FOCUSING ON MODULAR REFINEMENT TYPING

by

DIMITRIOS JAMES ECONOMOU

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Doctor of Philosophy

Queen's University
Kingston, Ontario, Canada
September 2024

Copyright © Dimitrios James Economou, 2025

For Raven and Toffee

Abstract

Refinement typing algorithms are hard to design and understand. Liquid Haskell (LH) has powerful features like modular, recursive refinement of inductive data. But Liquid Haskell is relatively hard to understand as it lacks an explicit phase distinction between indices and programs. Index based systems like Dependent ML (DML) tend toward less power but, enjoying an index-program phase distinction, are easier to understand. We apply techniques from logic, type theory, and category theory to design a correct and simple bidirectional typing algorithm for modular recursive index refinements. We use focusing to design logically a call-by-push-value (CBPV) with algebraic datatypes. CBPV is a language known empirically to have good semantic properties even in the presence of computational effects like nontermination and errors. Bidirectional type theories are a reliable way to combine rich type checking and inference. We prove our declarative system is semantically sound for standard mathematical models (domains). We prove our algorithmic system is decidable, sound, complete. Focusing and bidirectional typing combine elegantly with a new concept: *value-determined* existentials of input types under focus are guaranteed to be solved at the end of focusing stages, clearly outputting constraints decidable by an SMT solver. We believe this work improves our understanding of liquid refinement typing, and we hope it can guide or serve as the foundation for implementations thereof.

Acknowledgments

I thank many people including:

- My advisor Jana Dunfield for sparking my interest in formal logic, for letting me work on this interesting yet practical thesis (and believing I could do so adequately), and for her remarkable patience, understanding, and support
- My very helpful and insightful collaborator and mentor Neel Krishnaswami
- My (advisory)/[defense] committees: (Dorothea Blostein, Christian Muise, [Juergen Dingel), David Skillicorn, Sean Kauffman, Erin Meger, especially Noam Zeilberger]
- Anonymous reviewers of Economou et al. [2023] for their important feedback
- Ondrej Baranovič [2023] for implementing the system of Economou et al. [2023]
- Taylor Smith for introducing me to Queen's and December Stuart and Nick Mertin for conversations in a reading group on programming language theory
- Administrative staff including Debby Robertson, Erin Gunsinger, and Zannatin Tazreen
- Matt Hammer for getting me interested in programming language theory, Jonathan Wise for introducing me to category theory, Sriram Sankaranarayanan for hiring me to program stability verification of dynamical systems in OCaml (C++ kind of scared undergraduate me away from working on a computer simulation of ultrafast lasers)

Statement of Originality

I state that my thesis work is my own, some adapted from Economou et al. [2023]. Neel Krishnaswami and Jana Dunfield are co-authors of this paper who, besides introducing key ideas (like value-determined indices) and sketching the initial system therein, also contributed their suggestions and feedback as I worked on the system(s) and metatheory. They contributed further suggestions and feedback to extensions of that paper in this thesis.

Contents

Abstract ii

Acknowledgments iii

Statement of Originality iv

Contents v

Chapter 1: Introduction 1
- 1.1 Type Systems and Their Refinement . . . 1
- 1.2 Thesis . . . 12
- 1.3 Contributions . . . 12
- 1.4 Methodology . . . 14
- 1.5 Overview . . . 17
- 1.6 Updates and Corrections (May 1, 2025) . . . 18

Chapter 2: Examples 19

Chapter 3: Background 26
- 3.1 Typing . . . 29
- 3.2 Dependent Typing and Refinement Typing . . . 33
- 3.3 Judgments, Their Presuppositions, and Value-Determined Indices . . . 54
- 3.4 Focusing & Its Applications to Programming Languages . . . 68
- 3.5 A Reading Guide . . . 76

Chapter 4: Focusing on an Unrefined System 78
- 4.1 A Sequent Calculus for Intuitionistic Propositional Logic . . . 78
- 4.2 A Strongly Focused Sequent Calculus for Intuitionistic Proof Search . . . 87
- 4.3 A Weakly Focused Intuition. Sequent Calculus, The Core of Our PL . . . 94
- 4.4 A Bidirectionally Typed Weakly Focused Intuitionistic Sequent Calculus . . . 105

Chapter 5: Declarative Unrefined System 113

5.1 Recursive Expressions and Algebraic Datatypes Added . . . 113
5.2 Operational and Denotational Semantics and Adequacy . . . 120
5.3 Denotational Semantics of and Substitution in the Unrefined System . . . 138

Chapter 6: Declarative Refined System 153
6.1 Context Well-Formedness, Index Sorting, and Type Well-Formedness . . . 164
6.2 Properties of the Index Domain . . . 175
6.3 Subtyping and Submeasuring . . . 182
6.4 Unrolling . . . 189
6.5 Typing . . . 198
6.6 Program Substitution . . . 206

Chapter 7: Semantics and Typing Soundness 211

Chapter 8: Algorithmic System 229
8.1 Algorithmic Contexts and Substitution . . . 233
8.2 Well-Formedness . . . 234
8.3 Verifying Constraints and Solving Existentials . . . 235
8.4 Subtyping . . . 236
8.5 Typing . . . 242

Chapter 9: Algorithmic Decidability, Soundness, Completeness 244
9.1 Context Extension and Well-Formedness . . . 244
9.2 Decidability . . . 247
9.3 Algorithmic Soundness . . . 248
9.4 Algorithmic Completeness . . . 251
9.4.1 Semideclarative System . . . 252
9.4.2 Judgmental Equivalence . . . 257
9.4.3 Dependency Mediation . . . 260
9.4.4 Proving Completeness . . . 262

Chapter 10: Conclusion 266
10.1 Summary . . . 266
10.2 Future Work . . . 267

Bibliography 270

Appendix A: Definitions and Figures 289
A.1 Syntax . . . 290
A.2 Judgments and Their Presuppositions . . . 292
A.3 Declarative System . . . 293

A.4 Unrefined System and Its Denotational Semantics . . . 314
A.5 Index erasure . . . 322
A.6 Denotational Semantics (Refined System) . . . 324
A.7 Algorithmic System . . . 330
A.8 Intermediate Systems for Algorithmic Completeness . . . 349
A.9 Miscellaneous Definitions . . . 360

Appendix B: On *cl* and The Equivalence of *cl* and det **361**

Appendix C: Syntactic Metatheory of Declarative (Refined) System **368**
C.1 Structural Properties . . . 408
C.2 Subtyping Properties . . . 423
C.3 Substitution Lemma and Subsumption Admissibility . . . 429

Appendix D: Unrefined System and Erasure **482**

Appendix E: Semantic Metatheory of Declarative (Refined) System **512**
E.1 Refined Type and Substitution Soundness . . . 540

Appendix F: Basic Properties of Algorithmic System **555**
F.1 Algorithmic Extension . . . 556
F.2 Algorithmic Substitution and Well-Formedness Properties . . . 559
F.3 Decidability . . . 561
F.4 Algorithmic Soundness . . . 566

Appendix G: Algorithmic Completeness **569**
G.1 Intermediate Metatheory . . . 571
G.2 Dependency Mediation . . . 587
G.3 Algorithmic Subtyping . . . 609
G.4 Algorithmic Typing . . . 619

Chapter 1

Introduction

1.1 Type Systems and Their Refinement

Software failure tends to be bad. The Mars Climate Orbiter was destroyed partly due to a unit mismatch [Oberg, 1999]. A cryptographic library leaked secrets due to ill-formed network requests [Durumeric et al., 2014]. Therac-25 killed people by radiation overdose due to a race condition [Leveson and Turner, 1993]. Preventing software failure tends to be good. To do so there are many ways proposed: developmental methodologies, unit testing, static analysis, programming paradigms, proof assistants, formal verification, type systems, and so on. Type systems [Cardelli, 1996], in particular, check and infer types, which specify program properties that help to ensure programs don't go wrong.

Type systems can check and infer types of programs either *statically*, before program execution, at *compile* time, or *dynamically*, during program execution, at *run* time. It is usually preferable to check and infer types statically (if possible) rather than dynamically because dynamic typing incurs a cost in space and time when running the program. There is no general-purpose (Turing complete) programming language that can verify whether

an arbitrary program written in it satisfies an arbitrary semantic property: by Rice's theorem [Rice, 1953], all non-trivial properties of programs are undecidable. However, by using type annotations and requiring syntax to be structured in a certain way, we can work around Rice's theorem and design reasonable type systems, with sensible syntax, and without requiring too many annotations, in which many apparently non-trivial properties can be verified automatically. These properties help to ensure that our programs won't go wrong when run.

Indeed, "well-typed programs cannot 'go wrong' " [Milner, 1978], but only relative to a given semantics (if the type system is proven sound with respect to it and nothing goes wrong in the real world, like the runtime C library, the hardware, or the electric supply). Unfortunately, well-typed programs go wrong, in many ways that matter, but about which a conventional type system cannot speak: divisions by zero, out-of-bounds array accesses, information leaks. To prove a type system rules out (at compile time) more run-time errors, its semantics must be refined. However, there is often not enough type structure with which to express the semantics statically. So, we must refine our types with more information that tends to be related to programs. Great care is needed, though, because incorporating too much information (such as nonterminating programs themselves, as may happen in a *dependent* type system, where arbitrary programs may be used in types) can spoil good properties of the type system, like the soundness and decidability of type checking and inference.

Consider the inductive type $\mathsf{List}\ A$ of lists with entries of type A. Such a list is either empty/nil ($[]$) or a term x of type A together with a tail list xs (that is, $x :: xs$). In a typed functional language like Haskell or OCaml, the programmer can define such a type by specifying its constructors:

```
data List Type where
  [] : List A
  (::) : A → List A → List A
```

Suppose we define, by pattern matching, the function get, that takes a list xs and a natural number[1] y, and returns the yth element of xs (where the first element is numbered zero):

```
get [] y = error "Out of bounds"
get (x :: xs) zero = x
get (x :: xs) (suc y) = get xs y
```

A conventional type system has no issue checking get against, say, the type

$$\mathsf{List}\ A \to \mathsf{Nat} \to A$$

(for any type A), but get is possibly unsafe because it throws an out-of-bounds error when the input number is greater than or equal to the length of the input list[2]. If it should be impossible for get to throw such an error, then get must have a type where the input number is restricted to natural numbers strictly less than the length of the input list. Ideally, the programmer would simply refine the type of get, while leaving the program alone (except, perhaps, for omitting the first clause, the one throwing the out of bounds error).

This, in contrast to dependent types [Martin-Löf, 1984], is the chief aim of *refinement types* [Freeman and Pfenning, 1991]: to increase the expressive power of a pre-existing (unrefined) type system, while keeping the latter's good properties, like typing soundness and decidability of typing, so that programmers are not too burdened with refactoring their code or manually providing tedious proofs if they don't need to. In other words, the point of refinement types is to increase the expressive power of a given type system while preserving

[1] In this thesis we always count 0 as a natural number (in addition to $1, 2, \ldots$).

[2] The length of a list is its (natural) number of elements.

high automation (of typing for existing programs), whereas the point of dependent types is to be maximally expressive, which necessarily comes at the cost of automation (which dependent type system designers may try to increase after the fact of high expressivity, possibly using refinement types).

To refine get's type so as to rule out, statically, run-time out-of-bounds errors, we need to compare numbers against list lengths. Thus, we refine the type of lists by their length: $\{\nu : \mathsf{List}\, A \mid \mathsf{len}\, \nu = n\}$, the type of lists ν of length n. This type looks a bit worrying, though, because the measurement, $\mathsf{len}\, \nu = n$, seems to use a recursive program, len. The structurally recursive

$$\mathsf{len}\, [] = 0$$
$$\mathsf{len}\, (x :: xs) = 1 + \mathsf{len}\, xs$$

happens to terminate when applied to lists, but there is no general algorithm for deciding whether an arbitrary computation terminates [Turing, 1936]. As such, we might prefer not to use ordinary recursive programs directly in our refinement types at all. Indeed, doing so would seem to violate a phase distinction[3] [Moggi, 1989a, Harper et al., 1990] between the *static* (compile time) specification and the *dynamic* (run time) program, which seems indispensable for *decidable* typing.

The refinement type system Dependent ML (DML) [Xi, 1998] provides a phase distinction in refining ML by an index domain which has no run-time content.[4] Type checking and inference in DML is only decidable when it generates constraints whose validity is decidable. In practice, DML *did* generate decidable constraints, but that was not guaranteed by its design. To define the refinement type of lists of length n in DML we have to add an

[3] A language has a *phase distinction* if it can distinguish aspects that are relevant at run time from those that are relevant only at compile time.

[4] Perhaps today "Refinement ML" might seem more appropriate than Dependent ML. However, when DML was created, "refinement types" meant *datasort* refinement systems specifically. Nonetheless, the abstract of Xi [1998] describes DML as "another attempt towards refining ... type systems ..., following the step of refinement types (Freeman and Pfenning 1991)."

index representing n to the datatype and modify its constructors like so:

```
data List Type ℕ where
  [] : List A 0
  (::) : A → List A k → List A (k + 1)
```

It is not too hard to see that this style of specification quickly becomes unpleasantly crowded when we want to add more properties, especially if those properties can start referring to each other (within the type of one constructor). Not only that, but any annotations using an indexed type would have to be modified to reflect any changes in the index (any new properties added). However, logically we should be able to specify these different properties independently and then invoke them by name as needed in our annotations. The refinements of conventional DML are unfortunately not modular in this way. Theoretically speaking, we should be able to make it so, though, because Atkey et al. [2012] showed these two approaches to refining inductive types are equivalent (albeit in a different setting, fibrational semantics for dependent type theory).

On the plus side, DML's distinction between indices and programs allows it to support refinement types in the presence of computational effects (such as nontermination, exceptions, and mutable references) in a relatively straightforward manner. Further, the index-program distinction clarifies how to give a denotational semantics: a refinement type denotes (read: "means") a subset of what the type's erasure (of indexes) denotes and a program denotes precisely what its erasure denotes. Dependent type systems, by contrast, do not have such an erasure semantics.

It seems liquid type systems [Rondon et al., 2008, Kawaguchi et al., 2009, Vazou et al., 2013, 2014] achieve highly expressive *and modular*, yet sound and decidable *recursive refinements* [Kawaguchi et al., 2009] of inductive types by a kind of phase distinction: namely, by restricting the recursive predicates of specifications to terminating *measures*

(like len) that soundly characterize, in a theory decidable by off-the-shelf tools like SMT solvers, the static structure of inductive types. Unlike DML, liquid typing can, for example, modularly add the measure of whether a list of natural numbers is in increasing order (along with other measures), while remaining decidable. However, liquid typing's lack of index-program distinction makes it unclear how to give it a denotational semantics, and has also in the past led to subtleties involving the interaction between effects and evaluation strategy (we elaborate later in this section and Chapter 3). Vazou et al. [2014] appear to provide a denotational semantics in Section 3.3, but this is not really a denotational semantics in the sense we desire, because it is defined in terms of an operational semantics and not a separate and well-established mathematical model (such as domain theory).

Let's return to the get example. Following the tradition of index refinement [Xi, 1998], we maintain a phase distinction by syntactically distinguishing index terms, which can safely appear in types, from program terms, which cannot. In this approach, we want to check get against a more informative type

$$\forall l : \mathbb{N}. \underbrace{\{v : \mathsf{List}\, A \mid \mathsf{len}\, v = l\}}_{\mathsf{List}(A)(l)} \to \{v : \mathsf{Nat} \mid v < l\} \to A$$

quantifying over *indexes* l of *sort* $\mathbb{N}$ (natural numbers) and requiring the accessing number (second argument) to be less than l. However, this type isn't quite right, because Nat is a type and $\mathbb{N}$ is a sort, so writing "$v < l$" confounds our phase distinction between programs and indexes. Instead, the type should look more like

$$\forall l : \mathbb{N}. \{v : \mathsf{List}\, A \mid \mathsf{len}\, v = l\} \to \{v : \mathsf{Nat} \mid \mathsf{index}\, v < l\} \to A$$

where

$\mathsf{index}\ \mathsf{zero} = 0$
$\mathsf{index}\ (\mathsf{suc}\ y) = 1 + \mathsf{index}\ y$

computes the index term of sort $\mathbb{N}$ that corresponds to a program term of type Nat, by a structural recursion homologous to that of len. The third clause of get has a nonempty list as input, so its index (length) must be $1+m$ for some m; the type checker assumes $\mathsf{index}\,(\mathsf{suc}\,y) < 1+m$; by the aforementioned homology, these constraints are again satisfied at the recursive call ($\mathsf{index}\,y < m$), until the second clause returns ($0 < 1+m'$). The first clause of get is impossible, because no natural number is less than zero. We can therefore safely remove this clause, or (equivalently) replace `error` with `unreachable`, which checks against any type under a logically inconsistent context, such as $l : \mathbb{N}, n : \mathbb{N}, l = 0, n < l$ in this case.

$\mathsf{get} : \forall l, n : \mathbb{N}.\ \{v : \mathsf{List}\ A \mid \mathsf{len}\ v = l\} \to \{v : \mathsf{Nat} \mid \mathsf{index}\ v = n\} \wedge (n < l) \to A$
$\mathsf{get}\ []\ y = \mathtt{unreachable}$ -- $l = 0$ and $n : \mathbb{N}$ so $n \not< l$
$\mathsf{get}\ (x :: xs)\ \mathsf{zero} = x$
$\mathsf{get}\ (x :: xs)\ (\mathsf{suc}\ y) = \mathsf{get}\ xs\ y$

Applying get to a list and a number determines the indexes l and n. We say that l and n are *value-determined* (here by applying the function to values). If (perhaps in a recursive call) get is called with an empty list [] and a natural number, then l is determined to be 0, and since no index that is both negative and a natural number exists, no out-of-bounds error can arise by calling get. (Further, because $l : \mathbb{N}$ strictly decreases at recursive calls, calling get terminates.)

