$$

The consistency premise has the following form for some Q' :

$$\Psi \mid \Theta_0 [\mathbb{A}.P'] \vdash S[\llbracket \psi_a \rrbracket / \alpha] \dot{\sim} \mathbb{A}.Q' \dashv \Theta_1 \text{ (10)}.$$

Since $\llbracket S \rrbracket = \mathbb{A}.P$, either $S = \mathbb{A}.S'$ or $S = \mathbb{A}.S'$ with $\llbracket S' \rrbracket = P$ (2). Inverting (10) with decorated U-UNIVGHOST or U-UNIVGHOSTG gives

$$\Psi \mid \Theta_0 [P'] \vdash S'[\llbracket \psi_a \rrbracket / \alpha] \dot{\sim} Q' \dashv \Theta_1 \text{ (4)}$$

By IH on (1), (3), (2), $\alpha \notin \text{ftv}(\llbracket \psi_a \rrbracket)$, and (4), we have

$$\Psi, \Psi_a \mid \Theta_0 [P'] \vdash S'[\psi_a / \alpha] \dot{\sim} Q' \dashv \Theta_1 \text{ (6)}.$$

Our goal follows from decorated U-UNIVGHOST or U-UNIVGHOSTG on (6), according to the shape of S .

Case

$$\frac{\text{LE-UNIVGHOST1} \quad \mathbb{A}.P \sim P' \not\prec^\alpha \Xi; C \text{ (1)}}{\mathbb{A}.P \sim \forall \beta. P' \not\prec^\alpha \Xi; C}$$

The decorated constraint solving premise is

$$C; \psi_a \rightsquigarrow \Psi_a \text{ (3)}.$$

The consistency premise has the following form for some Q' :

$$\Psi \mid \Theta_0 [\forall \beta. P'] \vdash S[\llbracket \psi_a \rrbracket / \alpha] \dot{\sim} \forall \beta. Q' \dashv \Theta_1 \text{ (10)}.$$

Since $\llbracket S \rrbracket = \mathbb{A}.P$, inversion on (10) gives that $S = \mathbb{A}.S'$ with $\llbracket S' \rrbracket = P$ (2). Moreover,

$$\Psi \mid \Theta_0, \beta [P'] \vdash \mathbb{A}.S'[\llbracket \psi_a \rrbracket / \alpha] \dot{\sim} Q' \dashv \Theta_1, \beta \text{ (4)}$$

By IH on (1), (3), $\llbracket \mathbb{A}.S' \rrbracket = \mathbb{A}.P$, $\alpha \notin \text{ftv}(\llbracket \psi_a \rrbracket)$, and (4), we have

$$\Psi, \Psi_a \mid \Theta_0, \beta [P'] \vdash \mathbb{A}.S'[\psi_a / \alpha] \dot{\sim} Q' \dashv \Theta_1, \beta \text{ (7)}.$$

Decorated U-UNIVGHOST1 on (7) gives

$$\Psi, \Psi_a \mid \Theta_0 [\forall \beta. P'] \vdash \mathbb{A}.S'[\psi_a / \alpha] \dot{\sim} \forall \beta. Q' \dashv \Theta_1 \text{ (8)}.$$

Our goal follows from (8).

Case

$$\frac{\text{LE-UNIVGHOST2} \quad P' \text{ guarded} \quad P \sim P' \not\prec^\alpha \Xi; C \text{ (1)}}{\mathbb{A}.P \sim P' \not\prec^\alpha \Xi; C}$$

The decorated constraint solving premise and the consistency premise are

$$C; \psi_a \rightsquigarrow \Psi_a \text{ (3)} \quad \Psi \mid \Theta_0 [P'] \vdash S[\llbracket \psi_a \rrbracket / \alpha] \dot{\sim} Q \dashv \Theta_1 \text{ (10)}.$$

Since $\llbracket S \rrbracket = \mathbb{A}.P$ and P' is guarded, inversion on (10) gives two cases.

Subcase P' is a flexible variable. The consistency premise is derived by a flexible variable rule. By (1), we have $\Psi_a = \cdot$. Thus we have

$$\Psi \mid \Theta_0 [P'] \vdash S[\psi_a / \alpha] \dot{\sim} Q \dashv \Theta_1 \text{ (4)}.$$

Our goal follows from (4).

Subcase P' is not a flexible variable. The consistency premise is derived by U-UNIVGHOST2. Then $S = \mathbb{A}.S'$ with $\llbracket S' \rrbracket = P$ (2), and

$$\Psi \mid \Theta_0 [P'] \vdash S'[\llbracket \psi_a \rrbracket / \alpha] \dot{\sim} Q \dashv \Theta_1 \text{ (4)}.$$

By IH on (1), (3), (2), $\alpha \notin \text{ftv}(\llbracket \psi_a \rrbracket)$, and (4), we have

$$\Psi, \Psi_a \mid \Theta_0 [P'] \vdash S'[\psi_a / \alpha] \dot{\sim} Q \dashv \Theta_1 \text{ (5)}.$$

Our goal follows from decorated U-UNIVGHOST2 on (5).

Case

$$\frac{\text{LE-UNIVGHOSTR} \quad P \neq \alpha \quad P \text{ guarded} \quad P \sim P' \not\prec^\alpha \Xi; C \text{ (1)}}{P \sim \mathbb{A}.P' \not\prec^\alpha \Xi; C}$$

The decorated constraint solving premise and the consistency premise are

$$C; \psi_a \rightsquigarrow \Psi_a \text{ (3)} \quad \Psi \mid \Theta_0 [\mathbb{A}.P'] \vdash S[\llbracket \psi_a \rrbracket / \alpha] \dot{\sim} Q \dashv \Theta_1 \text{ (10)}.$$

Since $\llbracket S \rrbracket = P$, $P \neq \alpha$, and P is guarded, $S[\llbracket \psi_a \rrbracket / \alpha]$ is guarded. Inversion on (10) gives

$$\Psi \mid \Theta_0 [P'] \vdash S[\llbracket \psi_a \rrbracket / \alpha] \dot{\sim} Q \dashv \Theta_1 \text{ (4)}.$$

Case analysis on whether P' is a flexible variable.

Subcase P' is a flexible variable. We have $\Psi_a = \cdot$ and

$$\Psi \mid \Theta_0 [P'] \vdash S[\psi_a / \alpha] \dot{\sim} Q \dashv \Theta_1 \text{ (5)}.$$

Our goal follows from decorated U-UNIVGHOSTR on (5).

Subcase P' is not a flexible variable. By IH on (1), (3), $\llbracket S \rrbracket = P$, $\alpha \notin \text{ftv}(\llbracket \psi_a \rrbracket)$, and (4), we have

$$\Psi, \Psi_a \mid \Theta_0 [P'] \vdash S[\psi_a / \alpha] \dot{\sim} Q \dashv \Theta_1 \text{ (5)}.$$

Our goal follows from decorated U-UNIVGHOSTR on (5).

□

Lemma 95 (Consistency constraint decoration for variables). *Suppose*

$$\psi \sim P_0 \rightsquigarrow \Psi_a.$$

For any substitution ρ from rigid type variables to decorated skeletons, if

$$\Psi \mid \Theta_0 \mid P_0 \vdash \rho(\lfloor \psi \rfloor) \dot{\sim} Q \vdash \Theta_1,$$

then

$$\Psi, \Psi_a \mid \Theta_0 \mid P_0 \vdash \rho(\psi) \dot{\sim} Q \vdash \Theta_1.$$

Proof. Induction on the decorated constraint solving derivation $\psi \sim P_0 \rightsquigarrow \Psi_a$, generalising over Θ_0, Ψ, ρ , and the decorated consistency premise. We omit the auxiliary $\Delta_1 \mid \Delta_2$ contexts in decorated consistency judgements throughout this proof.

Case

SE-UNIT

$$\frac{}{1 \sim 1 \rightsquigarrow \cdot}$$

The consistency premise is

$$\Psi \mid \Theta_0 \mid 1 \vdash 1 \dot{\sim} Q \vdash \Theta_1 \text{ ①}.$$

Since $\Psi_a = \cdot$, our goal follows from ①.

Case

SE-VAR

$$\frac{}{\alpha \sim P_0 \rightsquigarrow \cdot}$$

The consistency premise is

$$\Psi \mid \Theta_0 \mid P_0 \vdash \rho(\alpha) \dot{\sim} Q \vdash \Theta_1 \text{ ①}.$$

Since $\Psi_a = \cdot$, our goal follows from ①.

Case

SE-VACUOUS

$$\frac{}{\psi \sim \mathbb{A} \rightsquigarrow \cdot}$$

The consistency premise is

$$\Psi \mid \Theta_0 \mid \mathbb{A} \vdash \rho(\lfloor \psi \rfloor) \dot{\sim} Q \vdash \Theta_1 \text{ ①}.$$

By a straightforward induction on ①, we get

$$\Psi \mid \Theta_0 \mid \mathbb{A} \vdash \rho(\psi) \dot{\sim} Q \vdash \Theta_1 \text{ ②}.$$

Since $\Psi_a = \cdot$, our goal follows from ②.

Case

$$\begin{array}{c} \text{SE-ARROW} \\ \frac{\psi_1 \sim P_1 \rightsquigarrow \Psi_{a,1} \text{ ①} \quad \psi_2 \sim P_2 \rightsquigarrow \Psi_{a,2} \text{ ②}}{\psi_1 \rightarrow \psi_2 \sim P_1 \rightarrow P_2 \rightsquigarrow \Psi_{a,1}, \Psi_{a,2}} \end{array}$$

The consistency premise has the following form for some Q_1 and Q_2 :

$$\Psi \mid \Theta_0 [P_1 \rightarrow P_2] \vdash \rho(\lfloor \psi_1 \rfloor) \rightarrow \rho(\lfloor \psi_2 \rfloor) \dot{\sim} Q_1 \rightarrow Q_2 \vdash \Theta_1 \text{ ③}.$$

By inversion on ③, there exist Ψ_1, Ψ_2 , and Θ' such that

$$\Psi_1 \mid \Theta_0 [P_1] \vdash \rho(\lfloor \psi_1 \rfloor) \dot{\sim} Q_1 \vdash \Theta' \text{ ④} \quad \Psi_2 \mid \Theta' [P_2] \vdash \rho(\lfloor \psi_2 \rfloor) \dot{\sim} Q_2 \vdash \Theta_1 \text{ ⑤}$$

and $\Psi = \Psi_1, \Psi_2$ ⑥. By IH on ①, ρ , and ④, we have

$$\Psi_1, \Psi_{a,1} \mid \Theta_0 [P_1] \vdash \rho(\psi_1) \dot{\sim} Q_1 \vdash \Theta' \text{ ⑦}.$$

By IH on ②, ρ , and ⑤, we have

$$\Psi_2, \Psi_{a,2} \mid \Theta' [P_2] \vdash \rho(\psi_2) \dot{\sim} Q_2 \vdash \Theta_1 \text{ ⑧}.$$

Decorated U-ARROW on ⑦ and ⑧ gives

$$\Psi_1, \Psi_{a,1}, \Psi_2, \Psi_{a,2} \mid \Theta_0 [P_1 \rightarrow P_2] \vdash \rho(\psi_1) \rightarrow \rho(\psi_2) \dot{\sim} Q_1 \rightarrow Q_2 \vdash \Theta_1 \text{ ⑨}.$$

Our goal follows from ⑨ and ⑥.

Case

$$\begin{array}{c} \text{SE-FORALLINH} \\ \frac{\psi \sim P \rightsquigarrow \Psi_a \text{ ①}}{\forall^\epsilon \beta. \psi \sim \forall \beta. P \rightsquigarrow \Psi_a, \epsilon} \end{array}$$

The consistency premise has the following form for some Q' :

$$\Psi \mid \Theta_0 [\forall \beta. P] \vdash \forall \beta. \rho(\lfloor \psi \rfloor) \dot{\sim} \forall \beta. Q' \vdash \Theta_1 \text{ ②}.$$

By inversion on ②, we have

$$\Psi \mid \Theta_0, \beta [P] \vdash \rho(\lfloor \psi \rfloor) \dot{\sim} Q' \vdash \Theta_1, \beta \text{ ③}.$$

By IH on ①, ρ , and ③, we get

$$\Psi, \Psi_a \mid \Theta_0, \beta [P] \vdash \rho(\psi) \dot{\sim} Q' \vdash \Theta_1, \beta \text{ ④}.$$

Decorated U-FORALLINH on ④ gives

$$\Psi, \Psi_a, \epsilon \mid \Theta_0 [\forall \beta. P] \vdash \forall^\epsilon \beta. \rho(\psi) \dot{\sim} \forall \beta. Q' \vdash \Theta_1 \text{ ⑤}.$$

Our goal follows from ⑤ and $\rho(\forall^\epsilon \beta. \psi) = \forall^\epsilon \beta. \rho(\psi)$.

Case

$$\frac{\text{SE-UNIVGHOSTG} \quad \psi \sim P \rightsquigarrow \Psi_a \text{ ①}}{\mathbb{A}.\psi \sim \mathbb{A}.P \rightsquigarrow \Psi_a}$$

The consistency premise has the following form for some Q' :

$$\Psi \mid \Theta_0 [\mathbb{A}.P] \vdash \mathbb{A}.\rho(\lfloor \psi \rfloor) \dot{\sim} \mathbb{A}.Q' \vdash \Theta_1 \text{ ②}.$$

By inversion on ② with decorated U-UNIVGHOST, we have

$$\Psi \mid \Theta_0 [P] \vdash \rho(\lfloor \psi \rfloor) \dot{\sim} Q' \vdash \Theta_1 \text{ ③}.$$

By IH on ①, ρ , and ③, we get

$$\Psi, \Psi_a \mid \Theta_0 [P] \vdash \rho(\psi) \dot{\sim} Q' \vdash \Theta_1 \text{ ④}.$$

Our goal follows from decorated U-UNIVGHOSTG on ④.

Case

$$\frac{\text{SE-FORALLSYN} \quad \psi \sim \mathbb{A}.P \rightsquigarrow \Psi_a \text{ ①}}{\forall^\epsilon \beta. \psi \sim \mathbb{A}.P \rightsquigarrow \Psi_a}$$

The consistency premise has the following form for some Q' :

$$\Psi \mid \Theta_0 [\mathbb{A}.P] \vdash \forall \beta. \rho(\lfloor \psi \rfloor) \dot{\sim} \forall \beta. Q' \vdash \Theta_1 \text{ ②}.$$

By inversion on ②, we have

$$\Psi \mid \Theta_0, \beta [\mathbb{A}.P] \vdash \rho(\lfloor \psi \rfloor) \dot{\sim} Q' \vdash \Theta_1, \beta \text{ ③}.$$

By IH on ①, ρ , and ③, we get

$$\Psi, \Psi_a \mid \Theta_0, \beta [\mathbb{A}.P] \vdash \rho(\psi) \dot{\sim} Q' \vdash \Theta_1, \beta \text{ ④}.$$

Decorated U-FORALLSYN on ④ gives

$$\Psi, \Psi_a \mid \Theta_0 [\mathbb{A}.P] \vdash \forall^\epsilon \beta. \rho(\psi) \dot{\sim} \forall \beta. Q' \vdash \Theta_1 \text{ ⑤}.$$

Our goal follows from ⑤ and $\rho(\forall^\epsilon \beta. \psi) = \forall^\epsilon \beta. \rho(\psi)$.

Case

$$\frac{\text{SE-UNIVGHOSTR} \quad \lfloor \psi \rfloor \text{ guarded} \quad \psi \sim P \rightsquigarrow \Psi_a \text{ ①}}{\psi \sim \mathbb{A}.P \rightsquigarrow \Psi_a}$$

The consistency premise is

$$\Psi \mid \Theta_0 [\mathbb{A}.P] \vdash \rho(\lfloor \psi \rfloor) \dot{\sim} Q \vdash \Theta_1 \text{ ②}.$$

If $\rho(\lfloor \psi \rfloor)$ is unguarded or flexible, then ψ is a rigid or flexible variable. Thus $\Psi_a = \cdot$ and $\rho(\psi) = \rho(\lfloor \psi \rfloor)$, so our goal follows from ②. Otherwise,

$$\Theta_0 \vdash \rho(\lfloor \psi \rfloor) \text{ guarded } \textcircled{3} \quad \rho(\lfloor \psi \rfloor) \text{ non-flex } \textcircled{4}.$$

Inversion on ② with decorated U-UNIVGHOSTR gives

$$\Psi \mid \Theta_0 [P] \vdash \rho(\lfloor \psi \rfloor) \sim Q \vdash \Theta_1 \textcircled{5}.$$

By IH on ①, ρ , and ⑤, we have

$$\Psi, \Psi_a \mid \Theta_0 [P] \vdash \rho(\psi) \sim Q \vdash \Theta_1 \textcircled{6}.$$

In this case, $\rho(\psi)$ has the same top-level shape as $\rho(\lfloor \psi \rfloor)$. Thus by ③ and ④, it is guarded and non-flex. Our goal follows from decorated U-UNIVGHOSTR on ⑥.

Case

SE-FLEXL

$$\frac{}{\hat{\alpha} \sim P_0 \rightsquigarrow \cdot}$$

The consistency premise is

$$\Psi \mid \Theta_0 [P_0] \vdash \hat{\alpha} \sim Q \vdash \Theta_1 \textcircled{1}.$$

Since $\Psi_a = \cdot$, our goal follows from ①.