While this kind of reasoning about get's refinement type may seem straightforward, how do we generalize it to recursion over any algebraic datatype (ADT)? What are its logical and semantic ingredients? How do we use these ingredients to concoct a type system with decidable typing, good (localized) error messages and so on, while also keeping its

metatheory relatively stable or maintainable under various extensions or different evaluation strategies?

Type systems that can do this kind of reasoning automatically, especially in a way that can handle any evaluation strategy, are hard to design correctly. Indeed, the techniques used in the original (call-by-value) liquid type system(s) [Rondon et al., 2008, Kawaguchi et al., 2009] had to be modified for Haskell, essentially because of Haskell's *call-by-name* evaluation order [Vazou et al., 2014]. The basic issue was that binders can bind in (static) refinement predicates, which is fine when binders only bind values (as in call-by-value), but not when they bind computations which may not terminate (as in call-by-name). Liquid Haskell regained (operational) typing soundness by introducing ad hoc restrictions that involve approximating whether binders terminate, and using the refinement logic to verify termination.

Levy [2004] introduced the paradigm and calculus *call-by-push-value* (CBPV) which puts both call-by-name and call-by-value on equal footing in the storm of computational effects (such as nontermination). CBPV subsumes both call-by-name (CBN) and call-by-value (CBV) functional languages, because it allows us to encode both via type discipline. In particular, CBPV polarizes types into (positive) value types P and (negative) computation types N, and provides polarity shifts[5] $\uparrow P$ (negative) and $\downarrow N$ (positive); the monads functional programmers use to manage effects arise as the composite $\downarrow\uparrow -$. These polarity shifts are the same as those arising from the focusing technique of proof theory. CBPV can be derived *logically* by way of focalization of intuitionistic logic [Espírito Santo, 2017], which is the approach we take: see Chapter 4. CBPV is a good foundation for a refinement

[5]We note a few differences from some of the usual CBPV notation and terminology. In CBPV, $\uparrow -$ is usually written $\mathsf{F}-$ and $\downarrow -$ is usually written $\mathsf{U}-$ (or similar). Think liFt and thUnk. Also, usually in CBPV positive/value types and negative/computation types use the same capital letter like A, B, and C but computation types are underlined $\underline{A}$. We use A, B, and C for types, P, Q, and R for positive types and N, M, and L for negative types.

typing algorithm: designing refinement typing algorithms is challenging and sensitive to effects and evaluation strategy, so it helps to refine a language that makes evaluation order explicit. We leverage focusing and our technique of value-determined indexes (a novelty in the DML tradition, as it were, which was inspired by the practical experience of Liquid Haskell) to guarantee (like Liquid Haskell) the generation of SMT-solvable constraints.

Andreoli [1992] introduced *focusing* to reduce the search space for proofs (programs) of logical formulas (types), by exploiting the property that some inference rules are invertible[6]. At roughly the same time, Girard [1991] independently introduced (weak—explained later) focusing for classical logic to restore an analogue of the Church–Rosser property for cut elimination. In relation to functional programming, focusing has been used, for example, to explain not only the interaction between evaluation order and effects [Zeilberger, 2009] which was at one point problematic for Liquid Haskell, but also features such as pattern matching [Krishnaswami, 2009], and also to reason about contextual program equivalence [Rioux and Zdancewic, 2020]. Focusing has been applied to design a union and intersection refinement typing algorithm [Zeilberger, 2009]. As far as we know, prior to this work focusing has not been used directly to design an *index* refinement typing algorithm.

Another special ingredient, bidirectional typing [Pierce and Turner, 2000] systematizes the difference between input (type checking) and output (type inference), and seems to fit nicely with focused systems [Dunfield and Krishnaswami, 2021]. Bidirectional typing has its own practical virtues: it is easy to implement (if inputs and outputs fit together properly, that is, if the system is well-moded); it scales well (to refinement types, higher-rank polymorphism [Dunfield and Krishnaswami, 2019], subtyping, effects—and so does CBPV); it

[6] An *invertible* rule is one in which its conclusion implies its premises, so no information is lost in moving from its conclusion to any of its premises.

leads to localized error messages; and it clarifies where type annotations are needed, typically in reasonable positions (such as at the top level) that are helpful as machine-checked documentation. In our system, annotations are needed only for recursive functions (to express termination refinements) and top-level definitions. Judgments other than typing ones can also be bidirectional, so perhaps the more general technique we are using here could be called *mode-conscious* judgment.

By focusing on the typing of function argument lists and results, our focused system guarantees that value-determined existential indexes (unification variables) are solved before passing output constraints to an SMT solver. For example, when our system infers a type for $\mathsf{get}([3,1,2],2)$, we first use the top-level annotation of get to synthesize the type

$$\downarrow(\forall l,n:\mathbb{N}.\ \{v:\mathsf{List}\,A \mid \mathsf{len}\ v = l\} \to \{v:\mathsf{Nat} \mid \mathsf{index}\ v = n\} \wedge (n < l) \to \uparrow A)$$

(in which we have added polarity shifts $\downarrow -$ and $\uparrow -$ arising from focusing). The downshift $\downarrow -$ takes a negative type to a positive type of suspended computations. Second, we check the argument list $([3,1,2],2)$ against the negative (universally quantified) type. The upshift $\uparrow -$ takes a positive type A to negative type $\uparrow A$ (computations returning a value of type A). In typechecking the argument list, the upshift signifies the end of a focusing stage, at which point the first argument value $[3,1,2]$ will have determined l to be 3 and the second argument value 2 will have determined n to be 2, outputting an SMT constraint without existentials: $2 < 3$.

A focused and bidirectional approach therefore appears suitable, both theoretically and practically, for designing and implementing an expressive language for refinement typing that can handle any evaluation strategy and effect. We design a foundation on which to build liquid typing features that allows us to establish clear semantic correctness results,

as well as the completeness of a decidable bidirectional typing algorithm. In other words, this thesis is a first step toward metatheoretically reconciling DML and Liquid Haskell, using the proof-theoretic technique of focusing. The main technique is focusing, which we combine naturally with bidirectional typing and value-determined indexes (the latter being a key ingredient to make measures work). We show that bidirectional typing and logical focusing work very well together at managing the complex flow of information pertaining to indexes of recursive data. In particular, value-determined existential indexes of input types are solved within focusing stages, ultimately leading to the output of constraints in the quantifier-free fragment solvable by SMT solvers.

A thesis is a goal and a method of obtaining it. The goal of this thesis is to create a foundation for refinement typing that seeks the best of both worlds: Dependent ML and Liquid Haskell. Dependent ML, due to its phase distinction between static indices and dynamic programs, is relatively easy to understand, but it does not necessarily generate SMT solvable constraints and it is inconvenient to refine types as it is not modular like Liquid Haskell. Liquid Haskell guarantees SMT solvable constraints and greatly alleviates the modularity problem, but it is relatively hard to understand and to prove important metatheoretic properties of it. The method of this thesis is essentially "the"[7] Curry–Howard–Lambek correspondence—whose historical predecessor is the Brouwer–Heyting–Kolmogorov interpretation of intuitionistic logic [Troelstra, 2011]—between proofs in a logic, programs in a typed programming language, and morphisms in a category.

We begin logically with an application of the technique of *focusing* from proof theory, add recursion and ADTs, prove this (declarative) unrefined system is semantically correct, soundly add a declarative refinement layer with an erasure semantics, and prove it also has

[7] It's really a family of correspondences, not specifically about one particular (intuitionistic) logic. Classical sequent calculus, for example, also has a Curry–Howard correspondence.

nice algorithmic properties (by way of an equivalent algorithmic system).

1.2 Thesis

The combination of logical focusing, bidirectional typing, categorical semantics, and our novel concept of value-determined indexes is an elegant foundation for a semantically correct and decidable typing algorithm for index refinements (à la Dependent ML) of algebraic data modularly by measures (à la Liquid Haskell).

1.3 Contributions

Our two key contributions are *both* a declarative/logical/semantic *and* an algorithmic account of modular, recursive, index-based refinement of algebraic data types. For the logical account, we design a declarative type system in a *bidirectional* and *focused* style, resulting in a system with clear operational and denotational semantics[8] and soundness/correctness proofs, and which is convenient for type theorists of programming languages. The declarative system conjures index solutions to existentials. For the algorithmic account, we design a type system mirroring the declarative one but mechanically solving existentials, and prove it is decidable, as well as sound and complete. We contribute:

- A polarized declarative type system, including (polarized) subtyping, universal types, existential types, and index refinements with ordinary SMT constraints, as well as modular, recursive predicates on inductive data (simple functional index "programs" which can express things like "this list is in increasing order").

[8]The operational semantics and the denotational semantics of a system is meant to capture the "meaning" of the system, but the former tends to capture more the computational sense and the latter more the mathematical reference.

- A proof that declarative typing is stable under substitution, which requires proving, among other things, that subtyping is transitive and typing subsumption is admissible.[9]

- An operational semantics of the declarative system which is based on an extension of cut elimination in intuitionistic sequent calculus and which is tied to the erasure of type annotations.

- A clear denotational semantics of the declarative system, based on elementary domain theory.

- Operational type soundness and the equivalence of the operational and denotational semantics.

- A typing soundness proof with respect to our denotational semantics, which implies, relatively easily, both the refinement system's logical consistency and total correctness (at least denotationally)—even if the programs are *non-structurally* recursive. To prove typing soundness, we prove that value-determined index dependencies are sound: that is, semantic values uniquely determine value-determined index dependencies, semantically speaking (in particular, see Lemmas 7.1 and 7.2). *Operational* total correctness is a corollary of denotational total correctness and computational adequacy (which is the larger half of the equivalence of the operational and denotational semantics).

- A polarized subtyping algorithm, together with proofs that it is sound, complete and decidable.

[9] A proposed inference rule is *admissible* with respect to a system if, whenever the premises of the proposed rule are derivable, we can derive the proposed rule's conclusion using the system's inference rules.

- A polarized typing algorithm, together with proofs that it is sound, complete and decidable. Completeness relies on the combination of our novel concept of *value-determined* indices, focusing, and bidirectional typing. In particular, see Lemmas 9.6, 9.13, and 9.14.

We relatively easily obtain both semantic and algorithmic results for a realistic language essentially by starting with one technique (based on fundamental logical principles): focusing.

Definitions and proofs are organized in the appendix.

1.4 Methodology

We apply techniques drawn from the Holy Trinity[10] of computation [Harper, 2011]
so that we can prove important properties of our system largely using structural induction [Burstall, 1969]. Here are some of the main concepts we draw from logic, languages, and categories:

(1) Logic and type theory

- Focused proof search and its resulting polarized type structure [Andreoli, 1992]
- Judgments[11] and their presuppositions [Martin-Löf, 1996]
- Bidirectional typing [Pierce and Turner, 2000]

(2) Languages

[10] I use this terminology with my tongue in my cheek (though maybe there is more to the devotional connotation than we would like to admit; regardless, we happen to find ourselves somewhere in this crusade).

[11] Martin-Löf [2011] gave a talk arguing about the history of how "judgment" came to be a term of logic. An interesting point that comes up in the questions is the tension between the logical and the juridical but this tension is not exclusive to the term judgment.

- Call-by-push-value [Levy, 2004] (basically lambda calculus with syntax for binding computations and returning values)
- Type annotations (bidirectional typing), let-normality [Flanagan et al., 1993]

(3) Category theory

- Semantics of algebraic datatypes [Goguen et al., 1977]
- Fibrational interpretation of refining inductive types [Atkey et al., 2012]

The steps we take:

(1) We begin in Chapter 4 with a standard intuitionistic sequent calculus:
a certain presentation of a fragment of LJ [Gentzen, 1935].
We briefly study cut elimination in it.

(2) We strongly focalize the logic,
in a style like Simmons [2014] and Espírito Santo [2017], preferring the latter, with a view toward functional programming and in particular modular refinement of algebraic datatypes. We briefly study cut elimination in it.

(3) We apply a series of transformations to turn it from a proof search algorithm to a fairly standard looking CBPV programming language. These transformations should be sound and complete but I have not had time (or interest) to check this. However, I show directly that the overall result is equivalent to the unfocused logic.

(4) We bidirectionalize the unrefined language to get an unrefined bidirectional type system for it. Its operational semantics is given in terms of its erasure to program terms

(that is, proof terms) without type annotations: erasing a type annotation takes an analytic[12] cut and makes it non-analytic.

(5) In Chapter 5 we add algebraic data types and obtain our declarative unrefined system. The operational semantics is extended. We prove the operational and denotational semantics are equivalent.

(6) We prove that unrefined typing is stable under syntactic substitution.

(7) We give the unrefined system a denotational semantics and prove it is sound. We prove syntactic substitution is semantically sound in the unrefined system.

(8) We add index refinements in Chapter 6, obtaining the declarative refined system. We pay close attention to the organization of judgments and their presuppositions, especially formation (well-formedness) presuppositions, where the new concept of value-determined indices is used.

(9) We prove that refined typing is stable under syntactic substitution.

(10) We give the refined system a denotational semantics and prove it is sound in Chapter 7. Total correctness and logical consistency of the refined system are easy corollaries. Total correctness and computational adequacy implies programs with total refinement types terminate when run on an abstract machine. We prove syntactic substitution is semantically sound in the refined system.

(11) We design an algorithmic system (Chapter 8) and prove (Chapter 9) it is decidable, as well as sound and complete with respect to the declarative system.

[12] In the sense of the cut formula or type occurring (as an annotation) in the proof/program term. This usage is not standard in type theory, but it seems completely analogous to the situation in proof theory where a cut formula is defined to be analytic if it occurs as a subformula of the conclusion judgment.

1.5 Overview

Chapter 2 gives examples.

Chapter 3 briefly discusses salient historical background and related work, and provides an overview of the thesis. (Some future work is also discussed.)

Chapter 4 begins with a standard intuitionistic sequent calculus, focalizes it, and then applies a few transformations to design the core for our unrefined system. We study cut elimination, the basis of the operational semantics of our system.

Chapter 5 discusses the unrefined system with algebraic datatypes and recursion added. We prove unrefined typing is stable under syntactic substitution, and that the unrefined system and syntactic substitution in it is semantically sound (at domains). We prove that the operational semantics (based on cut elimination in the pre-recursion system obtained by annotation erasure) and the denotational semantics of the unrefined system are equivalent.

Chapter 6 discusses the refined system with modular recursive refinements added.

Chapter 7 discusses the semantics and typing soundness of the refined system.
The total correctness and logical consistency of the refined system are corollaries. Together with computational adequacy, total correctness implies that (totally) refined programs never diverge operationally.

Chapter 8 discusses the algorithmic system.

Chapter 9 discusses the metatheory of the algorithmic system:
decidability, soundness, completeness.

Chapter 10 concludes and discusses future work.

1.6 Updates and Corrections (May 1, 2025)

- Fixed a bug in Lemma 9.13 (Main Complete) by restricting the form of unrolled values v of $\texttt{into}(v)$: in particular, v cannot be zero or more injections applied to zero or more right-associative pairs with the innermost and rightmost factor a variable
- Corrections to cut elimination of the the main strongly focused and weakly focused logics; delete cut elimination presentations of intermediate logics
- Remove 0 from contexts of strongly focused systems
- Correction to the recursion case of computational adequacy proof
- Fix statement of Theorem 5.2 to be general enough and clarify the sketch of its proof
- Fix title of Girard [1991]
- Deleted a parenthetical in the first section of this introduction
- Made minor typographical changes such as the font of index spines

Chapter 2

Examples

We show how to verify a non-structurally recursive mergesort function in our system: namely, that it terminates and returns an ordered list with the same length as the input list. We only consider sorting lists of natural numbers Nat, defined as $\exists n : \mathbb{N}.\, \mathsf{Nat}(n)$. For clarity, and continuity with Chapter 1, we sometimes use syntactic sugar such as clausal pattern-matching, combining let-binding with pattern-matching on the let-bound variable, using "if-then-else" rather than pattern-matching on boolean type, and combining two or more pattern-matching expressions into one with a nested pattern such as $x :: y :: xs$.

Given type A and $n : \mathbb{N}$, we define $\mathsf{List}(A)(n)$ by $\{v : \mathsf{List}\, A \mid \mathsf{len}\, v = n\}$. Modulo a small difference (see Sec. 6.4), our unrolling judgment unrolls $\mathsf{List}(A)(n)$ to

$$\big(1 \wedge (n = 0)\big) + \big(A \times \exists n' : \mathbb{N}.\, \big\{v : \mathsf{List}\, A \;\big|\; \mathsf{len}\, v = n'\big\} \wedge (n = 1 + n')\big)$$

which is a refinement of $1 + (|A| \times \mathsf{List}\ |A|)$. This is an unrolling of the inductive type, not the inductive type itself, so we must roll values of it into the inductive type. We use syntactic sugar: namely, [] stands for $\mathtt{into}(\mathtt{inj}_1\ ())$ and $x :: xs$ stands for $\mathtt{into}(\mathtt{inj}_2\ \langle x, xs\rangle)$.