Case

SE-FLEXR

$$\frac{\hat{\alpha} \notin \psi \quad \hat{\alpha} \notin \psi}{\psi \sim \hat{\alpha} \rightsquigarrow \cdot}$$

The consistency premise is

$$\Psi \mid \Theta_0 [\hat{\alpha}] \vdash \rho(\lfloor \psi \rfloor) \sim Q \vdash \Theta_1 \textcircled{1}.$$

Since ψ contains no dotted ghosts, erasure only removes quantifier tags. Since $\hat{\alpha}$ is a monotype, ① gives

$$\Psi \mid \Theta_0 [\hat{\alpha}] \vdash \rho(\psi) \sim Q \vdash \Theta_1 \textcircled{2}.$$

Since $\Psi_a = \cdot$, our goal follows from ②.

□

Lemma 96 (Completeness of consistency constraint collection and solving). *Suppose*

$$\Theta_0 \vdash P_0 \quad \Theta_0 \vdash P_i \quad [S] = P \quad \theta :: \Theta_0 \sqsubseteq \Theta \quad \alpha \notin \Theta.$$

If

$$\Theta \vdash^n \psi_a \# \Psi' \quad \Psi \text{ is disjoint from } \Psi' \quad \Psi_i, \Psi_a \subseteq \Psi'$$

and

$$\Theta \mid \Delta \vdash \theta P_i \leq_{\Psi_i} \psi_a \quad \Psi, \Psi_a \mid \Theta \vdash (\theta S)[\psi_a/\alpha] \sim Q \quad \Theta \vdash \theta P_0 \blacktriangleright Q,$$

then there exist C, Ξ_c, Ξ_s, P_a , and θ_a such that

$$P \sim P_0 \not\leq^\alpha \Xi_c; C \quad C; P_i \rightsquigarrow \Xi_s; P_a \quad \Theta_0, \Xi_c, \Xi_s \vdash P_a$$

and

$$\theta_a :: \Theta_0, \Xi_c, \Xi_s \sqsubseteq \Theta \quad \theta \equiv \theta_a :: \Theta_0 \sqsubseteq \Theta \quad \Theta \mid \Delta \vdash \theta_a P_a \leq_{\Psi_i, \Psi_a} \psi_a.$$

Proof. By induction on the derivation of

$$\Psi, \Psi_a \mid \Theta \vdash (\theta S)[\psi_a/\alpha] \sim Q,$$

generalising over $\Theta_0, P_0, P_i, P, S, \theta, \psi_a, \Psi', \Psi_i, \Psi_a, \Psi$, and Δ . We first discuss the three special cases.

If $P_0 = \mathbb{A}$, then by inversion on the refinement premise, Q is a synthesised skeleton. Thus $\Psi_a = \cdot$. By LE-VACUOUS and constraint solving, we have

$$P \sim \mathbb{A} \not\leq^\alpha \cdot \quad \cdot; P_i \rightsquigarrow P_i.$$

The goal follows by choosing $C = \cdot, \Xi_c = \cdot, P_a = P_i$, and $\theta_a = \theta$.

If $P = \alpha$, then $S = \alpha$. By well-formedness of ψ_a and disjointness of Ψ from Ψ' , we have $\Psi = \cdot$. By Lemma 97 with the identity substitution, which introduces the empty tag context, there are Ξ_s, P_a , and θ_a such that

$$\begin{aligned} P_i \sim P_0 \rightsquigarrow \Xi_s; P_a & \text{ ①} & \Theta_0, \Xi_s \vdash P_a & \text{ ②} \\ \theta_a :: \Theta_0, \Xi_s \sqsubseteq \Theta & \text{ ③} & \theta \equiv \theta_a :: \Theta_0 \sqsubseteq \Theta & \text{ ④} & \Theta \mid \Delta \vdash \theta_a P_a \leq_{\Psi_i, \Psi_a} \psi_a & \text{ ⑤}. \end{aligned}$$

By LE-VAR and constraint solving, we have

$$\alpha \sim P_0 \not\leq^\alpha \text{eq}(P_0) \quad \text{eq}(P_0); P_i \rightsquigarrow \Xi_s; P_a.$$

Our goal follows by choosing $C = \text{eq}(P_0)$ and $\Xi_c = \cdot$.

If $\alpha \notin \text{ftv}(P)$, then the source does not contain ψ_a , so exactness of solved tag contexts gives $\Psi_a = \cdot$. By LE-ABSENT and constraint solving, we have:

$$P \sim P_0 \not\leq^\alpha \cdot \quad \cdot; P_i \rightsquigarrow P_i$$

The goal follows by choosing $C = \cdot, \Xi_c = \cdot, P_a = P_i$, and $\theta_a = \theta$.

In the following induction cases, assume $P_0 \neq \mathbb{A}, P \neq \alpha$, and $\alpha \in \text{ftv}(P)$.

Case C-UNIT, C-VAR, C-GHOST, and C-GHOSTG. Impossible by our assumptions.

Case

$$\begin{array}{c} \text{C-FORALLINH} \\ \frac{\Psi_0, \Psi_a \mid \Theta, \beta \vdash (\theta S')[\psi_a/\alpha] \sim Q \text{ ①}}{\Psi_0, \Psi_a, \delta \mid \Theta \vdash \forall^\delta \beta. ((\theta S')[\psi_a/\alpha]) \sim \forall \beta. Q} \end{array}$$

In this case $S = \forall^\delta \beta. S'$ and $P = \forall \beta. P'$ with

$$\llbracket S' \rrbracket = P' \text{ ②}.$$

By inversion on the refinement premise, $P_0 = \forall \beta. P'_0$ and

$$\Theta, \beta \vdash \theta P'_0 \blacktriangleright Q \text{ ③}.$$

The conclusion gives $\Psi = \Psi_0, \delta$. By IH on ① with ②, ③, and context weakening on the other premises, there are C, Ξ_c, Ξ_s, P_a , and θ_a such that

$$\begin{array}{lll} P' \sim P'_0 \not\prec^\alpha \Xi_c; C \text{ ④} & C; P_i \rightsquigarrow \Xi_s; P_a \text{ ⑤} & \Theta_0, \beta, \Xi_c, \Xi_s \vdash P_a \text{ ⑥} \\ \theta_a :: \Theta_0, \beta, \Xi_c, \Xi_s \sqsubseteq \Theta, \beta \text{ ⑦} & \theta \equiv \theta_a :: \Theta_0, \beta \sqsubseteq \Theta, \beta \text{ ⑧} & \Theta, \beta \mid \Delta \vdash \theta_a P_a \leq_{\Psi_i, \Psi_a} \psi_a \text{ ⑨}. \end{array}$$

By LE-FORALLINH on ④, we have

$$\forall \beta. P' \sim \forall \beta. P'_0 \not\prec^\alpha \Xi_c; C \text{ ⑩}.$$

We know $\beta \notin \text{ftv}(P_a)$ because P_i and C cannot contain β . ⑥, ⑦, ⑧, and ⑨ hold without β . The goal follows from these judgements, ⑤, and ⑩.

Case

$$\begin{array}{c} \text{C-FORALLSYN} \\ \frac{\Psi, \Psi_a \mid \Theta, \beta \vdash (\theta S')[\psi_a/\alpha] \sim Q \text{ ①}}{\Psi, \Psi_a \mid \Theta \vdash \forall^\delta \beta. ((\theta S')[\psi_a/\alpha]) \sim \forall \beta. Q} \end{array}$$

In this case $S = \forall^\delta \beta. S'$ and $P = \forall \beta. P'$ with

$$\llbracket S' \rrbracket = P' \text{ ②}.$$

Since $P_0 \neq \mathbb{A}$, inversion on the refinement premise gives $P_0 = \mathbb{A}.P'_0$ and

$$\Theta, \beta \vdash \theta(\mathbb{A}.P'_0) \blacktriangleright Q \text{ ③}.$$

By IH on ① with ②, ③, and context weakening on the other premises, there are C, Ξ_c, Ξ_s, P_a , and θ_a such that

$$\begin{array}{lll} P' \sim \mathbb{A}.P'_0 \not\prec^\alpha \Xi_c; C \text{ ④} & C; P_i \rightsquigarrow \Xi_s; P_a \text{ ⑤} & \Theta_0, \beta, \Xi_c, \Xi_s \vdash P_a \text{ ⑥} \\ \theta_a :: \Theta_0, \beta, \Xi_c, \Xi_s \sqsubseteq \Theta, \beta \text{ ⑦} & \theta \equiv \theta_a :: \Theta_0, \beta \sqsubseteq \Theta, \beta \text{ ⑧} & \\ \Theta, \beta \mid \Delta \vdash \theta_a P_a \leq_{\Psi_i, \Psi_a} \psi_a \text{ ⑨}. & & \end{array}$$

By LE-FORALLSYN on ④, we have

$$\forall \beta. P' \sim \mathbb{A}.P'_0 \not\prec^\alpha \Xi_c; C \text{ ⑩}.$$

As in the case of C-FORALL_{INH}, our goal follows from the restricted judgements, ⑤–⑩, removing β as necessary.

Case

C-ARROW

$$\frac{\Psi_1, \Psi_{a,1} \mid \Theta \vdash (\theta S_1)[\psi_a/\alpha] \sim Q_1 \text{ ①} \quad \Psi_2, \Psi_{a,2} \mid \Theta \vdash (\theta S_2)[\psi_a/\alpha] \sim Q_2 \text{ ②}}{\Psi_1, \Psi_{a,1}, \Psi_2, \Psi_{a,2} \mid \Theta \vdash (\theta S_1)[\psi_a/\alpha] \rightarrow (\theta S_2)[\psi_a/\alpha] \sim Q_1 \rightarrow Q_2}$$

Similar to the Arrow case of Lemma 89. Our goal follows from applying IH twice on the function argument and result, then using LE-ARROW and sequential constraint solving.

Case C-UNIVGHOSTG and C-UNIVGHOST. Inversion on the refinement premise and IH give the premise of LE-UNIVGHOST, from which our goal follows.

Case C-UNIVGHOST1. As in C-FORALL_{INH}, IH under the extended context gives the premise of LE-UNIVGHOST1, from which our goal follows.

Case C-UNIVGHOST2. In this case $P = \mathbb{A}.P'$. By well-formedness of P and inversion on the consistency premise, Q is guarded. We proceed by case analysis on P_0 . If $P_0 = \mathbb{A}.P'_0$, then inversion on the refinement premise and IH give the premise of LE-UNIVGHOST. If P_0 is guarded, then inversion on the refinement premise and IH give the premise of LE-UNIVGHOST2. Our goal follows from the constraint solving rule in each case.

□

Lemma 97 (Completeness of consistency constraint collection and solving for variables).
Suppose

$$\Theta_0 \vdash P_0 \quad \Theta_0 \vdash P_i \quad \theta :: \Theta_0 \sqsubseteq \Theta.$$

Let ρ be a capture-avoiding substitution from rigid type variables to decorated skeletons, and Ψ be the solved tags introduced by ρ . If ψ_a is a well-formed instantiation and the premise solves Ψ_a :

$$\Theta \vdash^n \psi_a \# \Psi' \quad \Psi_i, \Psi_a \subseteq \Psi' \quad \Psi' \text{ fresh}$$

and

$$\Theta \mid \Delta \vdash \theta P_i \leq_{\Psi_i} \psi_a \quad \Psi, \Psi_a \mid \Theta \vdash \rho(\psi_a) \sim Q \quad \Theta \vdash \theta P_0 \blacktriangleright Q.$$

Moreover, for every rigid variable in ψ_a substituted by ρ , P_i has the same rigid variable in the same position. Then there exist Ξ_s , P_a , and θ_a such that

$$P_i \sim P_0 \rightsquigarrow \Xi_s; P_a \quad \Theta_0, \Xi_s \vdash P_a$$

and

$$\theta_a :: \Theta_0, \Xi_s \sqsubseteq \Theta \quad \theta \equiv \theta_a :: \Theta_0 \sqsubseteq \Theta \quad \Theta \mid \Delta \vdash \theta_a P_a \leq_{\Psi_i, \Psi_a} \psi_a.$$

Proof. By induction on the derivation of $\Psi, \Psi_a \mid \Theta \vdash \rho(\psi_a) \sim Q$, generalising over $\Theta_0, P_0, P_i, \theta, \psi_a, \Psi', \Psi_i, \Psi_a, \Psi, \rho$, and Δ . We first discuss three special cases.

If $P_0 = \mathbb{L}$, then by inversion on the refinement premise, Q is a synthesised skeleton. Hence $\Psi_a = \cdot$. By **SOLE-VACUOUSR**, we have

$$P_i \sim \mathbb{L} \rightsquigarrow P_i.$$

The goal follows by choosing $P_a = P_i$ and $\theta_a = \theta$.

If $P_0 = \hat{\gamma}$, we proceed by case analysis on P_i . If $P_i = \hat{\beta}$, rule **SOLE-FLEXL** gives the goal with the empty suffix, $P_a = P_i$, and $\theta_a = \theta$. If P_i is non-flexible, let $\text{guess}(P_i) = (\Xi_s; P_a)$. Rule **SOLE-FLEXR** gives $P_i \sim \hat{\gamma} \rightsquigarrow \Xi_s; P_a$. Our goals follow from a straightforward induction on $\text{guess}(P_i)$, where θ_a is constructed by extending θ over Ξ_s .

If ψ_a is a rigid variable substituted by ρ , then $P_i = \psi_a$. The tag contexts Ψ_i and Ψ_a are empty, and the goal follows from **SOLE-VAR**, choosing $P_a = P_i$ and $\theta_a = \theta$.

In the remaining cases, ρ preserves the top-level constructor of ψ_a , $P_0 \neq \mathbb{L}$, and P_0 is not a flexible variable.

Case C-UNIT, C-VAR, and C-GHOSTG. By inversion on the consistency derivation, $\Psi_a = \cdot$.

We proceed by case analysis on the shapes of P_i and P_0 exposed by the precision and refinement premises. When they are unit or rigid variables, we use **SOLE-UNIT** or **SOLE-VAR**. When they are ghosts or have universal ghosts, we use **SOLE-GHOSTL**, **SOLE-UNIVGHOST**, **SOLE-GHOSTGUARDED**, or **SOLE-GUARDEDGHOST**. When P_i is a flexible variable, we use **SOLE-FLEXL**.

Case C-GHOST. Impossible since ψ_a cannot contain an ordinary ghost.

Case

$$\begin{array}{c} \text{C-FORALLINH} \\ \hline \Psi, \Psi'_a \mid \Theta, \beta \vdash \rho(\psi') \sim Q \text{ ①} \\ \hline \Psi, \Psi'_a, \epsilon \mid \Theta \vdash \rho(\forall^\epsilon \beta. \psi') \sim \forall \beta. Q \end{array}$$

In this case $\psi_a = \forall^\epsilon \beta. \psi'$ and $\Psi_a = \Psi'_a, \epsilon$. We have $P_0 = \forall \beta. P'_0$. Inversion on the refinement premise gives

$$\Theta, \beta \vdash \theta P'_0 \blacktriangleright Q \text{ ②}.$$

We proceed by case analysis on P_i .

Subcase $P_i = \forall \beta. P'_i$. Inversion on precision gives

$$\Theta, \beta \mid \Delta, \beta \vdash \theta P'_i \leq_{\Psi'_i} \psi' \text{ ③} \quad \Psi_i = \Psi'_i, \epsilon.$$

IH on ①, ②, and ③ gives Ξ_s, P_b , and θ_a with

$$P'_i \sim P'_0 \rightsquigarrow \Xi_s; P_b \text{ ④}.$$

Our goal follows from **SOLE-FORALL** on ④ and choosing $P_a = \forall\beta.P_b$.

Subcase $P_i = \mathbb{A}$. IH on ① and ② with input $\mathbb{A}$ gives θ_a with

$$\mathbb{A} \sim P'_0 \rightsquigarrow P'_0 \text{ ④}.$$

Our goal follows from **SOLE-GHOSTL** and choosing $P_a = \forall\beta.P'_0$.

Subcase $P_i = \mathbb{A}.P'_i$. Inversion on precision gives

$$\Theta, \beta \mid \Delta \vdash \theta(\mathbb{A}.P'_i) \leq_{\Psi_i} \psi' \text{ ③}.$$

IH on ①, ②, and ③ gives Ξ_s, P_b , and θ_a with

$$\mathbb{A}.P'_i \sim P'_0 \rightsquigarrow \Xi_s; P_b \text{ ④}.$$

Our goal follows from **SOLE-GHOSTFORALL** on ④ and choosing $P_a = \forall\beta.P_b$.

Subcase The remaining non-flexible shapes of P_i are impossible by inversion on the precision premise. The flexible variable case follows from **SOLE-FLEXL** and the precision premise.

Case

$$\begin{array}{c} \text{C-FORALLSYN} \\ \hline \Psi, \Psi_a \mid \Theta, \beta \vdash \rho(\psi') \sim Q \text{ ①} \\ \hline \Psi, \Psi_a \mid \Theta \vdash \rho(\forall^\epsilon \beta. \psi') \sim \forall\beta.Q \end{array}$$

In this case $\psi_a = \forall^\epsilon \beta. \psi'$. We have $P_0 = \mathbb{A}.P'_0$. Inversion on the refinement premise gives

$$\Theta, \beta \vdash \theta(\mathbb{A}.P'_0) \triangleright Q \text{ ②}.$$

We proceed by case analysis on P_i .

Subcase $P_i = \forall\beta.P'_i$. Inversion on the precision premise gives

$$\Theta, \beta \mid \Delta, \beta \vdash \theta P'_i \leq_{\Psi'_i} \psi' \text{ ③} \quad \Psi_i = \Psi'_i, \epsilon.$$

IH on ①, ②, and ③ gives Ξ_s, P_b , and θ_a with

$$P'_i \sim \mathbb{A}.P'_0 \rightsquigarrow \Xi_s; P_b \text{ ④}.$$

Our goal follows from **SOLE-FORALLGHOST** on ④ and choosing $P_a = \forall\beta.P_b$.

Subcase $P_i = \mathbb{A}$. IH on ① and ② with input $\mathbb{A}$ gives θ_a with

$$\mathbb{A} \sim \mathbb{A}.P'_0 \rightsquigarrow \mathbb{A}.P'_0 \text{ ④}.$$

Our goal follows from **SOLE-GHOSTL** and choosing $P_a = \mathbb{A}.P'_0$.

Subcase $P_i = \mathbb{A}.P'_i$. Inversion on precision gives

$$\Theta, \beta \mid \Delta \vdash \theta(\mathbb{A}.P'_i) \leq_{\Psi_i} \psi' \text{ (3)}.$$

IH on (1), (2), and (3) gives Ξ_s, P_b , and θ_a with

$$P'_i \sim P'_0 \rightsquigarrow \Xi_s; P_b \text{ (4)}.$$

Our goal follows from SOLE-UNIVGHOST on (4) and choosing $P_a = \mathbb{A}.P_b$.

Subcase The remaining non-flexible shapes of P_i are impossible by inversion on the precision premise. The flexible case follows from SOLE-FLEXL.

Case

C-ARROW

$$\frac{\Psi_1, \Psi_{a,1} \mid \Theta \vdash \rho(\psi_1) \sim Q_1 \text{ (1)} \quad \Psi_2, \Psi_{a,2} \mid \Theta \vdash \rho(\psi_2) \sim Q_2 \text{ (2)}}{\Psi_1, \Psi_{a,1}, \Psi_2, \Psi_{a,2} \mid \Theta \vdash \rho(\psi_1) \rightarrow \rho(\psi_2) \sim Q_1 \rightarrow Q_2}$$

In this case $\psi_a = \psi_1 \rightarrow \psi_2$. We have

$$\Psi = \Psi_1, \Psi_2 \quad \Psi_a = \Psi_{a,1}, \Psi_{a,2}.$$

We assume $P_0 = P_{0,1} \rightarrow P_{0,2}$; if it has a top-level universal ghost, we can always deal with it in the case analysis below with proper constraint solving rules. Inversion on the refinement premise gives

$$\Theta \vdash \theta P_{0,1} \blacktriangleright Q_1 \text{ (3)} \quad \Theta \vdash \theta P_{0,2} \blacktriangleright Q_2 \text{ (4)}.$$

We proceed by case analysis on P_i .

Subcase $P_i = P_{i,1} \rightarrow P_{i,2}$. By inversion on the precision premise, there are $\Psi_{i,1}$ and $\Psi_{i,2}$ such that

$$\Psi_i = \Psi_{i,1}, \Psi_{i,2} \quad \Theta \mid \Delta \vdash \theta P_{i,1} \leq_{\Psi_{i,1}} \psi_1 \text{ (5)} \quad \Theta \mid \Delta \vdash \theta P_{i,2} \leq_{\Psi_{i,2}} \psi_2 \text{ (6)}.$$

IH on (1), (3), and (5) gives Ξ_1, P_1 , and θ_1 with

$$P_{i,1} \sim P_{0,1} \rightsquigarrow \Xi_1; P_1 \text{ (7)}.$$

After rewriting (4) and (6) with θ_1 , IH on (2) with input context Θ_0, Ξ_1 gives Ξ_2, P_2 , and θ_a with

$$P_{i,2} \sim P_{0,2} \rightsquigarrow \Xi_2; P_2 \text{ (8)}.$$

Our goal follows from SOLE-ARROW on (7) and (8) and choosing $\Xi_s = \Xi_1, \Xi_2$ and $P_a = P_1 \rightarrow P_2$.

Subcase $P_i = \mathbb{A}$. Our goal follows from IH on (1) and (3), IH on (2) and (4), SOLE-GHOSTL and choosing $P_a = P_{0,1} \rightarrow P_{0,2}$.

Subcase $P_i = \mathbb{A}.P'_i$. By well-formedness, P'_i is guarded. Inversion on precision gives

$$\Theta \mid \Delta \vdash \theta P'_i \leq_{\Psi_i} \psi_1 \rightarrow \psi_2 \text{ (5)}.$$

We reuse the previous subcases on (5) to obtain P_a , and then use **SOLE-GHOSTGUARDED**.

Subcase The remaining non-flexible shapes of P_i are impossible by inversion on the precision premise. The flexible case follows from **SOLE-FLEXL**.

Case

$$\frac{\text{C-UNIVGHOSTG} \quad \Psi, \Psi_a \mid \Theta \vdash \rho(\psi') \sim Q \text{ (1)}}{\Psi, \Psi_a \mid \Theta \vdash \mathbb{A}.\rho(\psi') \sim \mathbb{A}.Q}$$

In this case $\psi_a = \mathbb{A}.\psi'$. We have $P_0 = \mathbb{A}.P'_0$. Inversion on the refinement premise gives

$$\Theta \vdash \theta P'_0 \blacktriangleright Q \text{ (2)}.$$

We proceed by case analysis on P_i .

Subcase $P_i = \mathbb{A}$. IH on (1) and (2) with input $\mathbb{A}$ gives θ_a with

$$\mathbb{A} \sim P'_0 \rightsquigarrow P'_0 \text{ (3)}.$$

Our goal follows from **SOLE-GHOSTL** and choosing $P_a = \mathbb{A}.P'_0$.

Subcase $P_i = \mathbb{A}.P'_i$. Inversion on precision gives

$$\Theta \mid \Delta \vdash \theta P'_i \leq_{\Psi_i} \psi' \text{ (3)}.$$

IH on (1), (2), and (3) gives Ξ_s, P_b , and θ_a with

$$P'_i \sim P'_0 \rightsquigarrow \Xi_s; P_b \text{ (4)}.$$

Our goal follows from **SOLE-UNIVGHOST** on (4) and choosing $P_a = \mathbb{A}.P_b$.

Subcase The remaining non-flexible shapes of P_i are impossible by inversion on the precision premise. The flexible case follows from **SOLE-FLEXL**.

Case **C-UNIVGHOST**, **C-UNIVGHOST1**, and **C-UNIVGHOST2**. Impossible, since neither ψ_a nor $\rho(\psi_a)$ has an ordinary universal ghost at the top level.

□

D.5.2 Soundness of consistency

We prove Lemma 25.

Lemma 25 (Soundness of consistency).

1. Given a problem of consistency $(\Theta_0; P_0; P)$, if $\Theta_0 [P_0] \vdash P \sim P_1 \dashv \Theta_1$, then $(\Theta_0 \sqsubseteq \Theta_1; P_1)$ is a solution to it.
2. Given a problem of variable unification $(\Theta_0; \hat{\alpha}; \hat{\beta})$, if $\Theta_0 \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta_1$, then $\Theta_0 \sqsubseteq \Theta_1$ is a solution to it.
3. Given a problem of assignment $(\Theta_0; \Xi; \hat{\alpha}; P)$, if $\Theta_0 \mid \Xi \vdash \hat{\alpha} := P \dashv \Theta_1$, then $\Theta_0, \Xi \sqsubseteq \Theta_1$ is a solution to it.

Proof. We first augment the definition of a consistency problem to $(\Delta_1; \Delta_2; \Theta_0; P_0; P)$ where Δ_1 and Δ_2 are contexts of rigid type variables present in Θ_0 that no flexible variables depend on. We augment the definition of a solution $(\theta :: \Theta_0 \sqsubseteq \Theta; Q)$ to require $\Theta \vdash \theta P \sim Q$ and $\Theta \mid \Delta_1 \vdash \theta P_0 \blacktriangleright Q$ and $\Theta \mid \Delta_2 \vdash \theta P \blacktriangleright Q$ and that no flexible variables in Θ depend on variables in Δ_1 and Δ_2 . For part 1 of the lemma, it is enough to prove that given a consistency problem $(\Delta_1; \Delta_2; \Theta_0; P_0; P)$, if $\Theta_0 [P_0]_{\Delta_1 \mid \Delta_2} \vdash P \sim P_1 \dashv \Theta_1$, then $\Theta_0 \sqsubseteq \Theta_1$ and $\Theta_1 \vdash P \sim P_1$ and $\Theta_1 \mid \Delta_1 \vdash P_0 \blacktriangleright P_1$ and $\Theta_1 \mid \Delta_2 \vdash P \blacktriangleright P_1$.

By mutual induction on the algorithmic consistency derivation together with the two auxiliary judgements of variable unification and assignment. We first prove the cases for the two auxiliary judgements.

2. Unification of flexible variables. We prove that $\Theta_0 \sqsubseteq \Theta_1$ is a solution by induction on the unification derivation

$$\Theta_0 \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta_1.$$

Case

$$\frac{\text{U-FLEX-FLEX-ID} \quad \hat{\alpha} \in \text{dom}(\Theta_0)}{\Theta_0 \vdash \hat{\alpha} \simeq \hat{\alpha} \dashv \Theta_0}$$

Trivial.

Case

$$\frac{\text{U-FLEX-FLEX-L} \quad \hat{\alpha} \neq \hat{\beta}}{\Theta_0, \hat{\alpha} \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta_0, \hat{\alpha} := \hat{\beta}}$$

By definition of information increase, we have

$$\Theta_0, \hat{\alpha} \sqsubseteq \Theta_0, \hat{\alpha} := \hat{\beta} \text{ ②}.$$

By declarative consistency rule C-FLEXL, we have

$$\Theta_0, \hat{\alpha} := \hat{\beta} \vdash \hat{\alpha} \sim \hat{\beta} \text{ ③},$$

Our goal follows from ② and ③.

Case

$$\frac{\text{U-FLEX-FLEX-R} \quad \hat{\alpha} \neq \hat{\beta}}{\Theta_0, \hat{\beta} \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta_0, \hat{\beta} := \hat{\alpha}}$$

Similar to the case of U-FLEX-FLEX-L.

Case

$$\frac{\text{U-FLEX-FLEX-ASSIGN} \quad \Theta_0 \vdash \hat{\alpha}[\tau/\hat{\gamma}] \sim \hat{\beta}[\tau/\hat{\gamma}] \dashv \Theta_1 \text{ ②}}{\Theta_0, \hat{\gamma} := \tau \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta_1, \hat{\gamma} := \tau}$$

By IH on ②,

$$\Theta_0 \sqsubseteq \Theta_1 \text{ ③} \quad \Theta_1 \vdash \hat{\alpha}[\tau/\hat{\gamma}] \sim \hat{\beta}[\tau/\hat{\gamma}] \text{ ④}.$$

By definition of metasubstitution and ③,

$$\Theta_0, \hat{\gamma} := \tau \sqsubseteq \Theta_1, \hat{\gamma} := \tau \text{ ⑤}.$$

By a straightforward subsidiary induction on ④ with declarative consistency rules and weakening, we have

$$\Theta_1, \hat{\gamma} := \tau \vdash \hat{\alpha} \sim \hat{\beta} \text{ ⑥}.$$

Our goal follows from ⑤ and ⑥.

Case U-FLEX-FLEX-SKIP and U-FLEX-FLEX-SKIP'. Our goal follows from IH and weakening.

3. Assignment. We prove that $\Theta_0, \Xi \sqsubseteq \Theta_1$ is a solution by induction on the assignment derivation

$$\Theta_0 \mid \Xi \vdash \hat{\alpha} := P \dashv \Theta_1.$$

Case U-ASSIGN-UNIVGHOST-ID. Our goal follows from reflexivity of information increase, reflexivity of consistency, and C-UNIVGHOST2.

Case

$$\frac{\text{U-ASSIGN-SOLVEM} \quad \hat{\alpha} \notin \text{ftv}(P) \quad \text{guess}(P) = (\bar{\beta}; \tau)}{\Theta_0, \hat{\alpha} \mid \Xi \vdash \hat{\alpha} := P \dashv \Theta_0, \Xi, \bar{\beta}, \hat{\alpha} := \tau}$$

Let $\Theta_1 = \Theta_0, \Xi, \bar{\beta}, \hat{\alpha} := \tau$. By definition of information increase,

$$\Theta_0, \hat{\alpha}, \Xi \sqsubseteq \Theta_1 \text{ ②}.$$

By definition of $\text{guess}(P)$, we have

$$\Theta_1 \vdash P \sim \tau \text{ ③}.$$

Then by $\Theta_1 \ni \hat{\alpha} := \tau$, declarative C-FLEXR on ③ gives

$$\Theta_1 \vdash P \sim \hat{\alpha} \text{ ④.}$$

Our goal follows from ② and ④.

Case

$$\frac{\text{U-ASSIGN-ASSIGN} \quad \Theta_0, \Xi \vdash \hat{\alpha}[\tau/\hat{\beta}] \sim P[\tau/\hat{\beta}] \dashv \Theta_1 \text{ ②}}{\Theta_0, \hat{\beta} := \tau \mid \Xi \vdash \hat{\alpha} := P \dashv \Theta_1, \hat{\beta} := \tau}$$

② expands to, for some Q ,

$$\Theta_0, \Xi [P[\tau/\hat{\beta}]] \vdash \hat{\alpha}[\tau/\hat{\beta}] \sim Q \dashv \Theta_1 \text{ ③.}$$

By the IH of consistency on ③, we have

$$\Theta_0, \Xi \sqsubseteq \Theta_1 \text{ ④} \quad \Theta_1 \vdash P[\tau/\hat{\beta}] \blacktriangleright Q \text{ ⑤} \quad \Theta_1 \vdash \hat{\alpha}[\tau/\hat{\beta}] \sim Q \text{ ⑥.}$$

By ④ and definition of information increase, we have

$$\Theta_0, \hat{\beta} := \tau, \Xi \sqsubseteq \Theta_1, \hat{\beta} := \tau \text{ ⑦.}$$

By ⑤, ⑥, and a straightforward subsidiary induction, we have

$$\Theta_1 \vdash P[\tau/\hat{\beta}] \sim \hat{\alpha}[\tau/\hat{\beta}] \text{ ⑧,}$$

By another straightforward subsidiary induction on ⑧ with C-FLEXL, C-FLEXR, and weakening, we have

$$\Theta_1, \hat{\beta} := \tau \vdash P \sim \hat{\alpha} \text{ ⑨,}$$

Our goal follows from ⑦ and ⑨.

Case

$$\frac{\text{U-ASSIGN-DEPEND} \quad \hat{\alpha} \neq \hat{\beta} \quad \hat{\beta} \in \text{ftv}(P) \quad \Theta_0 \mid \hat{\beta}, \Xi \vdash \hat{\alpha} := P \dashv \Theta_1 \text{ ②}}{\Theta_0, \hat{\beta} \mid \Xi \vdash \hat{\alpha} := P \dashv \Theta_1}$$

Our goal follows from IH on ②.

Case U-ASSIGN-SKIP, U-ASSIGN-SKIPRIGID, and U-ASSIGN-SKIPTMVAR. Our goal follows from IH and context weakening.

1. Consistency. We prove the augmented statement above by induction on the algorithmic consistency derivation.

Case

$$\frac{\text{U-GHOST} \quad \Delta_2 \cap \text{ftv}(P_0) = \emptyset}{\Theta_0 [P_0]_{\Delta_1 \mid \Delta_2} \hat{\mathbb{L}} \sim P_0 \dashv \Theta_0}$$

We have $\Theta_0 \sqsubseteq \Theta_0$ by reflexivity. By C-GHOST, we have

$$\Theta_0 \vdash \mathbb{I} \sim P_0 \text{ (1)}.$$

By reflexivity of refinement and the side condition, we have

$$\Theta_0 \mid \Delta_1 \vdash P_0 \blacktriangleright P_0 \text{ (2)} \quad \Theta_0 \mid \Delta_2 \vdash \mathbb{I} \blacktriangleright P_0 \text{ (3)}.$$

Our goal follows from (1), (2), and (3).

Case U-UNIT and U-RIGID. Our goals follow from reflexivity of information increase, reflexivity of refinement, and rules C-UNIT and C-VAR.