Just as we need a natural number type associating natural number program values with

natural number indexes, we need a boolean type of values corresponding to boolean indexes. To this end, define the measure

$\mathsf{ixbool} : (1+1) \to \mathbb{B}$
$\mathsf{ixbool}\ \mathtt{true} = \mathsf{tt}$
$\mathsf{ixbool}\ \mathtt{false} = \mathsf{ff}$

Given $b : \mathbb{B}$, the singleton type of a program value corresponding to index boolean b is $\mathsf{Bool}(b) = \{v : \mathsf{Bool} \mid \mathsf{ixbool}\ v = b\}$. Our unrolling judgment (Sec. 6.4) outputs the following type, a refinement of the boolean type encoded as $1+1$:

$$\big(1 \wedge (b = \mathsf{tt})\big) + \big(1 \wedge (b = \mathsf{ff})\big)$$

We encode `true` as $\mathtt{into}(\mathtt{inj}_1\ ())$ which has singleton type $\mathsf{Bool}(\mathsf{tt})$, and `false` as $\mathtt{into}(\mathtt{inj}_2\ ())$ which has singleton type $\mathsf{Bool}(\mathsf{ff})$. The boolean type Bool is defined as $\exists b : \mathbb{B}.\, \mathsf{Bool}(b)$.

Assume we have the following:

$\mathsf{add} : \downarrow(\forall m,n : \mathbb{N}.\, \mathsf{Nat}(m) \to \mathsf{Nat}(n) \to \uparrow \mathsf{Nat}(m+n))$
$\mathsf{sub} : \downarrow(\forall m,n : \mathbb{N}.\, (n \le m) \supset \mathsf{Nat}(m) \to \mathsf{Nat}(n) \to \uparrow \mathsf{Nat}(m-n))$
$\mathsf{div} : \downarrow(\forall m,n : \mathbb{N}.\, (n \neq 0) \supset \mathsf{Nat}(m) \to \mathsf{Nat}(n) \to \uparrow \mathsf{Nat}(m \div n))$
$\mathsf{lt} : \downarrow(\forall m,n : \mathbb{N}.\, \mathsf{Nat}(m) \to \mathsf{Nat}(n) \to \uparrow \exists b : \mathbb{B}.\, \mathsf{Bool}(b) \wedge (b = (m < n)))$
$\mathsf{len} : \downarrow(\forall n : \mathbb{N}.\, \mathsf{List}(\mathsf{Nat})(n) \to \uparrow \mathsf{Nat}(n))$
$[] : \mathsf{List}(\mathsf{Nat})(0)$
$(::) : \downarrow(\forall n : \mathbb{N}.\, \mathsf{Nat} \to \mathsf{List}(\mathsf{Nat})(n) \to \uparrow \mathsf{List}(\mathsf{Nat})(1+n))$

The SMT solver Z3 [de Moura and Bjørner, 2008], for example, supports integer division (and modulo and remainder operators); internally, these operations are translated to multiplication. Here, we are considering natural number indexes, but we can add the constraint $n \geq 0$ (for naturals n) when translating them to integers in an SMT solver such as Z3. Integer division (nonlinear integer arithmetic) in general is not SMT decidable, but for this mergesort example, n is always instantiated to a constant (in this case, the numeral 2),

which is decidable. Note that Z3 supports division by zero, but our div has a guard requiring the divisor to be nonzero ($n \neq 0$), so we need not consider this. Division on naturals takes the floor of what would otherwise be a rational number (for example, $3 \div 2 = 1$).

First, we define the function merge for merging two lists while preserving order. It takes two vectors (lists indexed by length) as inputs and has a termination-ensuring guard that requires the output list to have a length that is the sum of the two input lengths. Since at least one list decreases in length at recursive calls, so does the sum of their lengths, implying the function terminates when applied.[1]

$$
\begin{aligned}
&\mathsf{merge} : \forall n, n_1, n_2 : \mathbb{N}.\, (n = n_1 + n_2) \supset \mathsf{List}(\mathsf{Nat})(n_1) \to \mathsf{List}(\mathsf{Nat})(n_2) \\
&\quad \to \uparrow \mathsf{List}(\mathsf{Nat})(n) \\
&\mathsf{merge}\ [\,]\ xs_2 = \mathtt{return}\, xs_2 \\
&\mathsf{merge}\ xs_1\ [\,] = \mathtt{return}\, xs_1 \\
&\mathsf{merge}\ (x_1 :: xs_1)\ (x_2 :: xs_2) = \\
&\quad \mathsf{if}\ \mathsf{lt}(x_1, x_2)\ \mathtt{then} \\
&\qquad \mathsf{let}\ \mathit{recresult} = \mathsf{merge}(xs_1, (x_2 :: xs_2)); \\
&\qquad \mathsf{let}\ \mathit{result} = x_1 :: \mathit{recresult}; \\
&\qquad \mathtt{return}\, \mathit{result} \\
&\quad \mathsf{else} \\
&\qquad \mathsf{let}\ \mathit{recresult} = \mathsf{merge}((x_1 :: xs_1), xs_2); \\
&\qquad \mathsf{let}\ \mathit{result} = x_2 :: \mathit{recresult}; \\
&\qquad \mathtt{return}\, \mathit{result}
\end{aligned}
$$

In a well-typed let-binding $\mathsf{let}\ x = g;\ e$ the bound expression g is a value-returning computation (that is, has upshift type), and e is a computation that binds to x the value (of positive type) resulting from computing g. (Liquid Haskell, lacking CBPV's type distinction between computations and values, instead approximates whether binders terminate to a value.) Since x has positive type, we can match it against patterns (see, for example, the final clause of split, discussed next).

[1] Economou et al. [2023] used a ghost parameter, a program value representing the sum n, but we don't need such a ghost in the upgraded system presented in this thesis: n_1 and n_2 are determined so $n = n_1 + n_2$ determines n.

We now define the function split that takes a list and splits it into two lists. It is a standard "every-other" implementation, and we have to be a bit careful about the refinement type so as not to be "off by one" in the lengths of the resulting lists.

$\mathsf{split} : \forall n : \mathbb{N}.\, \mathsf{List}(\mathsf{Nat})(n) \to \uparrow(\mathsf{List}(\mathsf{Nat})((n+1) \div 2)) \times (\mathsf{List}(\mathsf{Nat})(n - ((n+1) \div 2)))$
$\mathsf{split}\ [] = \mathtt{return}\ \langle [], [] \rangle$
$\mathsf{split}\ [x] = \mathtt{return}\ \langle [x], [] \rangle$
$\mathsf{split}\ x_1 :: x_2 :: xs =$
  $\mathsf{let}\ \mathit{recresult} = \mathsf{split}(xs);$
  $\mathtt{match}\ \mathit{recresult}\ \{$
    $\langle xs_1, xs_2 \rangle \Rightarrow \mathtt{return}\ \langle x_1 :: xs_1, x_2 :: xs_2 \rangle$
  $\}$

We are ready to implement a mergesort that is verified to terminate and to return a list of the same length as the input list. We introduce syntactic sugar for a let-binding followed by pattern-matching on its result.

$\mathsf{mergesort} : \forall n : \mathbb{N}.\, \mathsf{List}(\mathsf{Nat})(n) \to \uparrow \mathsf{List}(\mathsf{Nat})(n)$
$\mathsf{mergesort}\ [] = \mathtt{return}\ []$
$\mathsf{mergesort}\ [x] = \mathtt{return}\ [x]$
$\mathsf{mergesort}\ xs =$ -- $n \geq 2$
  $\mathsf{let}\ \langle \mathit{leftxs}, \mathit{rightxs} \rangle = \mathsf{split}(xs);$
  $\mathsf{let}\ \mathit{sortleftxs} = \mathsf{mergesort}(\mathit{leftxs});$
  $\mathsf{let}\ \mathit{sortrightxs} = \mathsf{mergesort}(\mathit{rightxs});$
  $\mathtt{return}\, \mathsf{merge}(\mathit{sortleftxs}, \mathit{sortrightxs})$

Note that mergesort is *not* structurally recursive: its recursive calls are on lists obtained by splitting the input list roughly in half, not on the structure of the list ($- :: -$).

Suppose we want to change the refinement of mergesort to verify that the output is in increasing order. In DML style we would have to edit the types of the separate data constructors, which quickly gets unwieldy. Instead, we treat measures that belong together independently of other measures that have nothing to do with them. We can modularly change the refinement type of our mergesort function to verify what we want.

measure isincr
isincr [] $y = \texttt{true}$
isincr $(x :: xs)\ y = (x \geq y) \wedge \mathsf{isincr}\ xs\ x$

$\mathsf{mergesort} : \forall n : \mathbb{N}.\ \{v : \mathsf{NatList} \mid \mathsf{len}\, v = n\} \to \uparrow \{v : \mathsf{NatList} \mid \mathsf{len}\, v = n\ \&\&\ \mathsf{isincr}\, v\, 0 = \texttt{true}\}$

If we add indices representing finite multisets (of natural numbers, say) [Piskac and Kuncak, 2008] we can use a bag measure to verify that mergesort returns a permutation of the original list.

measure bag
bag [] $= \emptyset$
bag $(x :: xs) = \{x\}$ & bag xs

If two lists have the same bag of elements then they have the same length, so we can do away with the len measurement.

$\mathsf{mergesort} : \forall X : \mathsf{bag}(\mathbb{N}).\ \{v : \mathsf{NatList} \mid \mathsf{bag}\, v = X\}$
$\quad \to \uparrow \exists Y : \mathsf{bag}(\mathbb{N}).\ \{v : \mathsf{NatList} \mid \mathsf{isincr}\, v\, 0 = \texttt{true}\ \&\&\ \mathsf{bag}\ v = Y\} \wedge Y = X$

Let's consider a few more examples. In these examples whenever we speak of a Tree we mean a binary tree with natural number keys at nodes.

```
data Tree where
  leaf : Tree
  node : Nat → Tree → Tree → Tree
```

Example 2.1 (Trees With Keys In Range)**.**

$\mathsf{inrange} : \mathsf{Tree} \to \mathbb{N} \to \mathbb{N} \to \mathbb{B}$
$\mathsf{inrange}\ \ \mathsf{leaf}\ \ x\ \ y\ =\ x \leq y$
$\mathsf{inrange}\ \ (\mathsf{node}\ \ k\ \ l\ \ r)\ \ x\ \ y\ =\ x \leq k \wedge k \leq y \wedge \mathsf{inrange}(l,x,y) \wedge \mathsf{inrange}(r,x,y)$

$\mathsf{DigiTree} = \{v : \mathsf{Tree} \mid \mathsf{inrange}\, v\, 0\, 9 = \mathsf{tt}\}$

Example 2.2 (Rose Trees [Bird and Wadler, 1988, Meertens, 1988])**.** In Haskell a rose tree is usually a labelled node together with a list of rose subtrees. Ignoring polymorphism

(labels are naturals), we can specify this as an ADT refinement. We write applications (of index variables) to index spines, which are lists of indices and left and right projections .1 and .2.

```
data PreRoseTree where
  node : Nat → PreRoseTree → PreRoseTree
  nil : PreRoseTree
  cons : PreRoseTree → PreRoseTree → PreRoseTree

rt : PreRoseTree → 𝔹 × 𝔹
rt (node n y) = (rt(y,.2), ff)
rt nil = (ff, tt)
rt (cons x y) = (ff, rt(x,.1) ∧ rt(y,.2))

RoseTree = ∃a. {v : PreRoseTree | rt v = a} ∧ a.1
```

Example 2.3. Trees where for each node, the number of descendent leaves of that node is less than that node's key.

```
f : Tree → ℕ × 𝔹
f leaf = (1, tt)
f (node k l r) = (f(l,.1) + f(r,.1), f(l,.1) + f(r,.1) < k ∧ f(l,.2) ∧ f(r,.2))

∃a. {v : Tree | f v = a} ∧ a.2
```

Example 2.4 (Well Scoped Lambda Terms)**.** We use de Bruijn indices, which are natural numbers, to represent variable binding in lambda abstraction.

```
data LambdaTerm where
  var : Nat → LambdaTerm
  lam : LambdaTerm → LambdaTerm
  app : LambdaTerm → LambdaTerm → LambdaTerm

wellscoped : LambdaTerm → ℕ → 𝔹
wellscoped (var n) c = n < c
wellscoped (lam a) c = wellscoped(a, 1 + c)
wellscoped (app a b) c = wellscoped(a,c) ∧ wellscoped(b,c)

WellScopedLambdaTerm = {v : LambdaTerm | wellscoped v 0 = tt}
```

Example 2.5 (Red-Black Trees [Okasaki, 1998])**.** Let tt represent the color black and let ff represent red. A *red-black tree* is a colored (binary) tree such that

(1) Every leaf is black

(2) Every child of a red node is black

(3) For every node, each path to a leaf has the same number of black nodes

The sort $(\mathbb{B} \times \mathbb{N}) \times \mathbb{B}$ represents $(\text{color} \times \text{black height}) \times \text{validity}$ where validity refers to whether or not the (colored and binary) tree is a valid red-black tree.

```
data ColoredTree where
  leaf : ColoredTree
  node : Bool → ColoredTree → ColoredTree → ColoredTree

rb : ColoredTree → (𝔹 × ℕ) × 𝔹
rb leaf = ((tt, 0), tt)
rb (node c l r) =
  if c = tt                                                          -- black
  then ((tt, 1 + rb(l,.1,.2)), rb(l,.1,.2) = rb(r,.1,.2))
  else
    ((ff, rb(l,.1,.2)), rb(l,.1,.2) = rb(r,.1,.2) ∧ rb(l,.1,.1) = rb(r,.1,.1) ∧ rb(r,.1,.1) = tt)

RedBlackTree = ∃a. {v : ColoredTree | rb v = a} ∧ a.2
```

Chapter 3

Background

This thesis is a step toward reconciling Dependent ML and Liquid Haskell. The main thing we get from DML is the index-program distinction. Liquid Haskell provides or inspires mainly three things. First, the observation of difficulties with effects and evaluation order inspired our use of CBPV. Second, we study measures, which are modular. Third, our concept of value-determined indices was inspired by the observation that variables appearing in liquid refinements correspond to inputs or outputs of functions. We contend that this thesis lays an index-based foundation for studying liquid refinement typing.

Before diving into the details of our systems, in this chapter we give an overview of the central logical, semantic, type-theoretic, and algorithmic issues informing their design. Ultimately, the main technique we use, which we think allows us to obtain both semantic and algorithmic results in a largely straightforward way, is focusing.

Let us take a step back, or spread our wings, take flight and have a bird's-eye view.

The *goal* of this thesis is to design a refinement type system for a functional programming language. And the *method*? Well, Harper [2011] blogged about the central organizing principle of the theory of programming languages. This blog post has one of my favorite quotes:

> The central dogma of computational trinitarianism holds that Logic, Languages, and Categories are but three manifestations of one divine notion of computation. There is no preferred route to enlightenment: each aspect provides insights that comprise the experience of computation in our lives.
>
> Computational trinitarianism entails that any concept arising in one aspect should have meaning from the perspective of the other two. If you arrive at an insight that has importance for logic, languages, and categories, then you may feel sure that you have elucidated an essential concept of computation—you have made an enduring scientific discovery.

I do not claim to have made an enduring scientific discovery with this thesis (that's up to among other factors its future reception by the scientific community). Nor am I necessarily committed philosophically to this view of science. But in this thesis we do draw from logic, languages, and categories in a somewhat unified way. Anyway, Harper [2016] textbooked the following, in case the reader wants something more authoritative than a blog post, though it does not mention categories.

> The unification of logic and programming is called the *propositions as types* principle. It is a central organizing principle of the theory of programming languages. Propositions are identified with types, and proofs… programs. A programming technique corresponds to a method of proof; a proof technique… a method of programming.

More details can be found at the nLab entry for "computational trilogy" [nLab authors, 2024]. Adding physics and topology, perhaps we get the or a Holy Quincunx, but we're straying too far afield [Baez and Stay, 2011]. Let us stay focused.

Harper [2016] concisely reviews the history of statically typed languages:[1]

> The concept of the static semantics[2] of a programming language was historically slow to develop, perhaps because the earliest languages had relatively few features and only very weak type systems. The concept of a static semantics in the sense considered here was introduced in the definition of the Standard ML programming language [Milner et al., 1997], building on much earlier work by Church and others on the typed lambda-calculus [Barendregt, 1993]. The concept of introduction and elimination[3], and the associated inversion principle[4], was introduced by Gentzen in his pioneering work on natural deduction[5] [Gentzen, 1935]. These principles were applied to the structure of programming languages by Martin-Löf [1979], Martin-Löf [1984].

Since we will begin with logic, let's discuss it a bit more. Gentzen created his system of *natural deduction* to represent formally the reasoning of mathematicians in a way more natural (in the sense of more faithful to real mathematical practice) than Hilbert-style logical systems. Gentzen's natural deduction most naturally resulted in a formalization of *intuitionistic* logic, but mathematicians tended to reason *classically* (and they still largely

[1] Our citations in the quote differ slightly from the original source. I add my own footnotes.

[2] In this dissertation, a static semantics is a type system.

[3] Rules for introducing and eliminating logical connectives in natural deduction. In sequent calculus, there are left and right introduction rules instead (no top-down elimination rules). A cut-free sequent calculus therefore satisfes the *subformula property* that for each rule all formulas in its prerequisite sequents are subformulas of formulas in its concluded sequent.

[4] The inversion principle is basically that the introduction rules and elimination rules in natural deduction are inverses. This has also been called harmony in some of Frank Pfenning's notes and stability/harmony by Dummett [1991]. In sequent calculus, the inversion principle is that the left and right introduction rules are inverses. The inversion principle entails the Hauptsatz of Gentzen [1935]. But the inversion principle and its name came from work by Paul Lorenzen in the 1950s and Dag Prawitz uses it about a decade later to study normalization (an analogue to cut elimination) in natural deduction [Moriconi and Tesconi, 2008].