Case

U-ARROW

$$\frac{\Theta_0 [P'_1]_{\Delta_1/\Delta_2} P_1 \sim Q_1 \vdash \Theta_1 \text{ (1)} \quad \Theta_1 [P'_2]_{\Delta_1/\Delta_2} P_2 \sim Q_2 \vdash \Theta_2 \text{ (2)}}{\Theta_0 [P'_1 \rightarrow P'_2]_{\Delta_1/\Delta_2} P_1 \rightarrow P_2 \sim Q_1 \rightarrow Q_2 \vdash \Theta_2}$$

IH on (1) gives

$$\Theta_0 \sqsubseteq \Theta_1 \text{ (3)} \quad \Theta_1 \vdash P_1 \sim Q_1 \text{ (4)} \quad \Theta_1 \mid \Delta_1 \vdash P'_1 \blacktriangleright Q_1 \text{ (5)} \quad \Theta_1 \mid \Delta_2 \vdash P_1 \blacktriangleright Q_1 \text{ (6)}.$$

IH on (2) gives

$$\Theta_1 \sqsubseteq \Theta_2 \text{ (7)} \quad \Theta_2 \vdash P_2 \sim Q_2 \text{ (8)} \quad \Theta_2 \mid \Delta_1 \vdash P'_2 \blacktriangleright Q_2 \text{ (9)} \quad \Theta_2 \mid \Delta_2 \vdash P_2 \blacktriangleright Q_2 \text{ (10)}.$$

Our goal follows from stability on (4), (5), and (6) along (7), rule C-ARROW, and definition of refinement.

Case

U-FORALLINH

$$\frac{\Theta_0, \alpha [P_0]_{\Delta_1, \alpha/\Delta_2, \alpha} P \sim Q \vdash \Theta_1, \alpha \text{ (1)}}{\Theta_0 [\forall \alpha. P_0]_{\Delta_1/\Delta_2} \forall \alpha. P \sim \forall \alpha. Q \vdash \Theta_1}$$

IH on (1) gives

$$\begin{aligned} \Theta_0, \alpha \sqsubseteq \Theta_1, \alpha \text{ (2)} \quad \Theta_1, \alpha \vdash P \sim Q \text{ (3)} \\ \Theta_1, \alpha \mid \Delta_1, \alpha \vdash P_0 \blacktriangleright Q \text{ (4)} \quad \Theta_1, \alpha \mid \Delta_2, \alpha \vdash P \blacktriangleright Q \text{ (5)}. \end{aligned}$$

By removing the type variable α from (2), we have

$$\Theta_0 \sqsubseteq \Theta_1 \text{ (6)}.$$

By C-FORALLINH on (3), we have

$$\Theta_1 \vdash \forall \alpha. P \sim \forall \alpha. Q \text{ (7)}.$$

By definition of refinement on (4), we have

$$\Theta_1 \mid \Delta_1 \vdash \forall \alpha. P_0 \blacktriangleright \forall \alpha. Q \text{ (8)}.$$

By definition of refinement on ⑤, we have

$$\Theta_1 \mid \Delta_2 \vdash \forall \alpha. P \blacktriangleright \forall \alpha. Q \text{ ⑨}.$$

Well-formedness of $\forall \alpha. Q$ comes from well-formedness of $\forall \alpha. P$ and $\forall \alpha. P_0$. Our goal follows from ⑥, ⑦, ⑧, and ⑨.

Case

$$\begin{array}{c} \text{U-FORALLSYN} \\ \hline \Theta_0, \alpha \mid \Delta_1.P_0 \vdash_{\Delta_1|\Delta_2, \alpha} P \sim Q \dashv \Theta_1, \alpha \text{ ①} \\ \hline \Theta_0 \mid \Delta_1.P_0 \vdash_{\Delta_1|\Delta_2} \forall \alpha. P \sim \forall \alpha. Q \dashv \Theta_1 \end{array}$$

IH on ① gives

$$\begin{array}{ll} \Theta_0, \alpha \sqsubseteq \Theta_1, \alpha \text{ ②} & \Theta_1, \alpha \vdash P \sim Q \text{ ③} \\ \Theta_1, \alpha \mid \Delta_1 \vdash \Delta_1.P_0 \blacktriangleright Q \text{ ④} & \Theta_1, \alpha \mid \Delta_2, \alpha \vdash P \blacktriangleright Q \text{ ⑤}. \end{array}$$

By removing the type variable α from ②, we have

$$\Theta_0 \sqsubseteq \Theta_1 \text{ ⑥}.$$

By C-FORALLSYN on ③, we have

$$\Theta_1 \vdash \forall \alpha. P \sim \forall \alpha. Q \text{ ⑦}.$$

By definition of refinement on ④, we have

$$\Theta_1 \mid \Delta_1 \vdash \Delta_1.P_0 \blacktriangleright \forall \alpha. Q \text{ ⑧}.$$

By definition of refinement on ⑤, we have

$$\Theta_1 \mid \Delta_2 \vdash \forall \alpha. P \blacktriangleright \forall \alpha. Q \text{ ⑨}.$$

Well-formedness of $\forall \alpha. Q$ comes from well-formedness of $\forall \alpha. P$ and $\Delta_1.P_0$. Our goal follows from ⑥, ⑦, ⑧, and ⑨.

Case

$$\begin{array}{c} \text{U-UNIVGHOST} \\ \hline \Theta_0 \mid P_0 \vdash_{\Delta_1|\Delta_2} P \sim Q \dashv \Theta_1 \text{ ①} \\ \hline \Theta_0 \mid \Delta_1.P_0 \vdash_{\Delta_1|\Delta_2} \Delta_1.P \sim \Delta_1.Q \dashv \Theta_1 \end{array}$$

IH on ① gives

$$\Theta_0 \sqsubseteq \Theta_1 \text{ ②} \quad \Theta_1 \vdash P \sim Q \text{ ③} \quad \Theta_1 \mid \Delta_1 \vdash P_0 \blacktriangleright Q \text{ ④} \quad \Theta_1 \mid \Delta_2 \vdash P \blacktriangleright Q \text{ ⑤}.$$

By C-UNIVGHOST on ③, we have

$$\Theta_1 \vdash \Delta_1.P \sim \Delta_1.Q \text{ ⑥}.$$

By definition of refinement on ④, we have

$$\Theta_1 \mid \Delta_1 \vdash \Delta_1.P_0 \blacktriangleright \Delta_1.Q \text{ ⑦}.$$

By definition of refinement on (5), we have

$$\Theta_1 \mid \Delta_2 \vdash \mathbb{A}.P \blacktriangleright \mathbb{A}.Q \text{ (8)}.$$

Well-formedness of $\mathbb{A}.Q$ comes from well-formedness of $\mathbb{A}.P$ and $\mathbb{A}.P_0$. Our goal follows from (2), (6), (7), and (8).

Case

$$\begin{array}{c} \text{U-UNIVGHOST1} \\ \hline \Theta_0, \alpha \mid P_0 \vdash_{\Delta_1, \alpha \mid \Delta_2} \mathbb{A}.P \sim Q \dashv \Theta_1, \alpha \text{ (1)} \\ \hline \Theta_0 \mid \forall \alpha. P_0 \vdash_{\Delta_1 \mid \Delta_2} \mathbb{A}.P \sim \forall \alpha. Q \dashv \Theta_1 \end{array}$$

IH on (1) gives

$$\begin{array}{ll} \Theta_0, \alpha \sqsubseteq \Theta_1, \alpha \text{ (2)} & \Theta_1, \alpha \vdash \mathbb{A}.P \sim Q \text{ (3)} \\ \Theta_1, \alpha \mid \Delta_1, \alpha \vdash P_0 \blacktriangleright Q \text{ (4)} & \Theta_1, \alpha \mid \Delta_2 \vdash \mathbb{A}.P \blacktriangleright Q \text{ (5)}. \end{array}$$

By removing the type variable α from (2), we have

$$\Theta_0 \sqsubseteq \Theta_1 \text{ (6)}.$$

By C-UNIVGHOST1 on (3), we have

$$\Theta_1 \vdash \mathbb{A}.P \sim \forall \alpha. Q \text{ (7)}.$$

By definition of refinement on (4), we have

$$\Theta_1 \mid \Delta_1 \vdash \forall \alpha. P_0 \blacktriangleright \forall \alpha. Q \text{ (8)}.$$

By definition of refinement on (5), we have

$$\Theta_1 \mid \Delta_2 \vdash \mathbb{A}.P \blacktriangleright \forall \alpha. Q \text{ (9)}.$$

Well-formedness of $\forall \alpha. Q$ comes from well-formedness of $\mathbb{A}.P$ and $\forall \alpha. P_0$. Our goal follows from (6), (7), (8), and (9).

Case

$$\begin{array}{c} \text{U-UNIVGHOST2} \\ \hline \Theta_0 \vdash P_0 \text{ guarded} \quad P_0 \text{ non-flex} \quad \Theta_0 \mid P_0 \vdash_{\Delta_1 \mid \Delta_2} P \sim Q \dashv \Theta_1 \text{ (1)} \\ \hline \Theta_0 \mid P_0 \vdash_{\Delta_1 \mid \Delta_2} \mathbb{A}.P \sim Q \dashv \Theta_1 \end{array}$$

IH on (1) gives

$$\Theta_0 \sqsubseteq \Theta_1 \text{ (2)} \quad \Theta_1 \vdash P \sim Q \text{ (3)} \quad \Theta_1 \mid \Delta_1 \vdash P_0 \blacktriangleright Q \text{ (4)} \quad \Theta_1 \mid \Delta_2 \vdash P \blacktriangleright Q \text{ (5)}.$$

By C-UNIVGHOST2 on (3), we have

$$\Theta_1 \vdash \mathbb{A}.P \sim Q \text{ (6)}.$$

By definition of refinement on (5), we have

$$\Theta_1 \mid \Delta_2 \vdash \mathbb{A}.P \blacktriangleright Q \text{ (7)}.$$

Our goal follows from (2), (6), (4), and (7).

Case

$$\frac{\text{U-FLEX} \quad \Theta_0 \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta_1 \text{ (1)}}{\Theta_0 [\hat{\beta}] \vdash_{\Delta_1|\Delta_2} \hat{\alpha} \sim \hat{\alpha} \dashv \Theta_1}$$

By IH of variable unification on (1), we have

$$\Theta_0 \sqsubseteq \Theta_1 \text{ (2)} \quad \Theta_1 \vdash \hat{\alpha} \sim \hat{\beta} \text{ (3)}.$$

By (3) we have $\Theta_1 \mid \Delta_1 \vdash \hat{\beta} \blacktriangleright \hat{\alpha}$ (4), since flexible variables do not depend on rigid type variables in Δ_1 . By reflexivity of refinement, we have $\Theta_1 \mid \Delta_2 \vdash \hat{\alpha} \blacktriangleright \hat{\alpha}$ (5), since flexible variables do not depend on rigid type variables in Δ_2 . Our goal follows from (2), (4), (5), and $\Theta_1 \vdash \hat{\alpha} \sim \hat{\alpha}$.

Case

$$\frac{\text{U-FLEXL} \quad P_0 \text{ non-flex} \quad \Theta_0 \mid \cdot \vdash \hat{\alpha} := P_0 \dashv \Theta_1 \text{ (1)}}{\Theta_0 [P_0] \vdash_{\Delta_1|\Delta_2} \hat{\alpha} \sim \hat{\alpha} \dashv \Theta_1}$$

By IH of assignment on (1), we have

$$\Theta_0 \sqsubseteq \Theta_1 \text{ (2)} \quad \Theta_1 \vdash P_0 \sim \hat{\alpha} \text{ (3)}.$$

By (3), we have $\Theta_1 \mid \Delta_1 \vdash P_0 \blacktriangleright \hat{\alpha}$ (4), since flexible variables do not depend on rigid type variables in Δ_1 . By reflexivity of refinement, we have $\Theta_1 \mid \Delta_2 \vdash \hat{\alpha} \blacktriangleright \hat{\alpha}$ (5), since flexible variables do not depend on rigid type variables in Δ_2 . Our goal follows from (2), (4), (5), and $\Theta_1 \vdash \hat{\alpha} \sim \hat{\alpha}$.

Case

$$\frac{\text{U-FLEXR} \quad P \text{ non-flex} \quad \Theta_0 \mid \cdot \vdash \hat{\alpha} := P \dashv \Theta_1 \text{ (1)}}{\Theta_0 [\hat{\alpha}] \vdash_{\Delta_1|\Delta_2} P \sim \hat{\alpha} \dashv \Theta_1}$$

By IH of assignment on (1), we have

$$\Theta_0 \sqsubseteq \Theta_1 \text{ (2)} \quad \Theta_1 \vdash P \sim \hat{\alpha} \text{ (3)}.$$

By reflexivity of refinement, we have $\Theta_1 \mid \Delta_1 \vdash \hat{\alpha} \blacktriangleright \hat{\alpha}$ (4). By (3), we have $\Theta_1 \mid \Delta_2 \vdash P \blacktriangleright \hat{\alpha}$ (5). Our goal follows from (2), (3), (4), and (5).

Case

$$\frac{\text{U-GHOSTR} \quad \Delta_1 \cap \text{ftv}(P) = \emptyset}{\Theta_0 [\hat{\alpha}] \vdash_{\Delta_1|\Delta_2} P \sim P \dashv \Theta_0}$$

Our goal follows from $\Theta_0 \sqsubseteq \Theta_0$, the side condition, definition of refinement, and $\Theta_0 \vdash P \sim P$.

Case

$$\frac{\text{U-UNIVGHOSTR} \quad \Theta_0 \vdash P \text{ guarded} \quad P \text{ non-flex} \quad \Theta_0 [P_0]_{\Delta_1|\Delta_2} P \sim Q \dashv \Theta_1 \text{ ①}}{\Theta_0 [\hat{\Delta}.P_0]_{\Delta_1|\Delta_2} P \sim Q \dashv \Theta_1}$$

IH on ① gives

$$\Theta_0 \sqsubseteq \Theta_1 \text{ ②} \quad \Theta_1 \vdash P \sim Q \text{ ③} \quad \Theta_1 \mid \Delta_1 \vdash P_0 \blacktriangleright Q \text{ ④} \quad \Theta_1 \mid \Delta_2 \vdash P \blacktriangleright Q \text{ ⑤}.$$

By definition of refinement on ④, we have

$$\Theta_1 \mid \Delta_1 \vdash \hat{\Delta}.P_0 \blacktriangleright Q \text{ ⑥}.$$

Our goal follows from ②, ③, ⑤, and ⑥.

□

D.5.3 Completeness of consistency

We prove Lemma 26.

Lemma 26 (Completeness and optimality of consistency).

1. Given a problem of consistency $(\Theta_0; P_0; P)$, if there exists a solution to it, then $\Theta_0 [P_0]_{\Delta_1|\Delta_2} P \sim P_1 \dashv \Theta_1$ gives its optimal solution $(\Theta_0 \sqsubseteq \Theta_1; P_1)$.
2. Given a problem of variable unification $(\Theta_0; \hat{\alpha}; \hat{\beta})$, if there exists a solution to it, then $\Theta_0 \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta_1$ gives its optimal solution $\Theta_0 \sqsubseteq \Theta_1$.
3. Given a problem of assignment $(\Theta_0; \Xi; \hat{\alpha}; P)$, if there exists a solution to it, then $\Theta_0 \mid \Xi \vdash \hat{\alpha} := P \dashv \Theta_1$ gives its optimal solution $\Theta_0, \Xi \sqsubseteq \Theta_1$.

Proof. We first augment the definition of a consistency problem to $(\Delta_1; \Delta_2; \Psi; \Theta_0; P_0; S)$ where Δ_1 and Δ_2 are contexts of rigid type variables present in Θ_0 that no flexible variables depend on. We augment the definition of a solution $(\theta :: \Theta_0 \sqsubseteq \Theta; Q)$ to require $\Psi \mid \Theta \vdash \theta S \sim Q$ and $\Theta \mid \Delta_1 \vdash \theta P_0 \blacktriangleright Q$ and $\Theta \mid \Delta_2 \vdash \theta [S] \blacktriangleright Q$ and that no flexible variables in Θ depend on variables in Δ_1 and Δ_2 . With the augmented definitions, for the first part of this lemma for consistency, it is enough to prove that, if $(\theta :: \Theta_0 \sqsubseteq \Theta; Q)$ is a solution to the consistency problem $(\Delta_1; \Delta_2; \Psi; \Theta_0; P_0; S)$, with $P = [S]$, we have

$$\Theta_0 [P_0]_{\Delta_1|\Delta_2} P \sim P_1 \dashv \Theta_1 \quad \theta_1 :: \Theta_1 \sqsubseteq \Theta \quad \theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \quad \Theta \mid \Delta_1, \Delta_2 \vdash \theta_1 P_1 \leq Q$$

We proceed by mutual induction for the three parts of this lemma, with the first part augmented as above.

2. Unification of flexible variables. It is enough to prove that given a unification problem $(\Theta_0; \hat{\alpha}; \hat{\beta})$ and a solution

$$\theta :: \Theta_0 \sqsubseteq \Theta \text{ ①} \quad \Theta \vdash \theta \hat{\alpha} \sim \theta \hat{\beta} \text{ ②},$$

there exists θ_1 and Θ_1 such that

$$\Theta_0 \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta_1 \quad \theta_1 :: \Theta_1 \sqsubseteq \Theta \quad \theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta.$$

By induction on the context Θ_0 and case analysis on which algorithmic unification rule to use. We first discuss the special case where $\hat{\alpha} = \hat{\beta}$.

Case

$$\begin{array}{c} \text{U-FLEX-FLEX-ID} \\ \hat{\alpha} \in \text{dom}(\Theta_0) \\ \hline \Theta_0 \vdash \hat{\alpha} \simeq \hat{\alpha} \dashv \Theta_0 \end{array}$$

Trivial.

Case

$$\begin{array}{c} \text{U-FLEX-FLEX-L} \\ \hat{\alpha} \neq \hat{\beta} \\ \hline \Theta'_0, \hat{\alpha} \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta'_0, \hat{\alpha} := \hat{\beta} \end{array}$$

In this case $\Theta_0 = \Theta'_0, \hat{\alpha}$. By ②, since both sides are monotypes, we have

$$\Theta \vdash \theta \hat{\alpha} \equiv \theta \hat{\beta} \text{ ③}.$$

By the definition of metasubstitution, ① and ③, we have

$$\theta :: \Theta'_0, \hat{\alpha} := \hat{\beta} \sqsubseteq \Theta \text{ ④}.$$

Our goal follows from U-FLEX-FLEX-L, ④, and $\theta \equiv \theta :: \Theta_0 \sqsubseteq \Theta$

Case

$$\begin{array}{c} \text{U-FLEX-FLEX-R} \\ \hat{\alpha} \neq \hat{\beta} \\ \hline \Theta'_0, \hat{\beta} \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta'_0, \hat{\beta} := \hat{\alpha} \end{array}$$

Similar to the case of U-FLEX-FLEX-L.

Case

$$\begin{array}{c} \text{U-FLEX-FLEX-ASSIGN} \\ \Theta'_0 \vdash \hat{\alpha}[\tau/\hat{\gamma}] \sim \hat{\beta}[\tau/\hat{\gamma}] \dashv \Theta'_1 \\ \hline \Theta'_0, \hat{\gamma} := \tau \vdash \hat{\alpha} \simeq \hat{\beta} \dashv \Theta'_1, \hat{\gamma} := \tau \end{array}$$

In this case $\Theta_0 = \Theta'_0, \hat{\gamma} := \tau$. By ① and definition of metasubstitution, we have

$$\Theta \vdash \theta \hat{\gamma} \equiv \theta \tau \text{ ③} \quad \theta :: \Theta'_0 \sqsubseteq \Theta \text{ ④}.$$

By ② and definition of metasubstitution, we have

$$\Theta \vdash \theta(\hat{\alpha}[\tau/\hat{\gamma}]) \sim \theta(\hat{\beta}[\tau/\hat{\gamma}]) \text{ ⑤.}$$

By definition of refinement, we have

$$\Theta \mid \cdot \vdash \theta(\hat{\beta}[\tau/\hat{\gamma}]) \blacktriangleright \theta(\hat{\beta}[\tau/\hat{\gamma}]) \text{ ⑥.}$$

④, ⑤, and ⑥ together give a solution to the consistency problem $(\cdot; \cdot; \cdot; \Theta'_0; \hat{\beta}[\tau/\hat{\gamma}]; \hat{\alpha}[\tau/\hat{\gamma}])$.

By the IH of consistency on this solution, we have, for some Q ,

$$\Theta'_0 \mid \hat{\beta}[\tau/\hat{\gamma}] \vdash \hat{\alpha}[\tau/\hat{\gamma}] \sim Q \vdash \Theta'_1 \text{ ⑦}$$

$$\theta_1 :: \Theta'_1 \sqsubseteq \Theta \text{ ⑧}$$

$$\Theta \mid \cdot \vdash \theta_1 Q \leq \theta(\hat{\beta}[\tau/\hat{\gamma}]) \text{ ⑨}$$

$$\theta \equiv \theta_1 :: \Theta'_0 \sqsubseteq \Theta \text{ ⑩}$$

By U-FLEX-FLEX-ASSIGN on ⑦, we have

$$\Theta'_0, \hat{\gamma} := \tau \vdash \hat{\alpha} \simeq \hat{\beta} \vdash \Theta'_1, \hat{\gamma} := \tau \text{ ⑪.}$$

By definition of metasubstitution, ⑧, ⑩, and ③, we extend θ_1 such that

$$\theta_1 :: \Theta'_1, \hat{\gamma} := \tau \sqsubseteq \Theta \text{ ⑫.} \quad \theta \equiv \theta_1 :: \Theta'_0, \hat{\gamma} := \tau \sqsubseteq \Theta \text{ ⑬.}$$

Our goal follows from ⑪, ⑫, and ⑬.

Case U-FLEX-FLEX-SKIP and U-FLEX-FLEX-SKIP'. Our goal follows from IH and weakening.

3. Assignment. It is enough to prove that given an assignment problem $(\Theta_0; \Xi; \hat{\alpha}; P)$ and a solution

$$\theta :: \Theta_0, \Xi \sqsubseteq \Theta \text{ ①} \quad \Theta \vdash \theta P \sim \theta \hat{\alpha} \text{ ②.}$$

there exist Θ_1 and θ_1 such that

$$\Theta_0 \mid \Xi \vdash \hat{\alpha} := P \vdash \Theta_1 \quad \theta_1 :: \Theta_1 \sqsubseteq \Theta \quad \theta \equiv \theta_1 :: \Theta_0, \Xi \sqsubseteq \Theta.$$

By induction on the pair of contexts $\Theta_0 \mid \Xi$ and case analysis on which algorithmic assignment rule to use.

Case U-ASSIGN-UNIVGHOST-ID. Our goal follows from U-ASSIGN-UNIVGHOST-ID, ①, and reflexivity of metasubstitution equivalence by taking $\theta_1 = \theta$.

Case

$$\frac{\text{U-ASSIGN-SOLVEM} \quad \hat{\alpha} \notin \text{ftv}(P) \quad \text{guess}(P) = (\bar{\hat{\beta}}; \tau)}{\Theta'_0, \hat{\alpha} \mid \Xi \vdash \hat{\alpha} := P \vdash \Theta'_0, \Xi, \bar{\hat{\beta}}, \hat{\alpha} := \tau}$$

In this case $\Theta_0 = \Theta'_0, \hat{\alpha}$. Let $\text{guess}(P) = (\bar{\beta}; \tau)$ ③. By definition of $\text{guess}(P)$, ③, and ②, we can construct θ_1 by extending θ with proper substitutions for $\bar{\beta}$ such that $\Theta \vdash \theta_1 \tau \equiv \theta \hat{\alpha}$. Thus we have

$$\theta_1 :: \Theta'_0, \Xi, \bar{\beta}, \hat{\alpha} := \tau \sqsubseteq \Theta \text{ ④} \quad \theta \equiv \theta_1 :: \Theta'_0, \hat{\alpha}, \Xi \sqsubseteq \Theta \text{ ⑤}.$$

Our goal follows from U-ASSIGN-SOLVEM, ④, and ⑤.

Case

$$\begin{array}{c} \text{U-ASSIGN-ASSIGN} \\ \frac{\Theta'_0, \Xi \vdash \hat{\alpha}[\tau/\hat{\beta}] \sim P[\tau/\hat{\beta}] \vdash \Theta'_1}{\Theta'_0, \hat{\beta} := \tau \mid \Xi \vdash \hat{\alpha} := P \vdash \Theta'_1, \hat{\beta} := \tau} \end{array}$$

In this case $\Theta_0 = \Theta'_0, \hat{\beta} := \tau$. By ① and definition of metasubstitution, we have

$$\Theta \vdash \theta \hat{\beta} \equiv \theta \tau \text{ ③} \quad \theta :: \Theta'_0, \Xi \sqsubseteq \Theta \text{ ④}.$$

By ② and definition of metasubstitution, we have

$$\Theta \vdash \theta(P[\tau/\hat{\beta}]) \sim \theta(\hat{\alpha}[\tau/\hat{\beta}]) \text{ ⑤}.$$

By ⑤, we have

$$\Theta \mid \cdot \vdash \theta(P[\tau/\hat{\beta}]) \blacktriangleright \theta(\hat{\alpha}[\tau/\hat{\beta}]) \text{ ⑥}. \quad \Theta \vdash \theta(\hat{\alpha}[\tau/\hat{\beta}]) \sim \theta(\hat{\alpha}[\tau/\hat{\beta}]) \text{ ⑦}.$$

④, ⑥, and ⑦ together form a solution to the consistency problem

$$(\cdot; \cdot; \cdot; \Theta'_0, \Xi; P[\tau/\hat{\beta}]; \hat{\alpha}[\tau/\hat{\beta}]).$$

By the IH of consistency on this solution, for some Q , we have

$$\begin{array}{l} \Theta'_0, \Xi \vdash [P[\tau/\hat{\beta}]] \vdash \hat{\alpha}[\tau/\hat{\beta}] \sim Q \vdash \Theta'_1 \text{ ⑧} \\ \theta_1 :: \Theta'_1 \sqsubseteq \Theta \text{ ⑨} \\ \Theta \mid \cdot \vdash \theta_1 Q \leq \theta(\hat{\alpha}[\tau/\hat{\beta}]) \text{ ⑩} \\ \theta \equiv \theta_1 :: \Theta'_0, \Xi \sqsubseteq \Theta \text{ ⑪} \end{array}$$

By U-ASSIGN-ASSIGN on ⑧, we have

$$\Theta'_0, \hat{\beta} := \tau \mid \Xi \vdash \hat{\alpha} := P \vdash \Theta'_1, \hat{\beta} := \tau \text{ ⑫}.$$

By definition of metasubstitution, ③, ⑨, and ⑪, we have

$$\theta_1 :: \Theta'_1, \hat{\beta} := \tau \sqsubseteq \Theta \text{ ⑬}. \quad \theta \equiv \theta_1 :: \Theta'_0, \hat{\beta} := \tau, \Xi \sqsubseteq \Theta \text{ ⑭}.$$

Our goal follows from ⑫, ⑬, and ⑭.

Case

$$\begin{array}{c} \text{U-ASSIGN-DEPEND} \\ \frac{\hat{\alpha} \neq \hat{\beta} \quad \hat{\beta} \in \text{ftv}(P) \quad \Theta'_0 \mid \hat{\beta}, \Xi \vdash \hat{\alpha} := P \vdash \Theta_1}{\Theta'_0, \hat{\beta} \mid \Xi \vdash \hat{\alpha} := P \vdash \Theta_1} \end{array}$$

The solution ① is also a solution to the problem $(\Theta'_0; \hat{\beta}, \Xi; \hat{\alpha}; P)$. Our goal follows from IH and U-ASSIGN-DEPEND.

Case U-ASSIGN-SKIP, U-ASSIGN-SKIPRIGID, and U-ASSIGN-SKIPTMVAR. Our goal follows from IH and context weakening.

1. Consistency. We prove for the augmented consistency problem and solution as stated at the beginning of this proof. We prove by induction on the declarative consistency derivation $\Psi \mid \Theta \vdash \theta S \sim Q$. The cases where P or P_0 is a flexible variable are handled by the other two parts of the lemma. If $P_0 = \mathbb{A}$, we use U-GHOSTR. If $P_0 = \mathbb{A}.P'_0$ while P is guarded, we use U-UNIVGHOSTR and continue with the problem for P'_0 . In the induction below, we assume the above three special cases have already been handled.

Case

$$\frac{\text{C-GHOST}}{\cdot \mid \Theta \vdash \mathbb{A} \sim Q}$$

In this case $S = \mathbb{A}$ and $P = \mathbb{A}$. The solution gives

$$\theta :: \Theta_0 \sqsubseteq \Theta \text{ ①} \quad \Theta \mid \Delta_1 \vdash \theta P_0 \blacktriangleright Q \text{ ②} \quad \Theta \mid \Delta_2 \vdash \theta \mathbb{A} \blacktriangleright Q \text{ ③}.$$

By ② and ③, the side condition of U-GHOST holds. Thus by U-GHOST, we have

$$\Theta_0 [P_0]_{\Delta_1 \Delta_2} \mathbb{A} \sim P_0 \dashv \Theta_0 \text{ ④}.$$

By ② and ③, we have

$$\Theta \mid \Delta_1, \Delta_2 \vdash \theta P_0 \leq Q \text{ ⑤}.$$

Our goal follows from ④, ①, ⑤, and reflexivity of metasubstitution equivalence.

Case

$$\frac{\text{C-GHOSTG}}{\cdot \mid \Theta \vdash \mathbb{A} \sim \mathbb{A}}$$

In this case $S = \mathbb{A}$ and $P = \mathbb{A}$. The solution gives

$$\theta :: \Theta_0 \sqsubseteq \Theta \text{ ①} \quad \Theta \mid \Delta_1 \vdash \theta P_0 \blacktriangleright \mathbb{A} \text{ ②} \quad \Theta \mid \Delta_2 \vdash \theta \mathbb{A} \blacktriangleright \mathbb{A} \text{ ③}.$$

By inversion on ②, $P_0 = \mathbb{A}$. Thus by U-GHOST, we have

$$\Theta_0 [\mathbb{A}]_{\Delta_1 \Delta_2} \mathbb{A} \sim \mathbb{A} \dashv \Theta_0 \text{ ④}.$$

By reflexivity of precision, we have

$$\Theta \mid \Delta_1, \Delta_2 \vdash \theta \mathbb{A} \leq \mathbb{A} \text{ ⑤}.$$

Our goal follows from ④, ①, ⑤, and reflexivity of metasubstitution equivalence.

Case

$$\begin{array}{c} \text{C-UNIT} \\ \hline \cdot \mid \Theta \vdash 1 \sim 1 \end{array}$$

In this case $S = 1$ and $P = 1$. The solution gives

$$\theta :: \Theta_0 \sqsubseteq \Theta \text{ (1)} \quad \Theta \mid \Delta_1 \vdash \theta P_0 \blacktriangleright 1 \text{ (2)} \quad \Theta \mid \Delta_2 \vdash \theta 1 \blacktriangleright 1 \text{ (3)}.$$

Since P_0 cannot be a ghost or flexible variable, inversion on (2) gives $P_0 = 1$. Our goal follows from U-UNIT, (1), and reflexivity of precision and metasubstitution equivalence.

Case

$$\begin{array}{c} \text{C-VAR} \\ \Theta \ni \alpha \\ \hline \cdot \mid \Theta \vdash \alpha \sim \alpha \end{array}$$

In this case $S = \alpha$ and $P = \alpha$. The solution gives

$$\theta :: \Theta_0 \sqsubseteq \Theta \text{ (1)} \quad \Theta \mid \Delta_1 \vdash \theta P_0 \blacktriangleright \alpha \text{ (2)} \quad \Theta \mid \Delta_2 \vdash \theta \alpha \blacktriangleright \alpha \text{ (3)}.$$

Since P_0 cannot be a ghost or flexible variable, inversion on (2) gives $P_0 = \alpha$. By inversion on (1), $\Theta_0 \ni \alpha$. Our goal follows from U-RIGID, (1), and reflexivity of precision and metasubstitution equivalence.

Case

$$\begin{array}{c} \text{C-ARROW} \\ \Psi_1 \mid \Theta \vdash \theta S_1 \sim Q_1 \text{ (1)} \quad \Psi_2 \mid \Theta \vdash \theta S_2 \sim Q_2 \text{ (2)} \\ \hline \Psi_1, \Psi_2 \mid \Theta \vdash \theta(S_1 \rightarrow S_2) \sim Q_1 \rightarrow Q_2 \end{array}$$

In this case $S = S_1 \rightarrow S_2$ and $P = P_1 \rightarrow P_2$ where $P_1 = \lfloor S_1 \rfloor$ and $P_2 = \lfloor S_2 \rfloor$. The solution gives

$$\theta :: \Theta_0 \sqsubseteq \Theta \text{ (3)} \quad \Theta \mid \Delta_1 \vdash \theta P_0 \blacktriangleright Q_1 \rightarrow Q_2 \text{ (4)} \quad \Theta \mid \Delta_2 \vdash \theta(P_1 \rightarrow P_2) \blacktriangleright Q_1 \rightarrow Q_2 \text{ (5)}.$$

Since P_0 cannot be a ghost or flexible variable, inversion on (4) gives $P_0 = P'_1 \rightarrow P'_2$ and

$$\Theta \mid \Delta_1 \vdash \theta P'_1 \blacktriangleright Q_1 \text{ (6)} \quad \Theta \mid \Delta_1 \vdash \theta P'_2 \blacktriangleright Q_2 \text{ (7)}.$$

By inversion on (5), we have

$$\Theta \mid \Delta_2 \vdash \theta P_1 \blacktriangleright Q_1 \text{ (8)} \quad \Theta \mid \Delta_2 \vdash \theta P_2 \blacktriangleright Q_2 \text{ (9)}.$$

IH on the left component with (3), (1), (6), (8), and $P_1 = \lfloor S_1 \rfloor$ gives

$$\begin{array}{l} \Theta_0 \mid P'_1 \vdash_{\Delta_1 \mid \Delta_2} P_1 \sim Q'_1 \vdash \Theta_1 \text{ (10)} \\ \theta_1 :: \Theta_1 \sqsubseteq \Theta \text{ (11)} \\ \Theta \mid \Delta_1, \Delta_2 \vdash \theta_1 Q'_1 \leq Q_1 \text{ (12)} \\ \theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \text{ (13)} \end{array}$$

By ⑬, the right-component premises ②, ⑦, and ⑨ can be rewritten with θ_1 . IH on the right component with $P_2 = \lfloor S_2 \rfloor$ gives

$$\Theta_1 \vdash_{\Delta_1|\Delta_2} P_2 \sim Q'_2 \dashv \Theta_2 \quad (14)$$

$$\theta_2 :: \Theta_2 \sqsubseteq \Theta \quad (15)$$

$$\Theta \mid \Delta_1, \Delta_2 \vdash \theta_2 Q'_2 \leq Q_2 \quad (16)$$

$$\theta_1 \equiv \theta_2 :: \Theta_1 \sqsubseteq \Theta \quad (17)$$

By U-ARROW on ⑩ and ⑭, we have

$$\Theta_0 \vdash_{\Delta_1|\Delta_2} P_1 \rightarrow P_2 \sim Q'_1 \rightarrow Q'_2 \dashv \Theta_2 \quad (18).$$

By ⑫, ⑮, and ⑰,

$$\Theta \mid \Delta_1, \Delta_2 \vdash \theta_2(Q'_1 \rightarrow Q'_2) \leq Q_1 \rightarrow Q_2 \quad (19).$$

By ⑬ and ⑰,

$$\theta \equiv \theta_2 :: \Theta_0 \sqsubseteq \Theta \quad (20).$$

Our goal follows from ⑮, ⑮, ⑳, and ⑲.