[5] Where he also invented sequent calculus, though natural deduction is anticipated by the work of the ancient philosopher Aristotle [Martin-Löf, 2011] and sequent calculus and proof theory in the work of the ancient Stoics [Bobzien, 2019].

do at this time, I'm guessing). Gentzen therefore invented *sequent calculus* as a mathematical tool to study formally judgments of consequence in *classical* logic, which pays closer attention to the way assumptions and conclusions are used, formalized by *structural rules* for manipulating sequences of assumptions and conclusions: exchange, weakening, contraction. He also studied intuitionistic sequent calculus. Both of his inventions—natural deduction and sequent calculus—were made in Gentzen's 1934/1935 PhD dissertation. This thesis will begin (in the next chapter) with a standard intuitionistic sequent calculus which is equivalent to a fragment (we omit universal and existential quantification over formulas, which corresponds to type polymorphism) of what Gentzen created. But the point of it is to be the core of a typed functional language, so let's move on to type systems.

A *type system* is a set of typing judgments together with rules for deriving them.

3.1 Typing

A *typing judgment* is derived by inference rules and asserts the type of a program term. For example $\Gamma \vdash e : A$ asserts that assuming or under the context Γ (assigning types to program variables), program term e has type A. Usually we presuppose metavariables in a judgment like Γ, e, and A are elements of languages generated by context-free grammars in Backus–Naur form. Assuming Γ is a given input, there are at least three ways to implement $\Gamma \vdash e : A$.

(1) Both e and A are inputs, and we check that e has type A (type *checking*).

(2) Given e as input, infer (output) a type A such that e has type A (type *inference*).

(3) Output a program e that has given input type A (*program synthesis*).

Using terminology from automated deduction and logic programming [Warren, 1978], we call the input or output status of a metavariable in a judgment its *mode*.

In complexity theory, it tends to be easier to check a given solution to a problem than to find one. Similarly, while inferring types is often desirable (it can be convenient for the programmer not to have to specify all the types), it tends to be computationally intractable as we move toward systems with richer types; but checking programs against given types is relatively easy.

The richer the type system, the harder it is to infer types. For example, System F, which extends the simply-typed lambda calculus with parametrically polymorphic types, makes type inference and type checking (without type annotations) undecidable [Wells, 1999, Pierce, 2002]. The Hindley–Milner type system for the lambda calculus is a restriction of System F that has a decidable inference algorithm called Damas–Milner [Damas and Milner, 1982], even without type annotations. However, it does not scale to more expressive systems, such as those with higher-rank polymorphism (functions that take polymorphic functions as arguments). This is why, for example, GHC, a Haskell compiler, has moved beyond Hindley–Damas–Milner (toward bidirectional typing) even though it serves as GHC's foundation; OCaml's type system has also done so for features such as so-called generalized algebraic data types (GADTs). Extending Hindley–Milner with datasort refinements retains decidability of type inference; however, type inference for index refinements in DML is undecidable (but its bidirectional typing is decidable, assuming the index constraints it generates are decidable, but this is not guaranteed in general) [Dunfield, 2007b]. Dependent type inference is undecidable [Dowek, 1993] but again annotations can be used to obtain decidability.

Bidirectional typing As we enrich our type systems, we can manage the resulting increase in complexity by systematically combining both checking and synthesis/inference (or input and output) modes: *bidirectional typing* [Pierce and Turner, 2000]. Bidirectional typing is a popular way to implement a wide variety of systems, including dependent types [Coquand, 1996, Abel et al., 2008], contextual types [Pientka, 2008], and object-oriented languages [Odersky et al., 2001]. Decidable and practical typing for higher-rank polymorphism was first achieved via bidirectional typing [Peyton Jones et al., 2007]. Modern presentations (à la ours, or vice versa really, though ours is not fundamentally about higher-rank polymorphism and unfortunately lacks polymorphism altogether) frame it more logically [Dunfield and Krishnaswami, 2013], and include extensions to GADTs [Dunfield and Krishnaswami, 2019]. The price programmers pay is the obligation to provide more type annotations. Fortunately, in practice, few annotations seem to be required, and anyway bidirectional typing clarifies where annotations are needed: for example, Dunfield and Krishnaswami's type system [Dunfield and Krishnaswami, 2013] only requires annotations for polymorphic and reducible expressions. Further, annotations are machine-checked documentation.

Dunfield and Krishnaswami [2013] also present a bidirectional type system for higher-rank polymorphism, but framed more proof theoretically; Dunfield and Krishnaswami [2019] extend it to a richer language with existentials, indexed types, sums, products, equations over type variables, pattern matching, polarized subtyping, and principality tracking. The algorithmic system of this thesis adapts some key techniques from these papers: instead of conjuring indices we introduce existential variables to the input context and output a context with solutions leading to the extension of the context with more information. The bidirectional system of this thesis uses logical techniques similar to these systems but it

does not consider polymorphism. However, whoever (maybe me) extends the system with polymorphism in future work may want to consult these two papers.

Bidirectional typing is helpful not only in terms of scalability, but also in improving the quality of error messages (by localizing them). Bidirectional type systems are also relatively easy to implement, if their typing rules are mode-correct: a rule is *mode-correct* if the input metavariables of its premises are known (based on the conclusion's inputs and outputs of previous premises), and if the conclusion's outputs are known when all premises are derived. The design principles of bidirectional typing are explored in a recent survey [Dunfield and Krishnaswami, 2021].

The survey paper by Dunfield and Krishnaswami [2021] discusses bidirectional typing's connections to proof theory. Basically, good bidirectional systems tend to distinguish checking and synthesizing terms or proofs according to their form, such as normal or neutral. In natural deduction, following the Pfenning recipe described in the survey tends to lead to a sensible bidirectional system. Rather than natural deduction, we begin (Chapter 4) with sequent calculus which only has introduction rules. To implement a typing judgment in a sequent calculus it is natural to begin with the goal (conclusion) and works upward (toward premises) relative to the presentation of inference rules (premises above, conclusion below). If a metavariable occurs in premises of a rule but not in its conclusion then its mode in an implementation must be an output (in our system, this only happens at cut rules), otherwise it is an input. Though we do not do this in the thesis, we can then add more inference to the system; the point of this thesis is to find a good starting point.

In our system, we distinguish judgments as checking or synthesizing by replacing the usual colon (:) in a type assignment judgment by $\Leftarrow$ (checking) or $\Rightarrow$ (synthesizing). In other judgments, sometimes other symbols indicate that a metavariable is an output: the

metavariables on the right of a $\dashv$ or $/$ or $\stackrel{\circ}{=}$ of a judgment, or in rightmost square brackets $[-]$ (like the value-determined dependencies ξ of type well-formedness $\Xi \vdash A\ \mathsf{type}[\xi]$), are always outputs.

3.2 Dependent Typing and Refinement Typing

From the perspective of a functional programmer, to a first approximation, dependent typing and refinement typing might seem to be almost the same thing. However, whereas dependent typing seems to begin with high expressivity and then attempts to recover automation, refinement typing seems to begin with high automation and then attempts to recover expressivity.

Dependent types, historically preceding index refinements, were introduced by Martin-Löf [1971, 1975] and first used to define a programming language by Martin-Löf [1979]. Mechanical proof assistants such as Agda [Norell, 2008], Rocq [Bertot and Castéran, 2013], and Lean [de Moura et al., 2015] rely on dependent types. Dependent types allow one to index types by general program terms (not just indices written in a more restrictive language to preserve properties of the unrefined type system); in refinement types, by contrast, the index domain (and ideally, in the liquid style, the constraints mentioning them) is deliberately restricted to preserve desirable properties of the underlying system such as decidability (of type inference in ML, for example), enabling automatic and decidable type inference/checking. Dependent types hence tend to be much more expressive than refinement types: the starting point is *full spectrum dependency* [Angiuli and Gratzer, 2024]. However, type checking and inference in dependent type systems is undecidable in general. In a fully dependent setting, one must manually provide proofs for things that cannot be automated which is more likely to arise given its relatively relaxed language. In refinement

type systems, by contrast, we usually begin with a less expressive decidable type system (like ML) and then enrich the types with refinements in such a way that programmers do not need to refactor their code or provide manual proofs for their original programs to type-check. This approach seems to make it easier to prove results but it is much less expressive: the structure of the underlying type of a refinement type cannot vary with its indices.

Considering an example from Angiuli and Gratzer [2024], in Agda, a dependently typed functional language and proof assistant, we can define the type

$$\begin{aligned}&\mathsf{nary} : \mathsf{Type} \to \mathsf{Nat} \to \mathsf{Type}\\ &\mathsf{nary}\ A\ 0 = A\\ &\mathsf{nary}\ A\ (\mathsf{suc}\ n) = A \to \mathsf{nary}\ A\ n\end{aligned}$$

and the function

$$\begin{aligned}&\mathsf{apply} : \{A : \mathsf{Type}\}\ \{n : \mathsf{Nat}\} \to \mathsf{nary}\ A\ n \to \mathsf{Vec}\ A\ n \to A\\ &\mathsf{apply}\ x\ [] = x\\ &\mathsf{apply}\ f\ (x :: xs) = \mathsf{apply}\ (f\ x)\ xs\end{aligned}$$

where $\mathsf{Vec}\ A\ n$ is a type[6] of lists of length n (specified similarly to $\mathsf{List}\ A\ n$ in Chapter 1) but the dependent type (here, $\{-\}$ indicates implicit universal quantification)

$$\{A : \mathsf{Type}\}\ \{n : \mathsf{Nat}\} \to \mathsf{nary}\ A\ n \to \mathsf{Vec}\ A\ n \to A$$

is not a refinement of any (simple[7]) unrefined type because $\mathsf{nary}\ A\ n$ is not a function type at $n = 0$ but is a function type at $n > 0$.

Semantically, refinement type systems seem to differ from dependent type systems mainly in that they refine a pre-existing type system, so that erasure of refinements always preserves typing of programs.

[6] In this thesis we write $\mathsf{List}\ A\ n$ but using Vec is common.

[7] Noam Zeilberger informs me that it should be possible to have these kinds of examples in a type refinement system with a top/universal type.

More on dependent types and systems using them Many dependent type systems impose their own restrictions for the sake of decidability. In Cayenne [Augustsson, 1998], typing can only proceed a given number of steps. All well-typed programs in Epigram [McBride and McKinna, 2004] are required to terminate so that its type equivalence is decidable. Epigram, and other systems [Chen and Xi, 2005, Licata and Harper, 2005], allow programmers to write explicit proofs of type equivalence.

Systems like ATS [Xi, 2004] and F* [Swamy et al., 2016] can be thought of as combining refinement and dependent types. These systems aim to bring the best of both refinement and dependent types, but ATS seems more geared to practical, effectful functional programming (hence refinement types), while F* is more geared to formal verification and dependent types. Unlike our system, they allow the programmer to provide proofs. The overall design of ATS is closer to our system than that of F*, due to its phase distinction between statics and dynamics; but it allows the programmer to write (in the language itself) proofs in order to simplify or eliminate constraints for the (external) constraint solver: Xi calls this *internalized constraint solving*. (It should be possible to internalize constraint solving to some extent in our system, but we don't try to do so in this thesis.) Liquid Haskell has a similar mechanism called *refinement reflection* [Vazou et al., 2017] in which programmers can write manual proofs (in Haskell) in cases where automatic Proof by Logical Evaluation and SMT solving fail.

Both ATS and F* have a CBV semantics, which is inherently monadic [Moggi, 1989b]. Our system is a variant of CBPV, which subsumes both CBV and CBN, and is also inherently monadic. These systems consider effects other than divergence, like exceptions, mutable state and input/output, which we hope to add to our system in future work; this should go relatively smoothly precisely because CBPV is monadic. The system F* allows

for termination metrics other than strong induction on natural numbers, such as lexicographic induction, but we think it would be straightforward to add such metrics to our system (which only has induction on natural numbers), in the way discussed in Sec. 6.5.

Refinement typing Although a rich type system such as Haskell's can prevent a large number of runtime errors, it can't catch them all. A division by zero can cause a runtime error, even though its arguments have integer type. An array index can have integer type, but cause an out-of-bounds error. A sorting function may output a list, but a classical type system cannot statically guarantee that it outputs sorted lists. Refinement types, however, can express these more precise program properties; a refinement type system can check these without making the programmer do much more work than otherwise (besides specifying the refinements in type annotations).

It seems Constable [1983] was first to introduce the basic idea of refinement types (though not by that name) in the sense of logical subsets of types, writing `{x:A|B}` for the *subset type* classifying terms `x` of type `A` that satisfy proposition `B`. However, we cannot really call these refinement types in the sense intended by this thesis because type checking with arbitrary `B` is undecidable and impractical. Freeman and Pfenning [1991] introduced refinement types (in a sense aligning with this thesis) to the programming language Standard ML via *datasort* refinements—inclusion hierarchies of ML-style (algebraic, inductive) datatypes—and intersection types for Standard ML: they showed that full type inference is decidable under a refinement restriction, and provided an algorithm based on abstract interpretation. The dangerous interaction of datasort refinements, intersection types, side effects, and call-by-value evaluation was first dealt with by Davies and Pfenning [2000] by way of a value restriction on intersection introduction; they also presented a bidirectional typing algorithm.

Dependent ML (DML) [Xi, 1998, Xi and Pfenning, 1999] extended the ML type discipline parametrically by *index* domains. DML is only decidable modulo decidability of index constraint satisfiability. DML uses a bidirectional type system with index refinements for a variant of ML, capable of checking properties ranging from in-bound array access [Xi and Pfenning, 1998] to program termination [Xi, 2002]. DML, similarly to our system, collects constraints from the program and passes them to a constraint solver, but does not guarantee that they are SMT solvable (unlike our system). This is also the approach of systems like Stardust [Dunfield, 2007a] (which combines both datasort and index refinements, and supports index refinements that are not value-determined, that is, *invaluable refinements*, which we do not consider) and those with liquid types [Rondon et al., 2008]. Liquid Haskell and Dependent ML are based on a Hindley–Milner approach; typically, Hindley–Damas–Milner inference algorithms [Hindley, 1969, Milner, 1978, Damas and Milner, 1982] generate typing constraints to be verified [Heeren et al., 2002]. We expect that index refinements subsume datasort refinements but have not checked this rigorously.

One application of DML was to eliminate statically via types manual array bounds and list tag checking [Xi and Pfenning, 1998]. Deviating slightly from the original notation and terminology, with index refinement types, we can write the type of get as follows.

$\texttt{get} \;:\; \forall n : \mathbb{N}.\, \forall l : \mathbb{N}.\, (n < l) \supset \mathsf{Nat}\, n \to \mathsf{List}\, A\, l \to A$

For a constraint φ and type A, the *guarded* type $\varphi \supset A$ reads "φ implies A", and is equivalent to A if φ is true, and otherwise represents unusable programs. As such, a successful call to get cannot be out of bounds (assuming hardware does not malfunction and the implementation is correct). We may regard the constraint $n < l$ as a *precondition* à la Hoare logic [Hoare, 1969]. Similarly, we can statically encode *postconditions*:

$\texttt{abs} \;:\; \forall n : \mathbb{Z}.\, \mathsf{Int}\, n \to \exists m : \mathbb{Z}.\, (\mathsf{Int}\, m) \wedge (m \geq 0) \wedge (m = n \vee m = 0 - n)$

```
abs n = if n >= 0 then n else 0 - n
```

specifies that abs returns a natural number of magnitude equal to its input. A value of *asserting* type $P \wedge \varphi$ has type P and also satisfies φ.

Liquid types are another way of representing refinement types. A liquid type is basically a base type together with a predicate on it, expressed like set comprehension: for example, the type $\{\nu : \mathsf{Nat} \mid \nu \leq 10\}$ represents values ν of base type Nat that are at most 10. In a liquid type system, templates can be provided by the programmer so that the type system can try to infer refinements (such as $\nu \leq 10$ above). In our system, we decompose this into the normal form $\{\nu : \mathsf{Nat} \mid \mathsf{index}\ \nu = a\} \wedge a \leq 10$ using a measurement $\mathsf{index}\ \nu = a$ and an assertion $a \leq 10$. Liquid types have been applied to programming languages such as OCaml [Kawaguchi et al., 2009], Haskell [Vazou et al., 2014], and C [Rondon et al., 2012]. A key novelty of liquid type systems is the ability to refine inductive types, such as lists, *modularly* by a restricted but practical class of inductive functions called *measures*: for example, one may express the type of lists of *increasing* integers. Such are called *recursive refinements* [Kawaguchi et al., 2009]. The main engine behind liquid types is a subtyping judgment that generates logical constraints whose validity are automatically decidable by a constraint solver, such as a satisfiability modulo theory (SMT) solver like Z3 [de Moura and Bjørner, 2008].

Algebraic data types and measures A novelty of liquid typing is the use of *measures*: functions, defined on algebraic data types, which may be structurally recursive, but are guaranteed to terminate and can therefore safely be used to refine the inductive types over which they are defined.