Case

C-FORALLINH

$$\Psi \mid \Theta, \alpha \vdash \theta S' \sim Q \quad (1)$$

$$\Psi, \delta \mid \Theta \vdash \theta(\forall^\delta \alpha. S') \sim \forall \alpha. Q$$

In this case $S = \forall^\delta \alpha. S'$ and $P = \forall \alpha. P'$ where $P' = \lfloor S' \rfloor$. The solution gives

$$\theta :: \Theta_0 \sqsubseteq \Theta \quad (2)$$

$$\Theta \mid \Delta_1 \vdash \theta P_0 \blacktriangleright \forall \alpha. Q \quad (3)$$

$$\Theta \mid \Delta_2 \vdash \theta(\forall \alpha. P') \blacktriangleright \forall \alpha. Q \quad (4).$$

Since P_0 cannot be a ghost or flexible variable, inversion on ③ gives $P_0 = \forall \alpha. P'_0$ and

$$\Theta, \alpha \mid \Delta_1, \alpha \vdash \theta P'_0 \blacktriangleright Q \quad (5).$$

By inversion on ④, we have

$$\Theta, \alpha \mid \Delta_2, \alpha \vdash \theta P' \blacktriangleright Q \quad (6).$$

By definition of metasubstitution, ② gives

$$\theta :: \Theta_0, \alpha \sqsubseteq \Theta, \alpha \quad (7).$$

IH on ①, ⑤, ⑥, and ⑦ gives

$$\Theta_0, \alpha \vdash_{\Delta_1, \alpha|\Delta_2, \alpha} P' \sim Q' \dashv \Theta_1, \alpha \quad (8)$$

$$\theta_1 :: \Theta_1, \alpha \sqsubseteq \Theta, \alpha \quad (9)$$

$$\Theta, \alpha \mid \Delta_1, \Delta_2, \alpha \vdash \theta_1 Q' \leq Q \quad (10)$$

$$\theta \equiv \theta_1 :: \Theta_0, \alpha \sqsubseteq \Theta, \alpha \quad (11)$$

By U-FORALLINH on ⑧ and $P_0 = \forall\alpha.P'_0$, we have

$$\Theta_0 [P_0]_{\Delta_1|\Delta_2} \forall\alpha.P' \sim \forall\alpha.Q' \dashv \Theta_1 \text{ ⑫}.$$

By removing the type variable α from ⑨ and ⑪, we have

$$\theta_1 :: \Theta_1 \sqsubseteq \Theta \text{ ⑬} \quad \theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \text{ ⑭}.$$

By ⑩, we have

$$\Theta \mid \Delta_1, \Delta_2 \vdash \theta_1(\forall\alpha.Q') \leq \forall\alpha.Q \text{ ⑮}.$$

Our goal follows from ⑫–⑮.

Case

C-FORALLSYN

$$\Psi \mid \Theta, \alpha \vdash \theta S' \sim Q \text{ ①}$$

$$\Psi \mid \Theta \vdash \theta(\forall^\delta\alpha.S') \sim \forall\alpha.Q$$

In this case $S = \forall^\delta\alpha.S'$ and $P = \forall\alpha.P'$ where $P' = \lfloor S' \rfloor$. The solution gives

$$\theta :: \Theta_0 \sqsubseteq \Theta \text{ ②} \quad \Theta \mid \Delta_1 \vdash \theta P_0 \blacktriangleright \forall\alpha.Q \text{ ③} \quad \Theta \mid \Delta_2 \vdash \theta(\forall\alpha.P') \blacktriangleright \forall\alpha.Q \text{ ④}.$$

Since P_0 cannot be a ghost or flexible variable, inversion on ③ gives $P_0 = \mathbb{A}.P'_0$ and

$$\Theta, \alpha \mid \Delta_1 \vdash \theta(\mathbb{A}.P'_0) \blacktriangleright Q \text{ ⑤}.$$

By inversion on ④, we have

$$\Theta, \alpha \mid \Delta_2, \alpha \vdash \theta P' \blacktriangleright Q \text{ ⑥}.$$

By definition of metasubstitution, ② gives

$$\theta :: \Theta_0, \alpha \sqsubseteq \Theta, \alpha \text{ ⑦}.$$

IH on ①, ⑤, ⑥, and ⑦ gives

$$\Theta_0, \alpha [\mathbb{A}.P'_0]_{\Delta_1|\Delta_2, \alpha} P' \sim Q' \dashv \Theta_1, \alpha \text{ ⑧}$$

$$\theta_1 :: \Theta_1, \alpha \sqsubseteq \Theta, \alpha \text{ ⑨}$$

$$\Theta, \alpha \mid \Delta_1, \Delta_2, \alpha \vdash \theta_1 Q' \leq Q \text{ ⑩}$$

$$\theta \equiv \theta_1 :: \Theta_0, \alpha \sqsubseteq \Theta, \alpha \text{ ⑪}$$

By U-FORALLSYN on ⑧ and $P_0 = \mathbb{A}.P'_0$, we have

$$\Theta_0 [P_0]_{\Delta_1|\Delta_2} \forall\alpha.P' \sim \forall\alpha.Q' \dashv \Theta_1 \text{ ⑫}.$$

By removing the type variable α from ⑨ and ⑪, we have

$$\theta_1 :: \Theta_1 \sqsubseteq \Theta \text{ ⑬} \quad \theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \text{ ⑭}.$$

By ⑩, we have

$$\Theta \mid \Delta_1, \Delta_2 \vdash \theta_1(\forall\alpha.Q') \leq \forall\alpha.Q \text{ ⑮}.$$

Our goal follows from ⑫–⑮.

Cases

$$\begin{array}{c}
\text{C-UNIVGHOST} \\
\frac{\Psi \mid \Theta \vdash \theta S' \sim Q \text{ ①}}{\Psi \mid \Theta \vdash \mathbb{A}. \theta S' \sim \mathbb{A}. Q}
\end{array}
\quad
\begin{array}{c}
\text{C-UNIVGHOSTG} \\
\frac{\Psi \mid \Theta \vdash \theta S' \sim Q \text{ ①}}{\Psi \mid \Theta \vdash \mathbb{A}. \theta S' \sim \mathbb{A}. Q}
\end{array}$$

In these cases $S = \mathbb{A}.S'$ or $S = \mathbb{A}.S'$, respectively, and $P = \mathbb{A}.P'$ where $P' = \lfloor S' \rfloor$. The solution gives

$$\theta :: \Theta_0 \sqsubseteq \Theta \text{ ②} \quad \Theta \mid \Delta_1 \vdash \theta P_0 \blacktriangleright \mathbb{A}.Q \text{ ③} \quad \Theta \mid \Delta_2 \vdash \theta(\mathbb{A}.P') \blacktriangleright \mathbb{A}.Q \text{ ④}.$$

Since P_0 cannot be a ghost or flexible variable, inversion on ③ gives $P_0 = \mathbb{A}.P'_0$ and

$$\Theta \mid \Delta_1 \vdash \theta P'_0 \blacktriangleright Q \text{ ⑤}.$$

By inversion on ④, we have

$$\Theta \mid \Delta_2 \vdash \theta P' \blacktriangleright Q \text{ ⑥}.$$

IH on ①, ⑤, ⑥, and ② gives

$$\begin{array}{l}
\Theta_0 \mid P'_0 \vdash_{\Delta_1 \mid \Delta_2} P' \sim Q' \vdash \Theta_1 \text{ ⑦} \\
\theta_1 :: \Theta_1 \sqsubseteq \Theta \text{ ⑧} \\
\Theta \mid \Delta_1, \Delta_2 \vdash \theta_1 Q' \leq Q \text{ ⑨} \\
\theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \text{ ⑩}
\end{array}$$

By U-UNIVGHOST on ⑦ and $P_0 = \mathbb{A}.P'_0$, we have

$$\Theta_0 \mid P_0 \vdash_{\Delta_1 \mid \Delta_2} \mathbb{A}.P' \sim \mathbb{A}.Q' \vdash \Theta_1 \text{ ⑪}.$$

By ⑨, we have

$$\Theta \mid \Delta_1, \Delta_2 \vdash \theta_1(\mathbb{A}.Q') \leq \mathbb{A}.Q \text{ ⑫}.$$

Our goal follows from ⑪, ⑧, ⑫, and ⑩.

Case

$$\begin{array}{c}
\text{C-UNIVGHOST1} \\
\frac{\Psi \mid \Theta, \alpha \vdash \mathbb{A}. \theta S' \sim Q \text{ ①}}{\Psi \mid \Theta \vdash \mathbb{A}. \theta S' \sim \forall \alpha. Q}
\end{array}$$

In this case $S = \mathbb{A}.S'$ and $P = \mathbb{A}.P'$ where $P' = \lfloor S' \rfloor$. The solution gives

$$\theta :: \Theta_0 \sqsubseteq \Theta \text{ ②} \quad \Theta \mid \Delta_1 \vdash \theta P_0 \blacktriangleright \forall \alpha. Q \text{ ③} \quad \Theta \mid \Delta_2 \vdash \theta(\mathbb{A}.P') \blacktriangleright \forall \alpha. Q \text{ ④}.$$

Since P_0 cannot be a ghost or flexible variable, inversion on ③ gives $P_0 = \forall \alpha. P'_0$ and

$$\Theta, \alpha \mid \Delta_1, \alpha \vdash \theta P'_0 \blacktriangleright Q \text{ ⑤}.$$

By inversion on ④, we have

$$\Theta, \alpha \mid \Delta_2 \vdash \theta(\mathbb{A}.P') \blacktriangleright Q \text{ ⑥}.$$

By definition of metasubstitution, ② gives

$$\theta :: \Theta_0, \alpha \sqsubseteq \Theta, \alpha \text{ ⑦.}$$

IH on ①, ⑤, ⑥, and ⑦ gives

$$\Theta_0, \alpha [P'_0]_{\Delta_1, \alpha | \Delta_2} \mathbb{R}.P' \sim Q' \dashv \Theta_1, \alpha \text{ ⑧}$$

$$\theta_1 :: \Theta_1, \alpha \sqsubseteq \Theta, \alpha \text{ ⑨}$$

$$\Theta, \alpha \mid \Delta_1, \Delta_2, \alpha \vdash \theta_1 Q' \leq Q \text{ ⑩}$$

$$\theta \equiv \theta_1 :: \Theta_0, \alpha \sqsubseteq \Theta, \alpha \text{ ⑪}$$

By U-UNIVGHOST1 on ⑧ and $P_0 = \forall \alpha. P'_0$, we have

$$\Theta_0 [P_0]_{\Delta_1 | \Delta_2} \mathbb{R}.P' \sim \forall \alpha. Q' \dashv \Theta_1 \text{ ⑫.}$$

By removing the type variable α from ⑨ and ⑪, we have

$$\theta_1 :: \Theta_1 \sqsubseteq \Theta \text{ ⑬} \quad \theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \text{ ⑭.}$$

By ⑩ and the precision rule for inherited quantifiers, we have

$$\Theta \mid \Delta_1, \Delta_2 \vdash \theta_1 (\forall \alpha. Q') \leq \forall \alpha. Q \text{ ⑮.}$$

Our goal follows from ⑫–⑮.

Case

C-UNIVGHOST2

$$\Psi \mid \Theta \vdash \theta S' \sim Q \text{ ①}$$

$$\Psi \mid \Theta \vdash \mathbb{R}.\theta S' \sim Q$$

In this case $S = \mathbb{R}.S'$ and $P = \mathbb{R}.P'$ where $P' = [S']$. The solution gives

$$\theta :: \Theta_0 \sqsubseteq \Theta \text{ ②} \quad \Theta \mid \Delta_1 \vdash \theta P_0 \blacktriangleright Q \text{ ③} \quad \Theta \mid \Delta_2 \vdash \theta (\mathbb{R}.P') \blacktriangleright Q \text{ ④.}$$

By well-formedness of $\mathbb{R}.P'$ and inversion on ①, Q is guarded.

We proceed by case analysis on P_0 .

Case $P_0 = \mathbb{R}.P'_0$. Inversion on ③ gives

$$\Theta \mid \Delta_1 \vdash \theta P'_0 \blacktriangleright Q \text{ ⑤.}$$

By guardedness of Q , inversion on ④ gives

$$\Theta \mid \Delta_2 \vdash \theta P' \blacktriangleright Q \text{ ⑥.}$$

IH on ①, ⑤, ⑥, and ② gives

$$\Theta_0 [P'_0]_{\Delta_1 | \Delta_2} P' \sim Q' \dashv \Theta_1 \text{ ⑦}$$

$$\theta_1 :: \Theta_1 \sqsubseteq \Theta \text{ ⑧}$$

$$\Theta \mid \Delta_1, \Delta_2 \vdash \theta_1 Q' \leq Q \text{ ⑨}$$

$$\theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \text{ ⑩}$$

By U-UNIVGHOST on ⑦, we have

$$\Theta_0 [\mathbb{A}.P'_0] \vdash_{\Delta_1|\Delta_2} \mathbb{A}.P' \sim \mathbb{A}.Q' \dashv \Theta_1 \text{ ⑪}.$$

By ⑨ and the precision rule for universal ghosts, we have

$$\Theta \mid \Delta_1, \Delta_2 \vdash \theta_1(\mathbb{A}.Q') \leq Q \text{ ⑫}.$$

Our goal follows from ⑪, ⑧, ⑩, and ⑫.

Case P_0 is guarded. By guardedness of Q , inversion on ④ gives

$$\Theta \mid \Delta_2 \vdash \theta P' \blacktriangleright Q \text{ ⑤}.$$

IH on ①, ③, ⑤, and ② gives

$$\Theta_0 [P_0] \vdash_{\Delta_1|\Delta_2} P' \sim Q' \dashv \Theta_1 \text{ ⑥}$$

$$\theta_1 :: \Theta_1 \sqsubseteq \Theta \text{ ⑦}$$

$$\Theta \mid \Delta_1, \Delta_2 \vdash \theta_1 Q' \leq Q \text{ ⑧}$$

$$\theta \equiv \theta_1 :: \Theta_0 \sqsubseteq \Theta \text{ ⑨}$$

By guardedness of Q , since P_0 cannot be a ghost or a flexible variable, we have

$$\Theta_0 \vdash P_0 \text{ guarded ⑩} \quad P_0 \text{ non-flex ⑪}.$$

By U-UNIVGHOST2 on ⑩, ⑪, and ⑥, we have

$$\Theta_0 [P_0] \vdash_{\Delta_1|\Delta_2} \mathbb{A}.P' \sim Q' \dashv \Theta_1 \text{ ⑫}.$$

Our goal follows from ⑫, ⑦, ⑧, and ⑨.

□

Bibliography

- Andreas Abel and Jean-Philippe Bernardy. A unified view of modalities in type systems. *Proc. ACM Program. Lang.*, 4(ICFP), aug 2020. doi: 10.1145/3408972. URL <https://doi.org/10.1145/3408972>.
- Danel Ahman. When programs have to watch paint dry. In Orna Kupferman and Pawel Sobocinski, editors, *Foundations of Software Science and Computation Structures - 26th International Conference, FoSSaCS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, volume 13992 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2023. doi: 10.1007/978-3-031-30829-1_1. URL https://doi.org/10.1007/978-3-031-30829-1_1.
- Hendrik Pieter Barendregt. *The lambda calculus - its syntax and semantics*, volume 103 of *Studies in logic and the foundations of mathematics*. North-Holland, 1985. ISBN 978-0-444-86748-3.
- Andrej Bauer and Matija Pretnar. An effect system for algebraic effects and handlers. *Log. Methods Comput. Sci.*, 10(4), 2014. doi: 10.2168/LMCS-10(4:9)2014. URL [https://doi.org/10.2168/LMCS-10\(4:9\)2014](https://doi.org/10.2168/LMCS-10(4:9)2014).
- Dariusz Biernacki, Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. Handle with care: relational interpretation of algebraic effects and handlers. *Proc. ACM Program. Lang.*, 2(POPL):8:1–8:30, 2018. doi: 10.1145/3158096. URL <https://doi.org/10.1145/3158096>.
- Dariusz Biernacki, Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. Abstracting algebraic effects. *Proc. ACM Program. Lang.*, 3(POPL):6:1–6:28, 2019. doi: 10.1145/3290319. URL <https://doi.org/10.1145/3290319>.
- Dariusz Biernacki, Maciej Piróg, Piotr Polesiuk, and Filip Sieczkowski. Binders by day, labels by night: effect instances via lexically scoped handlers. *Proc. ACM Program. Lang.*, 4(POPL):48:1–48:29, 2020. doi: 10.1145/3371116. URL <https://doi.org/10.1145/3371116>.
- Aleksander Boruch-Gruszecki, Martin Odersky, Edward Lee, Ondrej Lhoták, and Jonathan Immanuel Brachthäuser. Capturing types. *ACM Trans. Program. Lang. Syst.*, 45(4):21:1–21:52, 2023. doi: 10.1145/3618003. URL <https://doi.org/10.1145/3618003>.