For example, consider the type $\mathsf{BinTree}\,A$ of binary trees with terms of type A at leaves:

```
data BinTree Type where
  leaf : A → BinTree A
  node : BinTree A → BinTree A → BinTree A
```

Suppose we want to refine $\mathsf{BinTree}\,A$ by the height of trees. Perhaps the most direct way to specify this is to measure it using a function hgt defined by structural recursion:

$$\begin{aligned}&\mathsf{hgt} : \mathsf{BinTree}\,A \to \mathbb{N}\\ &\mathsf{hgt}\ \mathsf{leaf} = 0\\ &\mathsf{hgt}\ (\mathsf{node}\ t\ u) = 1 + \mathsf{max}(\mathsf{hgt}(t), \mathsf{hgt}(u))\end{aligned}$$

As another example, consider an inductive type Expr of expressions in a CBV lambda calculus:

$$\begin{aligned}&\texttt{data}\ \mathsf{Expr}\ \texttt{where}\\ &\quad \mathsf{var} : \mathsf{Nat} \to \mathsf{Expr}\\ &\quad \mathsf{lam} : \mathsf{Nat} \to \mathsf{Expr} \to \mathsf{Expr}\\ &\quad \mathsf{app} : \mathsf{Expr} \to \mathsf{Expr} \to \mathsf{Expr}\end{aligned}$$

Measures need not use recursive calls. For example, if we want to refine the type Expr to expressions Expr that are values (in the sense of CBV, not CBPV), then we can use isval:

$$\begin{aligned}&\mathsf{isval} : \mathsf{Expr} \to \mathbb{B}\\ &\mathsf{isval}\ (\mathsf{var}\ z) = \mathsf{tt}\\ &\mathsf{isval}\ (\mathsf{lam}\ z\ \mathit{expr}) = \mathsf{tt}\\ &\mathsf{isval}\ (\mathsf{app}\ \mathit{expr}\ \mathit{expr}') = \mathsf{ff}\end{aligned}$$

Because isval isn't recursive and returns indexes, it's safe to use it to refine Expr to expressions that are CBV values: $\{v : \mathsf{Expr} \mid \mathsf{isval}\ v = \mathsf{tt}\}$. But, as with len (Chapter 1), we may again be worried about using the seemingly dynamic, recursively defined hgt in a static refinement type. Again, we need not worry because hgt, like len, is a terminating function into a decidable logic [Barrett et al., 2009]. We can use it to specify that, say, a height function defined by pattern matching on trees of type $\{v : \mathsf{BinTree}\,A \mid \mathsf{hgt}\ v = n\}$ actually returns (the program value representing) n for any tree of height n. Given the phase distinction between indexes (like n) and programs, how do we specify such a function type? In this thesis we use refinement type unrolling and singletons.

Unrolling and singletons Let's consider a slightly simpler function, length, that takes a list and returns its length:

$$\mathsf{length}\ [] = \mathsf{zero}$$
$$\mathsf{length}\ (x :: xs) = \mathsf{suc}\ (\mathsf{length}\ xs)$$

What should be the type specifying that length actually returns a list's length? The proposal $\forall n : \mathbb{N}.\, \mathsf{List}(A)(n) \to \uparrow \mathsf{Nat}$ does not work because Nat has no information about the length n. Something like $\forall n : \mathbb{N}.\, \mathsf{List}(A)(n) \to \uparrow (n : \mathsf{Nat})$, read literally as returning the *index* n (and not the *program value* corresponding to it), would violate our phase distinction between programs and indices. Instead, we use a *singleton* type in the sense of Xi [1998]: a singleton type contains just those program terms (of the type's erasure), that correspond to exactly one semantic index. For example, given $n : \mathbb{N}$, we define the singleton type $\mathsf{Nat}(n)$ (which may also be written $\mathsf{Nat}\ n$) by $\{v : \mathsf{Nat} \mid \mathsf{index}\, v = n\}$ where

$$\mathsf{index} : \mathsf{Nat} \to \mathbb{N}$$
$$\mathsf{index}\ \mathsf{zero} = 0$$
$$\mathsf{index}\ (\mathsf{suc}\ x) = 1 + \mathsf{index}(x)$$

specifies the indexes (of sort $\mathbb{N}$) corresponding to program values of type Nat.

How do we check length against $\forall n : \mathbb{N}.\, \mathsf{List}(A)(n) \to \uparrow(\mathsf{Nat}(n))$? In the first clause, the input [] has type $\mathsf{List}(A)(n)$ for some n, but we need to know $n = 0$ (and that the index of zero is 0). Similarly, we need to know $x :: xs$ has length $n = 1 + n'$ where $n' : \mathbb{N}$ is the length of xs. To generate these constraints, we use an *unrolling* judgment (Sec. 6.4) that unrolls a refined inductive type. Unrolling puts the type's refinement constraints, expressed by *asserting* and *existential* types, in the structurally appropriate positions. An asserting type is written $Q \wedge \varphi$ (read "Q with φ"), where Q is a (positive) type and φ is an index proposition. If a term has type $Q \wedge \varphi$, then the term has type Q and also φ holds. (Dual to asserting types, we have the *guarded* type $\varphi \supset M$, which is equivalent to M if φ holds, but is otherwise useless.) We use asserting types to express that index equalities like $n = 0$ hold

for terms of inductive type. We use existentials to quantify over indexes that characterize the refinements of recursive subparts of inductive types, like n'. For example, modulo a small unimportant difference (see Sec. 6.4), $\mathsf{List}(A)(n)$ unrolls to

$$\big(1 \wedge (n = 0)\big) + \big(A \times \exists n' : \mathbb{N}.\ \big\{ v : \mathsf{List}\, A \;\big|\; \mathsf{len}\, v = n' \big\} \wedge (n = 1 + n')\big)$$

That is, to construct an A-list of length n, the programmer (or library designer) can either left-inject a unit value, *provided the constraint* $n = 0$ *holds*, or right-inject a pair of one A value and a tail list, *provided that* n'*, the length of the tail list, is* $n - 1$ (the equations $n = 1 + n'$ and $n - 1 = n'$ are equivalent): thus the output of unrolling corresponds to a usual `data` specification but without named constructors. These index constraints are not a syntactic part of the list itself. That is, a term of the above refined type is also a term of the type's *erasure* (of indexes):

$$1 + (|A| \times (\mathsf{List}\ |A|))$$

where $|-|$ erases indexes. Dual to *verifying* refined inductive values, pattern matching on refined inductive values, such as in the definition of length, allows us to *use* the index refinements locally in the bodies of the various clauses for different patterns. Liquid Haskell similarly extracts refinements of data constructors for use in pattern matching.

The shape of the refinement types generated by our unrolling judgment (such as the one above) is a judgmental and refined-ADT version of the fact that generalized ADTs (GADTs) can be desugared into types with equalities and existentials that express constraints of the return types for constructors [Cheney and Hinze, 2003, Xi et al., 2003]. It would be tedious and inefficient for the programmer to work directly with terms of types

produced by our unrolling judgment, but we can implement (in our system) singleton refinements of base types and common functions on them, such as addition, subtraction, multiplication, division, and the modulo operation on integers, and build these into the surface language used by the programmer, similarly to the implementation of Dependent ML [Xi, 1998].

More on liquid types Rondon et al. [2008] introduced *logically qualified data types*, that is, *liquid types*, in a system that extends Hindley–Milner to infer (by abstract interpretation) refinements based on built-in or programmer-provided refinement templates or qualifiers. Kawaguchi et al. [2009] introduced *recursive refinement* via sound and terminating *measures* on algebraic data types; they also introduced *polymorphic refinement*. Vazou et al. [2013] generalize recursive and polymorphic refinement into a single, more expressive mechanism: *abstract refinement*. Our system sadly lacks polymorphism, which we plan to study in future work; we anticipate similarities to past work in a similar vein [Dunfield and Krishnaswami, 2013] but with design issues peculiar to polymorphism and other liquid typing features like abstract refinements. Abstract refinements would of course go well with polymorphism (which we lack currently) but also the multi-argument measures of this thesis because abstract refinements may be thought of as predicates of higher-order sort (higher-order as in, sorts involving arrow sorts) and we encode multi-argument measures using higher-order sorts. In future work, it would be interesting to study other features of liquid typing in our setting. Extending our system with liquid inference of refinements, for example, would require adding a Hindley–Milner type inference of refinement templates, as well as mechanisms to solve these templates, possibly in an initial phase using abstract interpretation.

Unlike DML (and our system), liquid type systems do not distinguish index terms from

programs, at least not explicitly. Whilst this provides simplicity and convenience to the user (from their perspective, there is just one language), it makes it relatively difficult to provide liquid type systems a denotational semantics and to prove soundness results denotationally (rather than operationally), in contrast to our system (we should be able to recover some of this convenience, by requiring users, for example, to make measure annotations like Liquid Haskell). It creates other subtleties such as the fact that annotations for termination metrics in Liquid Haskell must be internally translated to ghost parameters. By contrast, if we extend our system with additional termination metrics, because these metrics are at the index level, we should have no need for such ghost parameters in an implementation. As we will see in an example soon, liquid types' lack of index distinction also makes it trickier to support computational effects and evaluation orders.

Initial work on liquid types [Rondon et al., 2008, Kawaguchi et al., 2009] used call-by-value languages, but Haskell uses lazy evaluation so Liquid Haskell was discovered to be unsound [Vazou et al., 2014]. Vazou et al. [2014] regained typing soundness by imposing operational-semantic restrictions on subtyping and let-binding. In their algorithmic subtyping, there is exactly one rule, $\preceq$-BASE-D, which pertains to refinements of base types (integers, booleans and so on) and inductive data types; however, these types have a *well-formedness restriction*, namely, that the refinement predicates have the type of boolean expressions that reduce to finite values. But this restriction alone does not suffice for soundness under laziness and divergence. As such, their algorithmic typing rule T-CASE-D, which combines let-binding and pattern matching, uses an operational semantics to approximate whether or not the bound expression terminates. If the bound expression might diverge, then so might the entire case expression; otherwise, it checks each branch in a context that assumes the expression reduces to a (potentially infinite Haskell) value.

We also have a type well-formedness restriction, but it is purely syntactic, and only on index quantification, requiring it to be associated with index folds that are necessarily decidable by virtue of a systematic phase distinction between the index level and the program level. Further, via type polarization, our let-binding rule requires the bound expression to return a value, we only allow value types in our program contexts, and we cannot extract index information (which happens in inversion stages) across polarity shifts (such as in a suspended computation). Therefore, in our system, there is no need to stratify our types according to an approximate criterion; rather, we exploit the systemic distinction between positive (value) types and negative (computation) types, that Levy [2004] designed or discovered to be semantically well-behaved. We suspect that liquid types' divergence-based stratification is indirectly grappling with logical polarity. Because divergence-based stratification is peculiar to the specific effect of nontermination, it is unclear how their approach may extend to other effects. By way of a standard embedding of CBN or CBV into (our focalized variant of) CBPV we can obtain CBN or CBV subtyping and typing relations automatically respecting any necessary value or covalue restrictions [Zeilberger, 2009]. Further, being a CBPV, our system is already in a good position to handle the addition of effects other than nontermination.

Dynamic typing and contract calculi Software *contracts* express program properties in the same language as the programs themselves; Findler and Felleisen [2002] introduced contracts for *run-time* verification of higher-order functional programs. These *latent* contracts are not types, but *manifest* contracts are [Greenberg et al., 2010]. Manifest contracts are akin to refinement types. Indeed, Vazou et al. [2013] sketch a proof of typing soundness for a liquid type system by translation from liquid types to the manifest contract calculus F_H of Belo et al. [2011]. However, there is no explicit translation back, from F_H to liquid

typing. They mention that the translated terms in F_H do not have *upcasts* because the latter in F_H are logically related to identity functions if they correspond to static subtyping (as they do in the liquid type system): an upcast lemma. Presumably, this facilitates a translation from F_H back to liquid types. However, there are technical problems in F_H that break typing soundness and the logical consistency of the F_H contract system; Sekiyama et al. [2017] fix these issues, resulting in the system F_H^σ, but do not consider subtyping and subsumption, and do not prove an upcast lemma. However, it looks like they do do these things in the sequel [Sekiyama and Igarashi, 2018]. It would be interesting to consider gradual typing in relation to all these topics, but dynamic/gradual typing is outside the scope of the thesis.

Refinement typing, evaluation strategy, and computational effects The interaction between refinement typing (and other fancy typing), evaluation strategy, and computational effects can be problematic. The combination of parametric polymorphism with effects in ML was at first unsound [Harper and Lillibridge, 1991]; the value restriction in Standard ML recovers soundness in the presence of mutable references by restricting polymorphic generalization to syntactic values [Wright, 1995]. The issue was also not peculiar to polymorphism: Davies and Pfenning [2000] discovered that a similar value restriction recovers typing soundness for intersection refinement types and effects in call-by-value languages. For union types, Dunfield and Pfenning [2003] obtained soundness by an evaluation context restriction on union elimination.

For similar reasons, Liquid Haskell was also found unsound in practice, and had to be patched; we adapt an example taken from Vazou et al. [2014] demonstrating the discovered unsoundness:

```
diverge :: Nat -> {v:Int | false}
```

```
diverge x = diverge x


safediv :: n:Nat -> {d:Nat | 0 < d} -> {v:Nat | v <= n}
safediv n d = if 0 < d then n / d else error "unreachable"


unsafe :: Nat -> Int
unsafe x = let notused = diverge 1 in let y = 0 in safediv x y
```

In typechecking `unsafe`, we need to check that the type of `y` (a singleton type of one value: 0) is a subtype of `safediv`'s second argument type (under the context of the let-binding). Due to the refinement of the let-bound `notused`, this subtyping generates a *constraint* or *verification condition* of the form "if false is true, then…". This constraint holds vacuously, implying that `unsafe` is safe. But `unsafe` really is unsafe because Haskell evaluates *lazily*: since `notused` is not used, `diverge` is never called, and hence `safediv` divides by zero (and crashes if uncaught). Vazou et al. [2014] recover typing soundness and decidable typing by restricting let-binding and subtyping, using an operational semantics to approximate whether or not expressions diverge, and whether or not terminating terms terminate to a *finite* value.

The value and evaluation context restrictions seem like ad hoc ways to cope with the failure of simple typing rules to deal with the interactions between effects and evaluation strategy. However, Zeilberger [2009] explains these restrictions in terms of a logical view of refinement typing. Not only does this perspective explain these restrictions, it provides theoretical tools for designing type systems for functional languages with effects. At the heart of Zeilberger's approach is the proof-theoretic technique of *focusing*, which we discuss further near the end of this chapter. An important question we address is whether polarization

and focusing can also help us understand Liquid Haskell's restrictions on let-binding and subtyping: basically, our let-binding rule requires the bound computation (negative type) to terminate to a value (positive type). In other words, focalized systems satisfy any necessary value (and covalue) restrictions by default.

Focalization can also yield systems with good semantic properties under computational effects, in particular, variants of call-by-push-value.

Refining call-by-push-value The type of say abs (absolute value), namely

$$\forall n : \mathbb{Z}.\ \mathsf{Int}\ n \to \exists m : \mathbb{Z}.\ (\mathsf{Int}\ m) \wedge (m \geq 0) \wedge (m = n \vee m = 0 - n)$$

can be viewed logically as a proposition that, for every integer n, there exists a nonnegative integer m equal in magnitude to n. The program abs, which constructs such an m for any n, can be seen as a proof of this proposition. The Curry–Howard (CH) correspondence, as originally conceived [Howard, 1980], is an isomorphism between natural deduction (for intuitionistic propositional logic) and the typed lambda calculus, linking logic and programming. But there are other bridges between logic and programming than this specific correspondence; this phenomenon is often called "the" CH(L) correspondence (L is for Lambek who extended CH to categories). The rough idea is that propositions correspond to types (and objects), and proofs correspond to programs (and morphisms).

The CHL correspondence is technically complicated by computational effects, such as divergence and writing to a store. Effectful program behavior is sensitive to evaluation strategy. The strategy *call-by-value* (CBV) evaluates expressions before binding them to variables, whereas *call-by-name* (CBN) evaluates expressions after binding them. For

example, in CBV function application, function arguments are first evaluated, and the resulting value is substituted into the function body, whereas in CBN, the unevaluated expression is substituted into the function body and evaluated as it occurs in the result (possibly in multiple places). The calculus *call-by-push-value* (CBPV) [Levy, 2004] unifies CBN and CBV semantics at the level of types. It syntactically polarizes types into computation types (for program terms "which *do*") and value types (for program terms "which *are*"[8]), and provides connectives for transitioning between them (by "thunking" a computation, or "lifting" a value). This allows us to use types to reason about effects under different evaluation strategies.

Technically, *core* CBPV (without recursion hence without divergence) does not include effects. However, CBPV behaves well when we add effects. In CBPV, even with effects, we need only prove one semantic metatheory to get both CBV and CBN metatheory for free. CBPV subsumes CBV and CBN in the sense that one can translate a CBV program to CBPV and a CBN program to CBPV, resulting in different programs that behave as CBV or CBN in the CBPV semantics: evaluating a translated CBV (respectively, CBN) program coincides with evaluating the original program with CBV (respectively, CBN).

Call-by-push-value subsumes both call-by-value and call-by-name by polarizing the usual terms and types of the λ-calculus into a finer structure that can be used to encode both evaluation strategies in a way that can accommodate computational effects: *value* (or *positive*) types (classifying terms which "are", that is, values v), *computation* (or *negative*) types (classifying terms which "do", that is, expressions e), and polarity shifts $\uparrow -$ and $\downarrow -$ between them. An upshift $\uparrow P$ *lifts* a (positive) value type P up to a (negative) computation type of expressions that compute values (of type P). A downshift $\downarrow N$ pushes a (negative) computation type N down into a (positive) value type of

[8]The CBPV slogans "computations *do*" and "values *are*" are from Paul Blain Levy.

thunked or *suspended* computations (of type N). We can embed the usual λ-calculus function type $A \to_\lambda B$ (written here with a subscript to distinguish it from the CBPV function type), for example, into CBPV (whose function types have the form $P \to N$ for positive P and negative N) so that it behaves like CBV, via the translation ι_{CBV} with $\iota_{\mathrm{CBV}}(A \to_\lambda B) = \downarrow(\iota_{\mathrm{CBV}}(A) \to \uparrow \iota_{\mathrm{CBV}}(B))$; or so that it behaves like CBN, via the translation ι_{CBN} with $\iota_{\mathrm{CBN}}(A \to_\lambda B) = (\downarrow \iota_{\mathrm{CBN}}(A)) \to \iota_{\mathrm{CBN}}(B)$.

Evaluation order is made explicit by CBPV type discipline. Therefore, adding a refinement layer on top of CBPV requires directly and systematically dealing with the interaction between refinement types and evaluation order. If we add this layer to CBPV correctly from the very beginning, then we can be confident that our refinement type system will be semantically well-behaved when extended with other computational effects. The semantics of CBPV are well-studied and this helps us establish semantic metatheory. In later parts of this overview, we show the practical effects of refining our focalized variant of CBPV, especially when it comes to algorithmic matters.

Typing soundness, totality, and logical consistency The unrefined system (Ch. 5) underlying our refined system (Ch. 6) has the computational effect of nontermination and hence is not total. To model nontermination, we give the unrefined system a standard CBPV domain-theoretic denotational semantics: value types denote predomains (which do not necessarily have a bottom element representing divergence) and computation types denote domains (which necessarily have a bottom element in the complete partial order, expressing when the denotation is least defined due to computational divergence). Semantic typing soundness says that a syntactic typing derivation can be faithfully interpreted

as a semantic typing derivation, that is, a morphism in a category[9], in this case a logical refinement of domains. Semantic typing soundness basically corresponds to syntactic typing soundness with respect to a big-step operational semantics. Our operational semantics is based on an extension of cut elimination in the unrefined system without recursion and ADTs and it is equivalent to the denotational semantics. Part of that equivalence is called computational adequacy: if a program denotes divergence then it really does diverge when run on an (abstract) machine. (It seems the first adequacy result appears, though not by that name, in the work of Plotkin [1977].) Because the unrefined system is (a focalized variant of) CBPV, proving typing soundness is relatively straightforward.

In contrast to dependent types, the denotational semantics of our refined system is defined in terms of that of its *erasure* (of indexes), that is, its underlying, unrefined system. A refined type denotes a logical subset of what its erasure denotes. An *unrefined* return type $\uparrow P$ denotes either what P denotes, or divergence/nontermination. A *refined* return type $\uparrow P$ denotes *only* what P denotes. Therefore, our *refined* typing soundness result together with computational adequacy implies that our refined system (without a *partial* upshift type) enforces termination, both denotationally and operationally. At the end of Ch. 7 we discuss how to extend the refined system (by a partial upshift type) to permit divergence while keeping typing soundness (which implies partial correctness for partial upshifts). Typing soundness also implies logical consistency, because a logically inconsistent refinement type denotes the empty set.

[9] A *category* is a collection of objects and morphisms or arrows between them where every object has an identity morphism and pairs of composable morphisms compose associatively into some morphism. A category can be viewed as a mathematical universe of discourse. In this thesis we do not use category theory very heavily; it is mainly an organizational tool.

Data abstraction and category theory In categorical semantics, (extensional)[10] inductive types are initial algebras of endofunctors[11]. We only consider certain *polynomial* endofunctors, which denote specifications of tree-shaped or algebraic data structures. A polynomial endofunctor F has an initial algebra $F(\mu F) \to \mu F$ whose carrier μF corresponds to an algebraic data type. Lambek's lemma tells us that μF is isomorphic to its unrolling $F(\mu F)$. The universal property called initiality is the recursion principle for μF: any algebra $F(X) \to X$ uniquely defines, in our intuitive set-and-functions-based setting, a recursive function $\mu F \to X$: if X is an index sort and F is a polynomial endofunctor on the category of sets and functions then we have defined a (semantic) measure on the ADT μF. We provide a semantically sound syntax for measures in this way. The power of abstraction lets us fit this all in one paragraph; in Ch. 7, we discuss these semantic issues in greater detail.

Our rolled refinement types refine *type* constructors μF. Sekiyama et al. [2015], again in work on manifest contracts, compare this to refining (types of) *data* constructors, and provide a translation from type constructor to data constructor refinements. According to Sekiyama et al. [2015] type constructor refinements (such as our $\{v : \mu F \mid \mathscr{M}\}$ where $\mathscr{M}$ is a list of measurements on v) are *easier for the programmer to specify*[12], but data constructor refinements (such as the output types of our unrolling judgment or the DML style definitions by named data constructors) are *easier to verify automatically*. Sekiyama et al. [2015] say that their translation from type to data constructor refinements is closely related to the work of Atkey et al. [2012] on refining inductive data in (a fibrational interpretation of) dependently typed languages. Atkey et al. [2012] provide "explicit formulas" computing inductive characterizations of type constructor refinements. These semantic formulas

[10] This is not relevant for this thesis because we do not consider identity types.

[11] A *functor* F from category $\mathbf{C}$ to category $\mathbf{D}$ sends objects to objects and morphisms to morphisms such that identity morphisms and composition of morphisms are preserved. If $\mathbf{C} = \mathbf{D}$ then F is an *endo*functor.

[12] They are modular.

resemble our syntactic unrolling judgment, which may be viewed as a translation from measure refinements to data constructor refinements.

Ornaments [McBride, 2011] describe how inductive types with different logical or ornamental properties can be systematically related using their algebraic and structural commonalities. Practical work in ornaments seems mostly geared toward code reuse [Dagand and McBride, 2012], code refactoring [Williams and Rémy, 2017] and such. In contrast, this thesis focuses on incorporating similar ideas in a foundational index refinement typing algorithm.

Melliès and Zeilberger [2015] provide a categorical theory of type refinement[13] in general, where functors are considered to be type refinement systems. This framework is based on Reynolds's distinction between *intrinsic* (or Church) and *extrinsic* (or Curry) views of typing [Reynolds, 1998]. We think that our system fits into this framework. This is most readily seen in the fact that the semantics of our refined system is simply the semantics (*intrinsic* to unrefined typing derivations) of its erasure of indexes, which express *extrinsic* properties of (erased) programs.

Inference, subtyping, and let-normality For a typed functional language to be practical, it must support some degree of inference, especially for function application (to eliminate universal types) and constructing values (to introduce existential types). To pass a value to a function, its type must be compatible with the function's argument type, but it would

[13] I have been writing "refinement type" and only now wrote "type refinement". I more or less use these terms interchangeably in this thesis. Similarly for "refinement type system" and "refinement typing". (In other settings, it may be apt to distinguish a refinement type as referring to syntax from type refinement as a research program started in the early 1990s.) Similarly, it would appear that only (the nomina sacra corresponding to) "Jesus Christ" was used in the canonical gospels, but (the nomina sacra corresponding to) "Christ Jesus" was used in the epistles as well [Linssen, 2023].

be burdensome to make programmers always have to prove this compatibility. In our setting, for example, if $x : \mathsf{Nat}(3)$ and $f : \downarrow(\forall a : \mathbb{N}.\, \mathsf{Nat}(a) \to \uparrow P)$, then we would prefer to write $f\ x$ rather than $f\ [3]\ x$, which would quickly make our programs incredibly—and unnecessarily—verbose.

Omitting index and type annotations, however, has significant implications. In particular, we need a mechanism to instantiate indexes somewhere in our typing rules: for example, if $g : \downarrow(\downarrow(\mathsf{Nat}(4+b) \to \uparrow P) \to N)$ and $h : \downarrow((\exists a : \mathbb{N}.\, \mathsf{Nat}(a)) \to \uparrow P)$, then to apply g to h, we need to know $\mathsf{Nat}(4+b)$ is compatible with $\exists a : \mathbb{N}.\, \mathsf{Nat}(a)$, which requires instantiating the bound a to a term logically equal to $4+b$. Our system does this kind of instantiation via subtyping. Indeed, type refinement naturally yields a subtyping relation between refinements of a common simple type: for example, any singleton type of natural number is a subtype of the natural numbers. Index instantiations are propagated locally across adjacent nodes in the syntax tree, similarly to Pierce and Turner [2000]. Liquid typing allows for more inference, including inference of refinements based on templates, which provides additional convenience for programmers, but we do not consider this kind of inference in this thesis.

We polarize subtyping into two, mutually recursive, positive and negative relations $\Theta \vdash P \leq^{+} Q$ and $\Theta \vdash N \leq^{-} M$ (where Θ is a *logical context* including index propositions). The algorithmic versions of these only introduce existential variables in positive supertypes and negative subtypes, guaranteeing they can always be solved by indexes without any existential variables. We delay checking constraints until the end of certain, logically designed stages (the *focusing* ones, as we will see), when all of their existential variables are guaranteed to be solvable based on all the static information obtained.

Our system requires intermediate computations like $h(s)$ to be explicitly named and

sequenced by let-binding (a kind of *A-normal* [Flanagan et al., 1993] or *let-normal form*). Combined with focusing, this allows us to use (within the value typechecking stage) subtyping only in the typing rule for (value) variables. This makes our subsumption rule syntax-directed, simplifying and increasing the efficiency of our algorithm. We nonetheless prove a general subsumption lemma, which is needed to prove that substitution respects typing, a crucial syntactic property.

Due to issues with existential index instantiation, the approach of Xi [1998] (incompletely) translated programs into a let-normal form [Sabry and Felleisen, 1993] before typing them, and Dunfield [2007b] provided a complete let-normal translation for similar issues. The system in this thesis is already let-normal due to being a sequent calculus.

Barendregt et al. [1983] discovered that a program that typechecks (in a system with intersection types) using subtyping, can also be checked without using subtyping, if the program is sufficiently η-expanded (eta-expanded). In a refinement typing interpretation, identity coercions can always be used to witness a subtyping relation, which requires η-expansion [Zeilberger, 2009]: this is the case for the refined system of this thesis.

3.3 Judgments, Their Presuppositions, and Value-Determined Indices

We more or less follow Martin-Löf [1996] in leaving judgmental presuppositions implicit. For example, in the unrefined system, a derivation of $\Gamma \vdash e \Leftarrow N$ presupposes a derivation of context well-formedness $\Gamma\ \mathsf{ctx}$ which limits program variables to being declared in it at most once (such a property cannot be enforced by a context-free grammar so we use inference rules). As another example, a derivation of $\Theta \vdash \varphi\ \mathsf{true}$ presupposes a derivation of the fact that φ has boolean sort $\mathbb{B}$ under $\overline{\Theta}$ where $\overline{}$ merely removes all the propositions occurring in Θ, that is, $\overline{\Theta} \vdash \varphi : \mathbb{B}$.

We only consider inference rules in which there are zero or more main premise judgments, zero or more side conditions (or secondary premises), and exactly one conclusion judgment. Metavariables in a judgment are presupposed to be generated by a grammar (which could be presented via inference rules, but that would be more verbose), or presupposed to be well-formed according to inference rules.

Presuppositions help us organize and understand our system(s) and reduce clutter. They also work well with the concept of value-determined indices which restrict the formation of types which is presupposed by typechecking for example.

Value-determined indexes and type well-formedness Like DML, we have an index-program distinction, but unlike DML and like Liquid Haskell, we want to guarantee SMT solvable constraints. We accomplish this with our technique of value-determined indexes. To guarantee that our algorithm can always instantiate quantifiers, we restrict quantification to indexes appearing in certain positions within types: namely, those that are uniquely determined (semantically speaking) by values of the type, according to measurements and assertions (of equality) made *before* crossing a polarity shift (which in this case marks the end of a focusing stage) and using information of value-determined indices in a previous polarity shift (those which have already been marked as being value-determined in the logical context).

For example, in $\{v : \mathsf{List}\ A \mid \mathsf{len}\ v = b\}$, the index b is uniquely determined by values of that type: the list $[x,y]$ uniquely determines b to be 2 (by the length measure). This value-determinedness restriction on quantification has served to explain why a similar restriction in the typing algorithm of Flux (Liquid Rust) seemed to work well in practice [Lehmann et al., 2023].

We make this restriction in the type well-formedness judgment, which outputs a set

ξ tracking value-determined index dependencies (which arise due to the use of equality assertions and higher-order sorts); well-formed types can only quantify over indexes that are value-determined according to ξ. For example, $\exists b : \mathbb{N}.\ \{v : \mathsf{List}\ A \mid \mathsf{len}\ v = b\}$ is well-formed. The dependencies in ξ also register type structure: for example, a value of a product type is a pair of values, where the first value has dependencies ξ_1 (for the first type component) and the second value has dependencies ξ_2 (second type component), so the ξ of the product type is their union $\xi_1 \cup \xi_2$. We also take the union for function types $R \to L$, because index information flows through argument types toward the return type, marked by a polarity shift.

By emptying ξ at shift types, we prevent lone existential variables from being introduced at a distance, across polarity shifts. In practice, this restriction on quantification is not onerous, because most functional types that programmers use are, in essence, of the form

$$\forall \Xi.\ \overrightarrow{\varphi} \supset \overrightarrow{R} \to \uparrow \exists \Xi'.\ R' \wedge \overrightarrow{\psi}$$

where the "$\forall$" quantifies over indexes Ξ of argument types $R_k \in \overrightarrow{R}$ and guards $\overrightarrow{\varphi}$ that are uniquely determined by argument values, and the "$\exists$" quantifies over indexes Ξ' of the return type with assertions $\overrightarrow{\psi}$ that are determined by fully applying the function and thereby constructing a value to return. The idea of this restriction was inspired by researchers of liquid types as they seem to follow it implicitly: variables appearing in liquid type refinements must ultimately come from arguments x to dependent functions $x : A \to B$ and their return values (however, these are not explicitly index variables).

Types that quantify only across polarity shifts tend to be empty, useless, or redundant. The ill-formed type $\forall n : \mathbb{N}.\ 1 \to \uparrow \mathsf{Nat}(n)$ is empty because no function returns a value representing any natural number when applied to unit. A term of ill-formed type

$\exists m : \mathbb{N}. \downarrow(\mathsf{Nat}(m) \to \uparrow\mathsf{Bool})$ can only be applied to an unknown number, which is useless because the number is unknown. The ill-formed type $\exists n : \mathbb{N}. \uparrow\downarrow\mathsf{Nat}(n)$ is redundant because it is semantically equivalent to $\downarrow\uparrow\exists n : \mathbb{N}. \mathsf{Nat}(n)$ (which does not quantify across a polarity shift), and similarly $\forall n : \mathbb{N}. \uparrow\downarrow(\mathsf{Nat}(n) \to \uparrow\mathsf{Nat}(n))$ is semantically equivalent to $\uparrow\downarrow(\forall n : \mathbb{N}. \mathsf{Nat}(n) \to \uparrow\mathsf{Nat}(n))$. Some refinements are *not* value-determined but useful nonetheless, such as dimension types [Kennedy, 1994, Dunfield, 2007b] which statically check that dimensions are used consistently (minutes can be added to minutes, but not to kilograms) but do not store the dimensions at run time. In this thesis, we do not consider these non-value-determined refinements, and Liquid Haskell does not support them either.

Our value-determinedness restriction on type well-formedness, together with focusing, is very helpful metatheoretically, because it means that our typing algorithm only introduces—and is guaranteed to solve—existential variables for indexes within certain logical stages. Consider checking say a list against the type $\exists b : \mathbb{N}. \{v : \mathsf{List}\, A \mid \mathsf{len}\, v = b\}$. An existential variable $\hat{b}$ for b is generated, and we check the unrolled list against the unrolling of $\left\{v : \mathsf{List}\, A \;\middle|\; \mathsf{len}\, v = \hat{b}\right\}$. A solution to $\hat{b}$ will be found within value typechecking (the right-focusing stage), using the invariant that no measure (such as len) contains any existential variables. Similarly, applying a function with universal quantifiers will solve all existential variables arising from these quantifiers by the end of a left-focusing stage, which typechecks an argument list of values.

Our examples in this discussion of value-determinedness so far have involved only (what we happen to call) *first-order* sorts. We use higher-order sorts (in which arrow sorts may occur) to allow the programmer to express recursive refinements relating different parts of an inductive datatype, such as elements in a list. An example is to refine lists of integers by whether they are ordered (or "sorted"; we want to avoid confusion with index

"sorts/sorting").

We use hereditary substitution [Watkins et al., 2004] (at the index level), which in addition to substituting reduces the result to a normal form, to maintain invariants in our system so that the constraints it generates are decidable by an SMT solver. Most importantly, in refined inductive types, equalities induced by measurements $(\mathsf{fold}_F\ \alpha)\, v\, \boldsymbol{u} = t$ must be SMT-decidable. But the equality of lambda terms (which are of higher-order sort) is not SMT-decidable, so we must fully reduce the application of recursive refinements to index spines $\boldsymbol{u}$ down to an index t of first-order sort. (A spine [Cervesato and Pfenning, 2003] is basically a leftward growing list but index spines include projections as well as index terms; a program spine s is a left growing list of values v so it may also be written $\overleftarrow{v}$, and a rightward growing list of values may be written $\overrightarrow{v}$. Similarly, we could write an index spine $\boldsymbol{u}$ as $\overleftarrow{u}$ but don't usually.)[14] This will be seen for example in the unrolling rule at unit functors, which uses what we call *hereditary application*, written $\langle - \mid - \rangle$, ultimately to output a normalized type asserting an equality of first-order indexes. If a value is checked against this type, an SMT solver should be able to decide the first-order equality.