- Roger Bosman, Birthe van den Berg, Wenhao Tang, and Tom Schrijvers. A calculus for scoped effects & handlers. *Log. Methods Comput. Sci.*, 20(4), 2024. doi: 10.46298/LMCS-20(4:17)2024. URL [https://doi.org/10.46298/lmcs-20\(4:17\)2024](https://doi.org/10.46298/lmcs-20(4:17)2024).
- Aghilas Y. Boussaa. A language with modal effect types and Fresco type inference, 2026. URL <https://github.com/4y8/modal-effect-types/tree/frost>. Branch: frost.
- Aghilas Y. Boussaa, Wenhao Tang, and Sam Lindley. Higher-order fork, modally, August 2026. URL <https://icfp26.sigplan.org/details/hope-2026-papers/2/Higher-order-fork-modally>. Talk presented at the 14th ACM SIGPLAN Workshop on Higher-Order Programming with Effects (HOPE 2026), Indianapolis, Indiana, USA.
- Jonathan Immanuel Brachthäuser, Philipp Schuster, and Klaus Ostermann. Effects as capabilities: effect handlers and lightweight effect polymorphism. *Proc. ACM Program. Lang.*, 4(OOPSLA):126:1–126:30, 2020. doi: 10.1145/3428194. URL <https://doi.org/10.1145/3428194>.
- Jonathan Immanuel Brachthäuser, Philipp Schuster, Edward Lee, and Aleksander Boruch-Gruszecki. Effects, capabilities, and boxes: from scope-based reasoning to type-based reasoning and back. *Proc. ACM Program. Lang.*, 6(OOPSLA1):1–30, 2022. doi: 10.1145/3527320. URL <https://doi.org/10.1145/3527320>.
- Jonathan Immanuel Brachthäuser, Philipp Schuster, and Klaus Ostermann. Effekt language: A language with lexical effect handlers and lightweight effect polymorphism, 2026. URL <https://effekt-lang.org>.
- Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In Jan Van den Bussche and Victor Vianu, editors, *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings*, volume 1973 of *Lecture Notes in Computer Science*, pages 316–330. Springer, 2001. doi: 10.1007/3-540-44503-X_20. URL https://doi.org/10.1007/3-540-44503-X_20.
- Albert Camus. *Le Mythe de Sisyphe*. Gallimard, Paris, France, 1942.
- Vikraman Choudhury and Neel Krishnaswami. Recovering purity with comonads and capabilities. *Proc. ACM Program. Lang.*, 4(ICFP):111:1–111:28, 2020. doi: 10.1145/3408993. URL <https://doi.org/10.1145/3408993>.
- Jacek Chrząszcz. Polymorphic subtyping without distributivity. In Luboš Brim, Jozef Gruska, and Jiří Zlatuška, editors, *Mathematical Foundations of Computer Science 1998*,

pages 346–355, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. ISBN 978-3-540-68532-6.

Ranald Clouston. Fitch-style modal lambda calculi. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10803 of *Lecture Notes in Computer Science*, pages 258–275. Springer, 2018. doi: 10.1007/978-3-319-89366-2_14. URL https://doi.org/10.1007/978-3-319-89366-2_14.

Lukas Convent, Sam Lindley, Conor McBride, and Craig McLaughlin. Doo bee doo bee doo. *J. Funct. Program.*, 30:e9, 2020. doi: 10.1017/S0956796820000039. URL <https://doi.org/10.1017/S0956796820000039>.

Chen Cui, Shengyi Jiang, and Bruno C. d. S. Oliveira. Greedy implicit bounded quantification. *Proc. ACM Program. Lang.*, 7(OOPSLA2):2083–2111, 2023. doi: 10.1145/3622871. URL <https://doi.org/10.1145/3622871>.

Luis Damas and Robin Milner. Principal type-schemes for functional programs. In *Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '82*, page 207–212, New York, NY, USA, 1982. Association for Computing Machinery. ISBN 0897910656. doi: 10.1145/582153.582176. URL <https://doi.org/10.1145/582153.582176>.

Rowan Davies and Frank Pfenning. A modal analysis of staged computation. *J. ACM*, 48(3):555–604, may 2001. ISSN 0004-5411. doi: 10.1145/382780.382785. URL <https://doi.org/10.1145/382780.382785>.

Paulo Emílio de Vilhena and François Pottier. A type system for effect handlers and dynamic labels. In Thomas Wies, editor, *Programming Languages and Systems - 32nd European Symposium on Programming, ESOP 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, volume 13990 of *Lecture Notes in Computer Science*, pages 225–252. Springer, 2023. doi: 10.1007/978-3-031-30044-8_9. URL https://doi.org/10.1007/978-3-031-30044-8_9.

Jack B. Dennis and Earl C. Van Horn. Programming semantics for multiprogrammed computations. *Commun. ACM*, 9(3):143–155, 1966. doi: 10.1145/365230.365252. URL <https://doi.org/10.1145/365230.365252>.

- Stephen Dolan. *Algebraic Subtyping*. PhD thesis, Computer Laboratory, University of Cambridge, United Kingdom, 9 2016.
- Jana Dunfield and Neel Krishnaswami. Bidirectional typing. *ACM Comput. Surv.*, 54(5):98:1–98:38, 2022. doi: 10.1145/3450952. URL <https://doi.org/10.1145/3450952>.
- Jana Dunfield and Neelakantan R. Krishnaswami. Complete and easy bidirectional typechecking for higher-rank polymorphism. In Greg Morrisett and Tarmo Uustalu, editors, *ACM SIGPLAN International Conference on Functional Programming, ICFP’13, Boston, MA, USA - September 25 - 27, 2013*, pages 429–442. ACM, 2013. doi: 10.1145/2500365.2500582. URL <https://doi.org/10.1145/2500365.2500582>.
- Jana Dunfield and Frank Pfenning. Tridirectional typechecking. In Neil D. Jones and Xavier Leroy, editors, *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004*, pages 281–292. ACM, 2004. doi: 10.1145/964001.964025. URL <https://doi.org/10.1145/964001.964025>.
- Frank Emrich, Sam Lindley, Jan Stolarek, James Cheney, and Jonathan Coates. FreezeML: complete and easy type inference for first-class polymorphism. In Alastair F. Donaldson and Emina Torlak, editors, *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, pages 423–437. ACM, 2020. doi: 10.1145/3385412.3386003. URL <https://doi.org/10.1145/3385412.3386003>.
- Frank Emrich, Jan Stolarek, James Cheney, and Sam Lindley. Constraint-based type inference for FreezeML. *Proc. ACM Program. Lang.*, 6(ICFP):570–595, 2022. doi: 10.1145/3547642. URL <https://doi.org/10.1145/3547642>.
- Matthias Felleisen. On the expressive power of programming languages. *Sci. Comput. Program.*, 17(1-3):35–75, 1991. doi: 10.1016/0167-6423(91)90036-W. URL [https://doi.org/10.1016/0167-6423\(91\)90036-W](https://doi.org/10.1016/0167-6423(91)90036-W).
- Jacques Garrigue and Didier Rémy. Semi-explicit first-class polymorphism for ML. *Inf. Comput.*, 155(1-2):134–169, 1999. doi: 10.1006/INCO.1999.2830. URL <https://doi.org/10.1006/inco.1999.2830>.
- Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987. doi: 10.1016/0304-3975(87)90045-4. URL [https://doi.org/10.1016/0304-3975\(87\)90045-4](https://doi.org/10.1016/0304-3975(87)90045-4).
- Daniel Gratzer. *Syntax and semantics of modal type theory*. PhD thesis, Aarhus University, 2023.

- Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. Multimodal dependent type theory. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 492–506. ACM, 2020. doi: 10.1145/3373718.3394736. URL <https://doi.org/10.1145/3373718.3394736>.
- Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. Multimodal dependent type theory. *Log. Methods Comput. Sci.*, 17(3), 2021. doi: 10.46298/LMCS-17(3:11)2021. URL [https://doi.org/10.46298/lmcs-17\(3:11\)2021](https://doi.org/10.46298/lmcs-17(3:11)2021).
- Daniel Gratzer, Evan Cavallo, G. A. Kavvos, Adrien Guatto, and Lars Birkedal. Modalities and parametric adjoints. *ACM Trans. Comput. Log.*, 23(3):18:1–18:29, 2022. doi: 10.1145/3514241. URL <https://doi.org/10.1145/3514241>.
- Adam Gundry, Conor McBride, and James McKinna. Type inference in context. In *MSFP@ICFP*, pages 43–54. ACM, 2010.
- Adam Michael Gundry. *Type inference, Haskell and dependent types*. PhD thesis, University of Strathclyde, Glasgow, UK, 2013. URL http://oleg.lib.strath.ac.uk/R/?func=dbin-jump-full&object_id=22728.
- Daniel Hillerström and Sam Lindley. Liberating effects with rows and handlers. In *TyDe*, pages 15–27. ACM, 2016.
- Daniel Hillerström and Sam Lindley. Shallow effect handlers. In Sukyoung Ryu, editor, *Programming Languages and Systems - 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings*, volume 11275 of *Lecture Notes in Computer Science*, pages 415–435. Springer, 2018. doi: 10.1007/978-3-030-02768-1_22. URL https://doi.org/10.1007/978-3-030-02768-1_22.
- Daniel Hillerström, Sam Lindley, and Robert Atkey. Effect handlers via generalised continuations. *J. Funct. Program.*, 30:e5, 2020. doi: 10.1017/S0956796820000040. URL <https://doi.org/10.1017/S0956796820000040>.
- Daniel Hillerström. *Foundations for Programming and Implementing Effect Handlers*. PhD thesis, The University of Edinburgh, UK, 2022. URL <http://dx.doi.org/10.7488/era/2122>.
- Alex Hubers and J. Garrett Morris. Generic programming with extensible data types: Or, making ad hoc extensible data types less ad hoc. *Proc. ACM Program. Lang.*, 7(ICFP):356–384, 2023. doi: 10.1145/3607843. URL <https://doi.org/10.1145/3607843>.

- IFIP Working Group 2.8. IFIP WG2.8 meeting 44, 2026. URL <https://ifip-wg28.github.io/44-viana-do-castelo-2026/>. Viana do Castelo, Portugal, 2–6 March 2026.
- Shengyi Jiang, Chen Cui, and Bruno C. d. S. Oliveira. Bidirectional higher-rank polymorphism with intersection and union types. *Proc. ACM Program. Lang.*, 9(POPL): 2118–2148, 2025. doi: 10.1145/3704907. URL <https://doi.org/10.1145/3704907>.
- Mark P. Jones. A theory of qualified types. *Sci. Comput. Program.*, 22(3):231–256, 1994. doi: 10.1016/0167-6423(94)00005-0. URL [https://doi.org/10.1016/0167-6423\(94\)00005-0](https://doi.org/10.1016/0167-6423(94)00005-0).
- Robin Jourde, Daniel Hillerström, and Sam Lindley. Generative labels, 2022. URL <https://github.com/links-lang/links/pull/1169>.
- Ohad Kammar, Sam Lindley, and Nicolas Oury. Handlers in action. In Greg Morrisett and Tarmo Uustalu, editors, *ACM SIGPLAN International Conference on Functional Programming, ICFP’13, Boston, MA, USA - September 25 - 27, 2013*, pages 145–158. ACM, 2013. doi: 10.1145/2500365.2500590. URL <https://doi.org/10.1145/2500365.2500590>.
- Georgios Karachalias, Matija Pretnar, Amr Hany Saleh, Stien Vanderhallen, and Tom Schrijvers. Explicit effect subtyping. *J. Funct. Program.*, 30:e15, 2020. doi: 10.1017/S0956796820000131. URL <https://doi.org/10.1017/S0956796820000131>.
- G. A. Kavvos and Daniel Gratzer. Under lock and key: a proof system for a multimodal logic. *Bull. Symb. Log.*, 29(2):264–293, 2023. doi: 10.1017/BSL.2023.14. URL <https://doi.org/10.1017/bsl.2023.14>.
- András Kovács. Elaboration with first-class implicit function types. *Proc. ACM Program. Lang.*, 4(ICFP):101:1–101:29, 2020. doi: 10.1145/3408983. URL <https://doi.org/10.1145/3408983>.
- András Kovács. Elaboration zoo, 2026. URL <https://github.com/AndrasKovacs/elaboration-zoo/>. First-class polymorphism with dynamic order elaboration.
- Peter J. Landin. The next 700 programming languages. *Commun. ACM*, 9(3):157–166, 1966. doi: 10.1145/365230.365257. URL <https://doi.org/10.1145/365230.365257>.
- John Launchbury and Simon L. Peyton Jones. State in Haskell. *LISP Symb. Comput.*, 8(4):293–341, 1995.
- Didier Le Botlan and Didier Rémy. ML^F : raising ML to the power of system F. In Colin Runciman and Olin Shivers, editors, *Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming, ICFP 2003, Uppsala, Sweden, August*

- 25-29, 2003, pages 27–38. ACM, 2003. doi: 10.1145/944705.944709. URL <https://doi.org/10.1145/944705.944709>.
- Daan Leijen. Extensible records with scoped labels. In Marko C. J. D. van Eekelen, editor, *Revised Selected Papers from the Sixth Symposium on Trends in Functional Programming, TFP 2005, Tallinn, Estonia, 23-24 September 2005*, volume 6 of *Trends in Functional Programming*, pages 179–194. Intellect, 2005.
- Daan Leijen. HMF: simple type inference for first-class polymorphism. In James Hook and Peter Thiemann, editors, *Proceeding of the 13th ACM SIGPLAN international conference on Functional programming, ICFP 2008, Victoria, BC, Canada, September 20-28, 2008*, pages 283–294. ACM, 2008. doi: 10.1145/1411204.1411245. URL <https://doi.org/10.1145/1411204.1411245>.
- Daan Leijen. Flexible types: robust type inference for first-class polymorphism. In Zhong Shao and Benjamin C. Pierce, editors, *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, pages 66–77. ACM, 2009. doi: 10.1145/1480881.1480891. URL <https://doi.org/10.1145/1480881.1480891>.
- Daan Leijen. Type directed compilation of row-typed algebraic effects. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL '17*, page 486–499, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346603. doi: 10.1145/3009837.3009872. URL <https://doi.org/10.1145/3009837.3009872>.
- Daan Leijen. Koka: A strongly typed functional-style language with effect types and handlers, 2026. URL <https://koka-lang.github.io>.
- Daan Leijen and Wenjia Ye. Principal type inference under a prefix: A fresh look at static overloading. *Proc. ACM Program. Lang.*, 9(PLDI), June 2025. doi: 10.1145/3729308. URL <https://doi.org/10.1145/3729308>.
- Paul Blain Levy. *Call-By-Push-Value*. PhD thesis, Queen Mary and Westfield College, University of London, 2001. URL <https://pblevy.github.io/papers/thesisqmwphd.pdf>.
- Paul Blain Levy. *Call-By-Push-Value: A Functional/Imperative Synthesis*, volume 2 of *Semantics Structures in Computation*. Springer, 2004. ISBN 1-4020-1730-8.

- Paul Blain Levy, John Power, and Hayo Thielecke. Modelling environments in call-by-value programming languages. *Inf. Comput.*, 185(2):182–210, 2003. doi: 10.1016/S0890-5401(03)00088-9. URL [https://doi.org/10.1016/S0890-5401\(03\)00088-9](https://doi.org/10.1016/S0890-5401(03)00088-9).
- Sam Lindley, Conor McBride, and Craig McLaughlin. Do be do be do. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017*, page 500–514, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346603. doi: 10.1145/3009837.3009897. URL <https://doi.org/10.1145/3009837.3009897>.
- Anton Lorenzen, Leo White, Stephen Dolan, Richard A. Eisenberg, and Sam Lindley. Oxidizing OCaml with modal memory management. *Proc. ACM Program. Lang.*, 8(ICFP):485–514, 2024. doi: 10.1145/3674642. URL <https://doi.org/10.1145/3674642>.
- Anton Lorenzen, Wenhao Tang, and Sam Lindley. A value trick for modal type systems, July 2026. URL <https://antonlorenzen.de/papers/value-trick-impla.pdf>. Extended abstract presented at the 8th Intuitionistic Modal Logic and Applications Workshop (IMLA 2026), Lisbon, Portugal.
- John M. Lucassen and David K. Gifford. Polymorphic effect systems. In Jeanne Ferrante and Peter Mager, editors, *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages, San Diego, California, USA, January 10-13, 1988*, pages 47–57. ACM Press, 1988. doi: 10.1145/73560.73564. URL <https://doi.org/10.1145/73560.73564>.
- Conor McBride. Basics of bidirectionality, 2018. URL <https://pigworker.wordpress.com/2018/08/06/basics-of-bidirectionality/>.
- Mark S. Miller. *Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control*. PhD thesis, Johns Hopkins University, Baltimore, Maryland, USA, 2006. AAI3245526.
- Robin Milner. A theory of type polymorphism in programming. *J. Comput. Syst. Sci.*, 17(3):348–375, 1978. doi: 10.1016/0022-0000(78)90014-4. URL [https://doi.org/10.1016/0022-0000\(78\)90014-4](https://doi.org/10.1016/0022-0000(78)90014-4).
- Robin Milner, Mads Tofte, and Robert Harper. *Definition of standard ML*. MIT Press, 1990. ISBN 978-0-262-63132-7.
- Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991. doi: 10.1016/0890-5401(91)90052-4. URL [https://doi.org/10.1016/0890-5401\(91\)90052-4](https://doi.org/10.1016/0890-5401(91)90052-4).