Consider the refinement type IncrNatList of lists of natural numbers in (monotonically) increasing order, which is defined as follows:

$$\begin{aligned}
\mathsf{IncrNatList} &= \{v : \mu\mathsf{IncrNatListF} \mid (\mathsf{fold}\ \mathsf{incr})\, v\, 0 = \mathsf{tt}\} \\
\mathsf{IncrNatListF} &= I \oplus (\underline{\exists b : \mathbb{N}.\, \mathsf{Nat}(b)} \otimes \mathrm{Id}) \\
\mathsf{incr} &= () \Rightarrow \lambda c.\, \mathsf{tt} \mid (\underline{b}, a) \Rightarrow \lambda c.\, (b \geq c) \wedge a(b)
\end{aligned}$$

Note especially the use of the packed index variable b in the body of the algebra (we use syntactic sugar in this example: $\underline{b}$ is $\mathtt{pack}(b, \top)$ where $\top$ is the wild pattern for constant

[14] We also use the $\overleftarrow{-}$ and $\overrightarrow{-}$ notation for metavariables other than program values.

functors, in this case $\underline{\mathsf{Nat}(b)}$). Unrolling the type IncrNatList yields a type with right summand (corresponding to a list entry and tail list)

$$\exists b : \mathbb{N}.\, \mathsf{Nat}(b) \times \exists a : \mathbb{B}.\, \{v : \mu \mathsf{IncrNatListF} \mid (\mathsf{fold\ incr})\, v\, b = a\} \wedge \mathsf{tt} = ((b \geq 0) \wedge a)$$

The subterm $\mathsf{Nat}(b)$ has value-determined index b, and $\mathsf{Nat}(b)$ is part of a product, so the outer existential is well-formed. But why is the inner existential well-formed, that is, why is a value-determined? Consider the value $\langle 3, [5] \rangle$. The value 3 determines b is 3. The existential index a represents whether or not $[5]$ is in increasing order starting from b. Therefore, a is determined to be true, because b has been determined to be 3 and $5 \geq 3$. How does this work? As typechecking proceeds and another unrolling happens, we must check the value $\langle 5, [] \rangle$ against

$$\exists b' : \mathbb{N}.\, \mathsf{Nat}(b') \times \exists a' : \mathbb{B}.\, \left\{v : \mu \mathsf{IncrNatListF} \;\middle|\; (\mathsf{fold\ incr})\, v\, b' = a'\right\} \wedge a = ((b' \geq b) \wedge a')$$

The value 5 determines b' is 5 and the value $[]$ determines a' is tt. Plugging in what we know yields

$$\mathsf{Nat}(5) \times \{v : \mu \mathsf{IncrNatListF} \mid (\mathsf{fold\ incr})\, v\, 5 = \mathsf{tt}\} \wedge a = ((5 \geq 3) \wedge \mathsf{tt})$$

in which it is evident that a is true.

For sake of contrast, consider the following syntax with free index variable c of sort $\mathbb{N}$:

$$\exists b : \mathbb{B}.\, \{v : \mu \mathsf{IncrNatListF} \mid (\mathsf{fold\ incr})\, v\, c = b\}$$

This type is *not* well-formed. In particular, the binding for b is invalid: b is not value-determined because it depends on c, which can be anything (for example, the value $[3]$ determines b is true if $c \leq 3$ and false otherwise). However, *if* a value *were to* determine c, *then* the same value would determine b in turn, similarly to the example above. This semantic consideration implies we should restrict indices (like c) in these positions to value-determined ones. This for example is fine:

$$\exists b : \mathbb{B}.\ \{v : \mu \mathsf{IncrNatListF} \mid (\mathsf{fold\ incr})\, v\, c = b\} \wedge c = 7$$

The value-determinedness restriction for spines of measurements is supported directly by attempting to prove type soundness. In order to give a simple definition of the semantics of (value-determined) existential types and things using it, such as the pack pattern for algebras, in the definition we avoid explicitly constructing semantic values for the packed indices. However, this presents a problem in proving unrolling soundness, subtyping soundness, and (something we call) upward closure (all of which are needed to prove type soundness due to our definition of packed algebras): we must know that a value *uniquely* determines value-determined indices. We factor this into a few lemmas discussed in Chapter 7.

We split the $a : \tau$ in Θ into two: $a \div \tau$ means that a is not value-determined, and $a \overset{\mathsf{d}}{\div} \tau$ means that a is value-determined. Alternatively, we could not split the colon and add $a\,\mathsf{det}$ to the context, but this is more verbose and it would complicate $\Theta \subseteq \Theta'$ which needs to respect value-determinedness: if a is (not) value-determined in Θ then it is (not) value-determined in Θ' too. Anyway, we introduce an operation $\overset{\mathsf{d}}{\div}\Theta$ that filters out everything in Θ not marked as value-determined (including propositions; the operation $\overset{\mathsf{d}}{\div}-$ is used only in proposition-independent judgments). For a well-formed $\{v : \mu F \mid \mathscr{M}\}$, for each

$(\mathsf{fold}_F\ \alpha)\, v\, \boldsymbol{t} = t \in \mathscr{M}$ we require $\boldsymbol{t}$ to be *value-determined*: all its free variables must be value-determined, that is, $FV(\boldsymbol{t}) \subseteq \overset{\mathrm{d}}{\div}\Theta$ for some well-formed Θ. If t is a variable, then we output the value-determinedness information $FV(\boldsymbol{t})\rightarrow t$ saying t is value-determined if the free variables of $\boldsymbol{t}$ are. However, if t is not a variable, we can probably relax the requirement that $\boldsymbol{t}$ be value-determined (we don't try that in this thesis).

This opens the question of where the system should mark indices as value-determined. Following a *delay principle*, we should do so when we have the most information possible for determining value-determinedness (the judgment $\xi \vdash a\ \mathsf{det}$, where ξ collects value-determinedness dependencies $\boldsymbol{b}\rightarrow c$ that is "if all of $\boldsymbol{b}$ is value-determined, then so is c", asserts a is value-determined under ξ). We have the most information possible at the binding sites of value-determined existential and universal types if we syntactically require these value-determined binders (for now, the only universal and existential type binders we consider are value-determined; refinement abstraction and programmer-provided refinements are not value-determined binders) to appear strictly on the outside of a type, using the dependency information generated by the inner part of the type without extractable value-determined binders (according to our previous work, if we are to minimize coupling to the SMT logic for the sake of a simple starter system, in positive types, we shouldn't extract under sum or shift types; in negative types, we shouldn't extract under shift types). By requiring value-determined binders to be outermost, we obviate the need to extract these binders with a judgment that extracts inside products and arrows. We also obviate extracting propositions by requiring asserted and guarding propositions to be between these binders and the remaining types from which nothing should be extracted.

Therefore, we stratify our type grammar. For positive types, we write P for existential types or asserting types, Q for asserting types or the remaining positive types, the *simple*

ones (those from which we can't soundly extract index information in inversion stages), and R for the simple positive types, the latter having $R_1 \times R_2$ (so nothing can be extracted) and $P_1 + P_2$ (nothing can be extracted under $+$, as unrolling must be allowed to output sums of existentials and asserting types). Going in the reverse direction alphabetically, for negative types, we write N for universal types or quarded types, M for guarded types and the remaining negative types, the simple ones, and L for the simple negative types (function and upshift types). Similarly to $R_1 \times R_2$, function types are $L \to R$ so nothing can be extracted from them. In the terminology of Economou et al. [2023], R and L are also said to be simple, that is, invariant under index extraction. This puts all of the types in our refined system into a canonical form: any positive type may be written $\exists^{\mathsf{d}}\Xi.\, R \wedge \overrightarrow{\varphi}$ for some ${}^{\mathsf{d}}\Xi$, $\overrightarrow{\varphi}$, and R; any negative type may be written $\forall^{\mathsf{d}}\Xi.\, \overrightarrow{\varphi} \supset L$ for some ${}^{\mathsf{d}}\Xi$, $\overrightarrow{\varphi}$, and L. A ${}^{\mathsf{d}}\Xi$ is a sequence of value-determined index sortings $a \stackrel{\mathsf{d}}{\div} \tau$. A $\overrightarrow{\varphi}$ is a (possibly empty) list of index propositions, which may also sometimes be written Φ or Ψ.

With premise $\xi_Q - \stackrel{\mathsf{d}}{\div}\Xi \vdash {}^{\mathsf{d}}\Xi\ \mathsf{det}$ for example we require binders to be value-determined:

$$\frac{{}^{\mathsf{d}}\Xi \neq \cdot \qquad \Xi, {}^{\mathsf{d}}\Xi \vdash Q\ \mathsf{type}[\xi_Q] \qquad \xi_Q - \stackrel{\mathsf{d}}{\div}\Xi \vdash {}^{\mathsf{d}}\Xi\ \mathsf{det}}{\Xi \vdash \exists^{\mathsf{d}}\Xi.\, Q\ \mathsf{type}[\xi_Q - {}^{\mathsf{d}}\Xi]}$$

The operation $\xi - a$ is defined by

$$\cdot - a = \cdot$$

$$(\xi, \boldsymbol{b} \to c) - a = \begin{cases} \xi - a & \text{if } c = a \\ (\xi - a) \cup ((\boldsymbol{b} - a) \to c) & \text{else} \end{cases}$$

and we define $\xi \vdash {}^{\mathsf{d}}\Xi\ \mathsf{det}$ by "$\xi \vdash a\ \mathsf{det}$ for all $a \in \mathit{dom}({}^{\mathsf{d}}\Xi)$", and similarly $\xi - \Xi$ by

subtraction of each variable in $dom(\Xi)$ from ξ. As seen with $\xi_Q - \overset{\mathsf{d}}{\div}\Xi$, subtracting $\overset{\mathsf{d}}{\div}\Xi$ from ξ_Q is our way of assuming the value-determinedness information already in context. We also use it to pop off binders from the output: $\xi_Q - {}^{\mathsf{d}}\Xi$.

Defining $\xi - a$ in this way means the order of forming value-determined binders makes no difference. For example, the ξ of $\mathsf{Nat}(a) \times \mathsf{Incr}(a,b)$ is $\cdot\to a, \{a\}\to b$, so both a and b are value-determined. If we first quantify over a as in $\exists a : \mathbb{N}.\, \mathsf{Nat}(a) \times \mathsf{Incr}(a,b)$, then the ξ of this type is $(\cdot\to a, \{a\}\to b) - a = \cdot\to b$, so we can quantify over b. If $\xi - a$ instead removed any $\boldsymbol{b}\to c$ with $a \in \boldsymbol{b}$, then quantifying over a first in this example would yield empty ξ and we wouldn't be able to quantify b.

In order for type well-formedness to be stable under substitution, well-formed substitutions must take value-determined index variables to value-determined index terms (that is, index terms whose index variables are all value-determined). Consider $\exists b : \mathbb{B}.\, \mathsf{Nat}(a) \times \mathsf{Incr}(a,b)$. The variable a is value-determined. A well-formed substitution t/a requires t to be value-determined. Thus $[t/a](\exists b : \mathbb{B}.\, \mathsf{Nat}(a) \times \mathsf{Incr}(a,b))$ is still well-formed (the free variables of t, which are all value-determined, are subtracted from $FV(t)\to b$ yielding $\emptyset\to b$).

Substitution transforms the ξ output of type well-formedness. For first-order sorts, ξ can only have unit entries $\cdot\to a$, and we define $[\sigma]\xi$ by

$$[\sigma]\cdot = \cdot$$

$$[\sigma](\xi, c) = \begin{cases} [\sigma]\xi \cup \sigma(c) & \text{if } \sigma(c) \text{ is a variable} \\ [\sigma]\xi & \text{else} \end{cases}$$

We extend this to higher-order sorts in a natural way:

$$[\sigma]\cdot = \cdot$$

$$[\sigma](\xi, \boldsymbol{b}{\to}c) = \begin{cases} [\sigma]\xi \cup (\langle\sigma\rangle\boldsymbol{b}{\to}\sigma(c)) & \text{if } \sigma(c) \text{ is a variable} \\ [\sigma]\xi & \text{else} \end{cases}$$

where $\langle\sigma\rangle\mathfrak{D}$ is the *free-variable image* of σ on index variable set $\mathfrak{D}$, defined as follows if $\mathfrak{D} \subseteq dom(\sigma)$ (where $dom(\sigma) = \cup_{_/\mathscr{O}\in\sigma}\{\mathscr{O}\}$)[15]

$$\langle\sigma\rangle\mathfrak{D} = \cup_{a\in\mathfrak{D}} FV(\sigma(a))$$

By our restrictions in type well-formedness, $\boldsymbol{b}$ is always value-determined, so if σ is well-sorted, then $\langle\sigma\rangle\boldsymbol{b}$ is also value-determined.

More on hereditary substitution of indices Prawitz [1965] proved normalization of propositional intuitionistic logic in natural deduction (which implies its consistency) morally by lexicographic induction, first, on the structure of formulas and, second, on the structure of proofs. Gentzen [1935] morally used a similar metric to prove cut elimination in intuitionistic sequent calculus (which also implies consistency). By the Curry-Howard correspondence, something like this metric can be used to prove consistency of the simply-typed lambda calculus (see Girard et al. [1989]). Watkins et al. [2004] analyzed these proofs and captured their constructive content in a *hereditary substitution* operation that is like ordinary syntactic substitution but continues to perform substitution until a normal form (in the

[15]The symbol _ means "don't care about this (unused) metavariable" and $\mathscr{O}$ is a generic metavariable which can be specified by context.

sense of no β-redexes, that is, beta-redexes) is obtained (or it gets stuck). For example, ordinary substitution $[(\lambda x.x)/f](f(2))$ equals $(\lambda x.x)(2)$, but hereditary substitution reduces this to a normal form:

$$[(\lambda x.x)/f](f(2)) = \langle \lambda x.x \mid 2 \rangle = \langle [2/x]x \mid \cdot \rangle = \langle 2 \mid \cdot \rangle = 2$$

We also use the notation $\langle - \mid - \rangle$ for proof terms for primary cuts in sequent calculus but when used there it is not "active" like hereditary application as used in the definition of hereditary substitution. However, the cut elimination relation $\rightsquigarrow$ performs a similar reduction.

Hereditary substitution seems not to scale extremely well to the richest of type systems (system F, dependent types), in which case logical relations [Tait, 1967, Plotkin, 1976] are usually used to prove consistency, which is a technique relating syntax and semantics (whereas hereditary substitution as understood conventionally is purely syntactic). Tait used logical relations[16] to prove strong normalization of system T. Girard extended Tait's method to prove system F is strongly normalizing. But hereditary substitution is still useful for rich type systems, while being more down to earth. Stucki and Giarrusso [2021] for example use hereditary substitution to prove weak normalization of types in an extension of system F^{ω} with higher-order subtyping. We use a simple formulation of hereditary substitution to ensure that our refinement types are in a canonical, SMT decidable form.

It is common to define the hereditary substitution operation by indexing it with the type of the term being substituted. The key induction measure includes the size of this

[16]Plotkin [1973, 1976] seems first to describe in print a relation as "logical" in this way, but Neel Krishnaswami says (in the history tab of his site `https://www.cl.cam.ac.uk/~nk480/`) that Rick Statman tells him that Mike Gordon was first to do so (for some reason), and that "he and Gordon Plotkin picked it up from him". What is logical about it? My best guess is that the (Fregean) distinction between sense and reference, which are related in a logical relation, can be said to be a logical one.

type. Alternatively, we can omit types or sorts from the definition of substitution and prove substitution terminates on well-typed (or well-sorted) terms. This alternative is the approach we take.

It is most convenient to work with index spines, that is, left-growing lists of normal indexes. Index spines $\boldsymbol{t}$ ultimately synthesize the return sort after processing the entire input sort. The remaining (that is, non-spinal) index terms t are normal in that they cannot be β-reduced. The only index-level application form is $a(\boldsymbol{t})$ and must have a variable a as a head; when a term, even a lambda, is substituted for a, it is reduced to a normal form. We distinguish the passive and active parts of hereditary substitution: the *active* part, which we write as $\langle u \mid \boldsymbol{t} \rangle$, can be thought of as a meta-level application of u (which need not be a variable) to $\boldsymbol{t}$, and fully reduces to a normal index t; and the *passive* part is ordinary substitution $[u/a]t$. The active and passive aspects of hereditary substitution are mutually recursive.[17] Instead of "parallel" the word "simultaneous" is also used [Stoughton, 1988].

It is tricky to prove that hereditary application is admissible, but similar to proving cut elimination:

$$\frac{\Xi \vdash u : \tau \qquad \Xi; [\tau] \vdash \boldsymbol{t} : \kappa}{\Xi \vdash \langle u \mid \boldsymbol{t} \rangle : \kappa}$$

This is a mutually recursive part of the syntactic substitution lemma at the index level: Lemma C.17. We write τ or ω for so-called *higher-order* sorts, specifically sorts in which at least one arrow sort $\Rightarrow$ occurs; arrow sorts classify index-level lambdas $\lambda a.u$ binding a (of first-order sort κ) in u. Really, these are second-order sorts in that they can't take an arrow sort as input. We would want to consider sorts of order higher than second when we allow in future work index spines in measurements to mention lambdas for abstract

[17] I got the terminology "active" and "passive" from some notes by Abel [2019] on parallel hereditary substitution, which is also our flavor of index substitution.

refinements. We reserve κ for first-order sorts, which we define to be sorts without any occurrence of an arrow sort.