- J. Garrett Morris. Another type inference algorithm for first-class implicit polymorphism. *Proc. ACM Program. Lang.*, 10(ICFP):304:1–304:29, 2026. doi: 10.1145/3828702. URL <https://doi.org/10.1145/3828702>.
- J. Garrett Morris and James McKinna. Abstracting extensible data types: Or, rows by any other name. *Proc. ACM Program. Lang.*, 3(POPL), jan 2019. doi: 10.1145/3290325. URL <https://doi.org/10.1145/3290325>.
- Aleksandar Nanevski. *Functional programming with names and necessity*. PhD thesis, Carnegie Mellon University, USA, 2004. AAI3143944.
- Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. Contextual modal type theory. *ACM Trans. Comput. Log.*, 9(3):23:1–23:49, 2008. doi: 10.1145/1352582.1352591. URL <https://doi.org/10.1145/1352582.1352591>.
- Alistair O’Brien, Didier Rémy, and Gabriel Scherer. Omnidirectional type inference for ML: principality any way, 2025. URL <https://arxiv.org/abs/2511.10343>.
- Martin Odersky, Christoph Zenger, and Matthias Zenger. Colored local type inference. In Chris Hankin and Dave Schmidt, editors, *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, pages 41–53. ACM, 2001. doi: 10.1145/360204.360207. URL <https://doi.org/10.1145/360204.360207>.
- Dominic Orchard, Vilem-Benjamin Liepelt, and Harley Eades III. Quantitative program reasoning with graded modal types. *Proc. ACM Program. Lang.*, 3(ICFP), jul 2019. doi: 10.1145/3341714. URL <https://doi.org/10.1145/3341714>.
- Lionel Parreaux. Decoupling resolution from type inference. In *Proceedings of the Workshop on the Implementation of Type Systems (WITS) at the 2026 ACM SIGPLAN International Conference on Principles of Programming Languages (POPL 2026)*, 2026. URL <https://popl26.sigplan.org/details/wits-2026-papers/6/Decoupling-Resolution-from-Type-Inference>. Presented at WITS, Rennes, France, January 17 2026.
- Lionel Parreaux, Aleksander Boruch-Gruszecki, Andong Fan, and Chun Yin Chau. When subtyping constraints liberate: A novel type inference approach for first-class polymorphism. *Proc. ACM Program. Lang.*, 8(POPL):1418–1450, 2024. doi: 10.1145/3632890. URL <https://doi.org/10.1145/3632890>.

- Simon L. Peyton Jones, Dimitrios Vytiniotis, Stephanie Weirich, and Mark Shields. Practical type inference for arbitrary-rank types. *J. Funct. Program.*, 17(1):1–82, 2007. doi: 10.1017/S0956796806006034. URL <https://doi.org/10.1017/S0956796806006034>.
- Frank Pfenning. On the undecidability of partial polymorphic type reconstruction. *Fundam. Informaticae*, 19(1/2):185–199, 1993.
- Benjamin C. Pierce and David N. Turner. Local type inference. *ACM Trans. Program. Lang. Syst.*, 22(1):1–44, 2000. doi: 10.1145/345099.345100. URL <https://doi.org/10.1145/345099.345100>.
- Maciej Piróg, Tom Schrijvers, Nicolas Wu, and Mauro Jaskelioff. Syntax and semantics for operations with scopes. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 809–818. ACM, 2018. doi: 10.1145/3209108.3209166. URL <https://doi.org/10.1145/3209108.3209166>.
- G.D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theoretical Computer Science*, 1(2):125–159, 1975. ISSN 0304-3975. doi: 10.1016/0304-3975(75)90017-1. URL <https://www.sciencedirect.com/science/article/pii/0304397575900171>.
- Gordon D. Plotkin and John Power. Algebraic operations and generic effects. *Appl. Categorical Struct.*, 11(1):69–94, 2003. doi: 10.1023/A:1023064908962. URL <https://doi.org/10.1023/A:1023064908962>.
- Gordon D. Plotkin and Matija Pretnar. Handling algebraic effects. *Log. Methods Comput. Sci.*, 9(4), 2013. doi: 10.2168/LMCS-9(4:23)2013. URL [https://doi.org/10.2168/LMCS-9\(4:23\)2013](https://doi.org/10.2168/LMCS-9(4:23)2013).
- François Pottier and Yann Régis-Gianas. Stratified type inference for generalized algebraic data types. *SIGPLAN Not.*, 41(1):232–244, January 2006. ISSN 0362-1340. doi: 10.1145/1111320.1111058. URL <https://doi.org/10.1145/1111320.1111058>.
- Matija Pretnar. Inferring algebraic effects. *Log. Methods Comput. Sci.*, 10(3), 2014. doi: 10.2168/LMCS-10(3:21)2014. URL [https://doi.org/10.2168/LMCS-10\(3:21\)2014](https://doi.org/10.2168/LMCS-10(3:21)2014).
- Matija Pretnar. An introduction to algebraic effects and handlers. *Electron. Notes Theor. Comput. Sci.*, 319:19–35, 2015. doi: 10.1016/j.entcs.2015.12.003. URL <https://www.eff-lang.org/handlers-tutorial.pdf>.
- D. Rémy. Type checking records and variants in a natural extension of ML. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '89*, page 77–88, New York, NY, USA, 1989. Association

- for Computing Machinery. ISBN 0897912942. doi: 10.1145/75277.75284. URL <https://doi.org/10.1145/75277.75284>.
- Didier Rémy. Type inference for records in a natural extension of ML. In *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design*. MIT Press, 1994.
- Dennis Ritchie and Ken Thompson. The UNIX time-sharing system. *Commun. ACM*, 17 (7):365–375, 1974.
- Claudio V. Russo and Dimitrios Vytiniotis. QML: explicit first-class polymorphism for ML. In *Proceedings of the 2009 ACM SIGPLAN Workshop on ML, ML ’09*, page 3–14, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585093. doi: 10.1145/1596627.1596630. URL <https://doi.org/10.1145/1596627.1596630>.
- Alejandro Serrano, Jurriaan Hage, Dimitrios Vytiniotis, and Simon Peyton Jones. Guarded impredicative polymorphism. In Jeffrey S. Foster and Dan Grossman, editors, *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18–22, 2018*, pages 783–796. ACM, 2018. doi: 10.1145/3192366.3192389. URL <https://doi.org/10.1145/3192366.3192389>.
- Alejandro Serrano, Jurriaan Hage, Simon Peyton Jones, and Dimitrios Vytiniotis. A quick look at impredicativity. *Proc. ACM Program. Lang.*, 4(ICFP):89:1–89:29, 2020. doi: 10.1145/3408971. URL <https://doi.org/10.1145/3408971>.
- Michael Shulman. Brouwer’s fixed-point theorem in real-cohesive homotopy type theory. *Math. Struct. Comput. Sci.*, 28(6):856–941, 2018. doi: 10.1017/S0960129517000147. URL <https://doi.org/10.1017/S0960129517000147>.
- Michael Shulman. Semantics of multimodal adjoint type theory. In Marie Kerjean and Paul Blain Levy, editors, *Proceedings of the 39th Conference on the Mathematical Foundations of Programming Semantics, MFPS XXXIX, Indiana University, Bloomington, IN, USA, June 21–23, 2023*, volume 3 of ENTICS. EpiSciences, 2023. doi: 10.46298/ENTICS.12300. URL <https://doi.org/10.46298/entics.12300>.
- Wenhao Tang. Artifact for Modal Effect Types, 2025. URL <https://github.com/thwfhk/met-oopsla25-artifact>.
- Wenhao Tang and Sam Lindley. Rows and capabilities as modal effects. *Proc. ACM Program. Lang.*, 10(POPL):923–950, 2026. doi: 10.1145/3776674. URL <https://doi.org/10.1145/3776674>.

- Wenhao Tang and Tom Schrijvers. From high to low: Simulating nondeterminism and state with state. *J. Funct. Program.*, 34, 2024. doi: 10.1017/S0956796824000133. URL <https://doi.org/10.1017/s0956796824000133>.
- Wenhao Tang, Daniel Hillerström, James McKinna, Michel Steuwer, Ornela Dardha, Rongxiao Fu, and Sam Lindley. Structural subtyping as parametric polymorphism. *Proc. ACM Program. Lang.*, 7(OOPSLA2):1093–1121, 2023. doi: 10.1145/3622836. URL <https://doi.org/10.1145/3622836>.
- Wenhao Tang, Daniel Hillerström, Sam Lindley, and J. Garrett Morris. Soundly handling linearity. *Proc. ACM Program. Lang.*, 8(POPL):1600–1628, 2024. doi: 10.1145/3632896. URL <https://doi.org/10.1145/3632896>.
- Wenhao Tang, Leo White, Stephen Dolan, Daniel Hillerström, Sam Lindley, and Anton Lorenzen. Modal effect types. *Proc. ACM Program. Lang.*, 9(OOPSLA1):1130–1157, 2025. doi: 10.1145/3720476. URL <https://doi.org/10.1145/3720476>.
- Wenhao Tang, Shengyi Jiang, Aghilas Y. Boussaa, Sam Lindley, and Bruno C. d. S. Oliveira. Bidirectional typing with freezing, skeletons, and ghosts. *CoRR*, abs/2607.16061, 2026. doi: 10.48550/ARXIV.2607.16061. URL <https://doi.org/10.48550/arXiv.2607.16061>.
- Jerzy Tiuryn and Pawel Urzyczyn. The subtyping problem for second-order types is undecidable. In *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*, 1996.
- Birthe van den Berg and Tom Schrijvers. A framework for higher-order effects & handlers. *Sci. Comput. Program.*, 234:103086, 2024. doi: 10.1016/J.SCICO.2024.103086. URL <https://doi.org/10.1016/j.scico.2024.103086>.
- Dimitrios Vytiniotis, Stephanie Weirich, and Simon L. Peyton Jones. Boxy types: inference for higher-rank types and impredicativity. In John H. Reppy and Julia Lawall, editors, *Proceedings of the 11th ACM SIGPLAN International Conference on Functional Programming, ICFP 2006, Portland, Oregon, USA, September 16-21, 2006*, pages 251–262. ACM, 2006. doi: 10.1145/1159803.1159838. URL <https://doi.org/10.1145/1159803.1159838>.
- Philip Wadler. Is there a use for linear logic? In Charles Consel and Olivier Danvy, editors, *Proceedings of the Symposium on Partial Evaluation and Semantics-Based Program Manipulation, PEPM’91, Yale University, New Haven, Connecticut, USA, June 17-19, 1991*, pages 255–273. ACM, 1991. doi: 10.1145/115865.115894. URL <https://doi.org/10.1145/115865.115894>.

- Philip Wadler. The essence of functional programming. In *POPL*, pages 1–14. ACM, 1992. doi: 10.1145/143165.143169. URL <https://doi.org/10.1145/143165.143169>.
- Philip Wadler. The marriage of effects and monads. In Matthias Felleisen, Paul Hudak, and Christian Queinnec, editors, *Proceedings of the third ACM SIGPLAN International Conference on Functional Programming (ICFP '98)*, Baltimore, Maryland, USA, September 27–29, 1998, pages 63–74. ACM, 1998. doi: 10.1145/289423.289429. URL <https://doi.org/10.1145/289423.289429>.
- Mitchell Wand. Complete type inference for simple objects. In *Proceedings of the Symposium on Logic in Computer Science (LICS '87)*, Ithaca, New York, USA, June 22–25, 1987, pages 37–44. IEEE Computer Society, 1987. URL <http://www.ccs.neu.edu/home/wand/papers/wand-lics-87.pdf>.
- J. B. Wells. Typability and type checking in system F are equivalent and undecidable. *Ann. Pure Appl. Log.*, 98(1-3):111–156, 1999. doi: 10.1016/S0168-0072(98)00047-5. URL [https://doi.org/10.1016/S0168-0072\(98\)00047-5](https://doi.org/10.1016/S0168-0072(98)00047-5).
- Ludwig Wittgenstein. Logisch-philosophische abhandlung. *Annalen der Naturphilosophie*, 14(3/4):185–262, 1921.
- Andrew K. Wright. Simple imperative polymorphism. *LISP Symb. Comput.*, 8(4):343–355, 1995.
- Nicolas Wu, Tom Schrijvers, and Ralf Hinze. Effect handlers in scope. In Wouter Swierstra, editor, *Proceedings of the 2014 ACM SIGPLAN symposium on Haskell*, Gothenburg, Sweden, September 4–5, 2014, pages 1–12. ACM, 2014. doi: 10.1145/2633357.2633358. URL <https://doi.org/10.1145/2633357.2633358>.
- Ningning Xie, Jonathan Immanuel Brachthäuser, Daniel Hillerström, Philipp Schuster, and Daan Leijen. Effect handlers, evidently. *Proc. ACM Program. Lang.*, 4(ICFP):99:1–99:29, 2020. doi: 10.1145/3408981. URL <https://doi.org/10.1145/3408981>.
- Ningning Xie, Youyou Cong, Kazuki Ikemori, and Daan Leijen. First-class names for effect handlers. *Proc. ACM Program. Lang.*, 6(OOPSLA2):30–59, 2022. doi: 10.1145/3563289. URL <https://doi.org/10.1145/3563289>.
- Yichen Xu, Oliver Bračevac, Cao Nguyen Pham, and Martin Odersky. What’s in the box: Ergonomic and expressive capture tracking over generic data structures. *Proc. ACM Program. Lang.*, 9(OOPSLA2), October 2025. doi: 10.1145/3763112. URL <https://doi.org/10.1145/3763112>.

- Xu Xue and Bruno C. d. S. Oliveira. Contextual typing. *Proc. ACM Program. Lang.*, 8 (ICFP), August 2024. doi: 10.1145/3674655. URL <https://doi.org/10.1145/3674655>.
- Xu Xue, Chen Cui, Shengyi Jiang, and Bruno C. d. S. Oliveira. Local contextual type inference. *Proc. ACM Program. Lang.*, 10(POPL), January 2026. doi: 10.1145/3776653. URL <https://doi.org/10.1145/3776653>.
- Jeremy Yallop. Effects bibliography, 2026. URL <https://github.com/yallop/effects-bibliography>.
- Zhixuan Yang. *Structure and Language of Higher-Order Algebraic Effects*. PhD thesis, Imperial College London, 2024. URL <https://doi.org/10.25560/122484>.
- Zhixuan Yang and Nicolas Wu. Modular models of monoids with operations. *Proc. ACM Program. Lang.*, 7(ICFP):566–603, 2023. doi: 10.1145/3607850. URL <https://doi.org/10.1145/3607850>.
- Zhixuan Yang and Nicolas Wu. Handling higher-order effectful operations with judgemental monadic laws. *Proc. ACM Program. Lang.*, 10(POPL):1036–1065, 2026. doi: 10.1145/3776678. URL <https://doi.org/10.1145/3776678>.
- Zhixuan Yang, Marco Paviotti, Nicolas Wu, Birthe van den Berg, and Tom Schrijvers. Structured handling of scoped effects. In Ilya Sergey, editor, *Programming Languages and Systems - 31st European Symposium on Programming, ESOP 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings*, volume 13240 of *Lecture Notes in Computer Science*, pages 462–491. Springer, 2022. doi: 10.1007/978-3-030-99336-8_17. URL https://doi.org/10.1007/978-3-030-99336-8_17.
- Takuma Yoshioka, Taro Sekiyama, and Atsushi Igarashi. Abstracting effect systems for algebraic effect handlers. *CoRR*, abs/2404.16381, 2024. doi: 10.48550/ARXIV.2404.16381. URL <https://doi.org/10.48550/arXiv.2404.16381>.
- Yizhou Zhang and Andrew C. Myers. Abstraction-safe effect handlers via tunneling. *Proc. ACM Program. Lang.*, 3(POPL):5:1–5:29, 2019. doi: 10.1145/3290318. URL <https://doi.org/10.1145/3290318>.
- Jinxu Zhao and Bruno C. d. S. Oliveira. Elementary type inference. In Karim Ali and Jan Vitek, editors, *36th European Conference on Object-Oriented Programming, ECOOP 2022, June 6-10, 2022, Berlin, Germany*, volume 222 of *LIPICs*, pages 2:1–2:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi: 10.4230/LIPICs.ECOOP.2022.2. URL <https://doi.org/10.4230/LIPICs.ECOOP.2022.2>.

Nikita Zyuzin and Aleksandar Nanevski. Contextual modal types for algebraic effects and handlers. *Proc. ACM Program. Lang.*, 5(ICFP):1–29, 2021. doi: 10.1145/3473580. URL <https://doi.org/10.1145/3473580>.