It is also tricky to situate hereditary substitution properly with respect to the overall system. In particular, in other parts of the metatheory, we need to prove Barendregt's[18] distribution property of substitution that looks something like

$$[t/a]([u/b]t') = [[t/a]u/b]([t/a]t')$$

provided $b \notin FV(t)$. I am in the habit (picked up from Matt Hammer) of referring to this broadly as "Barendregt's substitution lemma" because a similar result by the name "substitution lemma" is proved in his standardizing textbook on the untyped lambda calculus. To do this, we already need a somewhat complicated lexicographic induction measure, first, on the structure of the sort of b, second, on the part number of the lemma (which includes the part shown above for a normal index term t', a similar part for index spines, and a part saying that the active and passive aspects of hereditary substitution commute), and third, on the structure of the index term. However, in view of the rest of the system, it is convenient to generalize the distribution property from the single substitution t/a to an arbitrary substitution σ. In this case, the induction metric gets a bit more complicated: in the first part of the lexicographic induction, to the size of the sort of b, we must add the sizes of the sorts of non-identity substitutions in σ.

These induction metrics for index substitution metatheory are similar to those used to prove cut elimination but a bit more complicated.

[18] A similar lemma is in the standard textbook on untyped lambda calculus by Barendregt [1981]

3.4 Focusing & Its Applications to Programming Languages

Focusing is a technique for reducing nondeterminism in proof search procedures. From a Curry–Howard perspective, formulas correspond to types, and proofs correspond to programs, so proof search corresponds to program synthesis: given a context and a type, automatically synthesize a program of the given type under the given context. However, program synthesis, while (perhaps) theoretically interesting and practically convenient, is outside the scope of this thesis, which focuses on type inference and type checking (checking a program against a type or inferring a type for a program).

Andreoli [1992] first studied focusing in the setting of linear logic; its corresponding notion of polarity, discussed shortly, also appeared in Girard's work on unifying classical, intuitionistic, and linear logics [Girard, 1993]. A rule is invertible if its conclusion implies its premises. In proof search, where and when invertible rules are applied doesn't affect whether a proof can be found, so a search procedure can nondeterministically choose to apply invertible rules. However, one can eagerly apply invertible rules without worrying about possibly needing to backtrack, so the search procedure can avoid this nondeterminism. Here enters a notion of *polarity* in sequent calculus proof search: *negative* formulas (or types) are those with invertible right rules, and *positive* formulas (or types) are those with invertible left rules. Further, Andreoli observed that one can select a formula on which to focus—that is, to decompose fully, up to an atom or a polarity shift, using non-invertible rules—without losing soundness or completeness. Focusing is characterized primarily by this latter, subtler observation: a proof search procedure which exploits such, and *only* such focusing is *weakly focused*; if, in addition to this, the procedure eagerly applies invertible rules, then it is said to be *strongly focused* [Simmons, 2014]. The terminology of *weak* and *strong* focalization seems to appear first in an unpublished note by Laurent [2004] though

in a slightly different way.

Focusing and polarization are computationally interesting. They have been used to explain many common phenomena in programming languages. It has been used to illuminate aspects of pattern matching [Zeilberger, 2008a, Krishnaswami, 2009], higher-order abstract syntax [Zeilberger, 2008b], functional program compilation [Downen, 2017], and the value and evaluation context restrictions that arise in type systems with intersection and union types [Zeilberger, 2009]. For example, Krishnaswami [2009] explains that pattern matching arises as the left-inversion stage of focused systems (by the way, the system in that paper is bidirectional). More broadly, Downen [2017] discusses many logical dualities common in programming languages.

Focusing is closely related to CBPV. From a CH perspective, the polarization of focused proof search syntactically reflects the semantic duality between CBN and CBV [Zeilberger, 2008a]. Brock-Nannestad et al. [2015] study the relation between polarized intuitionistic logic and CBPV. They obtain a bidirectionally typed system of natural deduction related to a variant of the focused sequent calculus *LJF* [Liang and Miller, 2009] by η-expansion (for inversion stages). Espírito Santo [2017] does a similar study, but starts with a focused sequent calculus for intuitionistic logic much like the system of Simmons [2014] (but without positive products), proves it equivalent to a natural deduction system (we think the lack of positive products helps establish this equivalence), and defines, also via η-expansion, a variant of CBPV in terms of the natural deduction system. Our system is not natural deduction, but rather sequent calculus. Our system relates to usual presentations of CBPV in a similar way—via η-expansion (see definition of force in Ch. 5).

As such, we may regard CBPV's computation types as negative and value types as positive, and lift and thunk types as polarity shifts.

Focusing, CBPV, and bidirectional typing An inference rule is *invertible* if its conclusion implies its premises. For example, in intuitionistic sequent calculus, the right rule for implication is invertible because its premise $\Gamma, A \vdash B$ can be derived from its conclusion $\Gamma \vdash A \to B$:

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash A \to B}\,(\text{Assume} \to\text{R conclusion})}{\Gamma, A \vdash A \to B}\,(\text{Weaken}) \qquad \dfrac{\dfrac{}{\Gamma, A \vdash A} \qquad \dfrac{}{\Gamma, A, B \vdash B}}{\Gamma, A, A \to B \vdash B}\,{\to}\text{L}}{\Gamma, A \vdash B}\,(\text{Cut})$$

However, both right rules for disjunction, for example, are not invertible, which we can prove with a counterexample: $A_1 + A_2 \vdash A_1 + A_2$ but $A_1 + A_2 \nvdash A_1$ and $A_1 + A_2 \nvdash A_2$. In a sequent calculus, *positive* formulas have invertible left rules and *negative* formulas have invertible right rules. A *weakly* focused sequent calculus eagerly applies non-invertible rules as far as possible (in either *left-* or *right-focusing* stages); a *strongly* focused sequent calculus does too, but also eagerly applies invertible rules as far as possible (in either *left-* or *right-inversion* stages). There are also *stable* stages (or moments) in which a decision has to be made between focusing on the left, or on the right [Espírito Santo, 2017]: this is the main source of backtracking in an implementation of focused proof search. The decision can be made explicitly via proof terms (specifically, cuts): in our system, a principal task of let-binding, a kind of cut, is to focus on the left (to process the list of arguments in a bound function application); and a principal task of pattern matching, another kind of cut, is to focus on the right (to decompose the value being matched against a pattern).

From a Curry–Howard view, let-binding and pattern matching are different kinds of

cuts. The cut formula A—basically, the type being matched or let-bound—must be *synthesized* (inferred) as an output (judgmentally, $\cdots \Rightarrow A$) from *heads* h (variables and annotated values) or *bound expressions* g (function application and annotated returner expressions); and ultimately, the outcomes of these cuts in our system are synthesized. But all other program terms are *checked* against input types A: judgmentally, $\cdots \Leftarrow A \cdots$ or $\cdots [A] \vdash \cdots$ (in future work, it may be interesting to add more type inference). In this sense, both our declarative and algorithmic type systems are *bidirectional* [Dunfield and Krishnaswami, 2021].

Borrowing terminology from Kant, Martin-Löf [1994] considers an *analytic* judgment to be one that is derivable using information found only in its inputs (in the sense of the bidirectional modes, input and output). A *synthetic* judgment, in contrast, requires us to look beyond the inputs of the judgment in order to find a derivation. The metatheoretic results for our algorithmic system demonstrate that our judgments are analytic, *except* the judgment $\Theta \vdash \varphi$ true, which is verified by an external SMT solver. As such, our system may be said to be analytic modulo an external SMT solver. Focusing, in proper combination with bidirectional typing (which clarifies where to put type annotations), let-normality[19] and value-determinedness, guarantees that all information needed to generate verification conditions suitable for an SMT solver may be found in the inputs to judgments. In our system, cut formulas can always be inferred from a type annotation or by looking up a variable in the program context, making all our cuts (in the bidirectional system) analytic in a sense analogous to Smullyan [1968].

We design refinement type syntax so that in inversion stages, that is, expression type-checking (where a negative type is on the right of a turnstile) and pattern matching (where a

[19] Let-normality refers to that property that computations must be named and sequenced.

positive type is on the left of a turnstile), refinements are eagerly *extracted* from types in order to be used. For example, suppose we want to check the expression $\lambda x.\texttt{return}\,x$ against the type $\mathsf{ff} \supset 1 \to \uparrow(1 \wedge \mathsf{ff})$, which is semantically equivalent to $(1 \wedge \mathsf{ff}) \to \uparrow(1 \wedge \mathsf{ff})$ but the latter is ungrammatical because function input type $1 \wedge \mathsf{ff}$ is not simple (ff can soundly be extracted from it). We need to assume ff (that is, put it in the input logical context) so that we can later use it (in typechecking the value x) to verify ff (of function output type $\uparrow(1 \wedge \mathsf{ff})$). We require types of program variables to be simple. If we instead allowed say $x : 1 \wedge \mathsf{ff}$ in the program context, then ff would not be usable (unless we could extract it from the program context elsewhere, but it's simpler to extract from types as soon as possible) and typechecking would fail (it should succeed). Similarly, since subtyping may be viewed as implication, index information from positive subtypes or negative supertypes are eagerly extracted for use.

Our declarative typing system (Ch. 6) includes two focusing stages, one (value typechecking) for positive types on the right of the turnstile ($\vdash$), and the other (spine typing) for negative types on the left. Our algorithmic system (Ch. 8) closely mirrors the declarative one, but does not conjure index instantiations or witnesses (like σ in DeclSpine$\forall$ below), and instead systematically introduces and solves existential variables (like solving the existential variables $\widehat{{}^{\mathsf{d}}\Xi}, \Delta$ as (the algorithmic part of) Ω in AlgSpine$\forall$ below), which we keep

in algorithmic contexts $\hat{\Theta}$ (and Δ but these have no solutions by a convention we stipulate).

$$\frac{\overset{\mathsf{d}}{\div}\Theta \vdash \sigma : {}^{\mathsf{d}}\Xi \qquad \Theta;\Gamma;[[\sigma]N] \vdash s \Rightarrow \uparrow P}{\Theta;\Gamma;[\forall^{\mathsf{d}}\Xi.N] \vdash s \Rightarrow \uparrow P}\ \text{DeclSpine}\forall$$

$$\frac{{}^{\mathsf{d}}\Xi \text{ may be } \cdot \qquad \Theta,\widehat{{}^{\mathsf{d}}\Xi};\Gamma;[[\widehat{{}^{\mathsf{d}}\Xi}/{}^{\mathsf{d}}\Xi]M] \vdash s \Rightarrow \uparrow P \mathbin{/} \chi \dashv \Delta \qquad \overline{\Theta},\widehat{{}^{\mathsf{d}}\Xi},\Delta \vdash \chi\ \mathsf{Wf}[\xi] \qquad \Theta,[\xi](\widehat{{}^{\mathsf{d}}\Xi},\Delta);\Gamma \vdash \chi\ \mathsf{fixInstChk} \dashv \Omega}{\Theta;\Gamma;[\forall^{\mathsf{d}}\Xi.M] \rhd s \Rightarrow \uparrow[\Omega]([\Omega]P)}\ \text{AlgSpine}[\forall]$$

For example, applying a function of type $\forall b : \mathbb{N}.\,\mathsf{List}(\mathsf{Nat})(b) \to \cdots$ to the list $[4,1,2]$ should solve b to an index semantically equal to 3; the declarative system conjures an index term (like $3+0+0+1+0-1$), but the algorithmic system mechanically solves a possibly different but necessarily SMT-equivalent index (based on the mechanics of unrolling and subtyping).

The judgment $\Theta;\Gamma;[\forall^{\mathsf{d}}\Xi.M] \rhd s \Rightarrow \uparrow[\Omega]([\Omega]P)$ is top level algorithmic left-focusing. It focuses on decomposing $\forall^{\mathsf{d}}\Xi.M$ (the type of the function being applied), introducing its existential variables for the arguments in the value list s (or spine), and outputting a type with solutions to these existentials under which all constraints hold. By "top level" we mean that it is the judgment used by the other (non-focusing) judgments: they don't necessarily check against a universal type, which the side condition "${}^{\mathsf{d}}\Xi$ may be $\cdot$" is intended to clarify. Its first main premise $\Theta,\widehat{{}^{\mathsf{d}}\Xi};\Gamma;[[\widehat{{}^{\mathsf{d}}\Xi}/{}^{\mathsf{d}}\Xi]M] \vdash s \Rightarrow \uparrow P \mathbin{/} \chi \dashv \Delta$ is algorithmic left-focusing and in the end it outputs constraints χ which may have (in addition to $\widehat{{}^{\mathsf{d}}\Xi}$) evars newly generated by typechecking values of algebraic datatypes. We use the value-determined dependency information ξ of the entire focusing stage in the algorithm to delay solving evars which we know can be solved later using invariants of polarized subtyping,

possibly with backtracking. Ultimately, this ξ retains the value-determined dependencies of the input type (but with new evars Δ mediating it in a solvable way) and we use this information to that all evars are guaranteed to be solved at the end of focusing: the context Ω in $\Theta, [\xi](\widehat{\mathsf{d}\Xi}, \Delta); \Gamma \vdash \chi$ fixInstChk $\dashv \Omega$ is *complete* (every evar in its domain is solved). The algorithmic left focusing stage thus ultimately outputs $\uparrow[\Omega]([\Omega]P)$ which is *ground*, that is, has no existential variables, and hence SMT solvable (Ω is applied twice because evars that had been delayed to solve later were allowed in solutions to the evars which were not delayed). In this way no existential variables leak to non-focusing stages.

The existential variables introduced at the beginning of left-focusing stages flow to the right-focusing stage (value typechecking) and are solved there, possibly via subtyping. Constraints χ are only checked right at the end of focusing stages, when all their existential variables are solved. Dually, our top level algorithmic right-focusing judgment has the form $\hat{\Theta}; \Gamma \rhd v \Leftarrow P$ and main premise $\hat{\Theta}; \Gamma \vdash v \Leftarrow P \,/\, \chi \dashv \Delta'$, where χ is an output list of typing constraints and Δ' is an output context that includes index solutions to existentials. For example, consider our rule for (fully) applying a function h to a list of arguments s:

$$\frac{\Theta; \Gamma \rhd h \Rightarrow \downarrow N \qquad \Theta; \Gamma; [N] \rhd s \Rightarrow \uparrow P}{\Theta; \Gamma \rhd h(s) \Rightarrow \uparrow P}$$

After synthesizing a thunk type $\downarrow N$ for the function h we are applying, we process the entire list of arguments s using the *top level* judgment $\Theta; \Gamma; [N] \rhd s \Rightarrow \uparrow P$ where all evars have been solved and all constraints generated within the focusing stage have been verified under them and ultimately a type $\uparrow P$ without evars is inferred.

The polarization of CBPV helps guarantee all solutions have no existential variables. Focusing stages introduce existential variables to input types, which may appear initially

as a positive supertype in the subtyping premise for typechecking (value) variables. These existential variables are solved using the positive subtype, which never has existential variables. Dually, negative subtypes may have existential variables, but negative supertypes never do.

Focusing also gives us pattern matching for free [Krishnaswami, 2009]: from a Curry–Howard view, the left-inversion stage is pattern matching. The (algorithmic[20]) left-inversion stage in our system is written $\Theta;\Gamma;[P] \triangleright \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$: it decomposes the positive P (representing the pattern being matched) on the left of the turnstile (written $\triangleright$ to distinguish the algorithmic judgment from the corresponding declarative judgment, which instead uses $\vdash$). Our system is weakly focused: it does not eagerly apply invertible rules. Pattern matching in our system thereby resembles the original presentation of pattern matching in CBPV. From a Curry–Howard view, increasing the strength of focusing would permit nested patterns.

A pattern type can have index equality constraints, such as for refined ADT constructors (for example, that the length of an empty list is zero) as output by unrolling. By using these equality constraints, we get a coverage-checking algorithm. For example, consider checking get (introduced in Chapter 1) against the type

$$\forall l,k : \mathbb{N}.\ \{v : \mathsf{List}\ A \mid \mathsf{len}\ v = l\} \to (\{v : \mathsf{Nat} \mid \mathsf{index}\ v = k\} \wedge (k < l)) \to \uparrow A$$

At the clause

$$\mathtt{get}\ [\,]\ y = \mathtt{unreachable}$$

we extract a logically inconsistent context ($l : \mathbb{N}, k : \mathbb{N}, l = 0, k < l$), which entails that

[20] We give the algorithmic judgment to highlight that existential variables do not flow through it, or any of the non-focusing stages: there is no hat on an input context.

`unreachable` checks against any type. Proof-theoretically, this use of equality resembles the elimination rule for Girard–Schroeder-Heister equality [Girard, 1992, Schroeder-Heister, 1994].

3.5 A Reading Guide

Judgments and relations are *moded*: metavariables in a judgment are assigned exclusively either an *input* or an *output* mode. Metavariables appearing on the left of a turnstile are always in input mode. Output-moded metavariables always appear toward the right of a judgment. Judgments do not mix input and output modes in such a way that (for example) there is an input, then an output, then an input; a judgment always has some number of input metavariables on the left, followed by zero or more output metavariables on the right.

We follow the variable conventions (adapted to our setting) of Barendregt [1981], which he justifies using de Bruijn indices to represent binders canonically.

(1) We identify alpha equivalent terms, that is terms which are identical up to consistent renaming of bound variables.

(2) If $\mathscr{O}$ occurs in a certain mathematical context (like a definition or proof) then every bound variable occurring in $\mathscr{O}$ is chosen to be different from the free variables. That is, we freshen bound variables when adding them to an input context so they are always new to that context.

We define many metaoperations. For example, we define $dom(\Theta)$ to be the set of index

variables sorted in Θ:

$$dom(\cdot) = \cdot$$

$$dom(\Theta, a : \tau) = dom(\Theta) \cup \{a\}$$

$$dom(\Theta, \varphi) = dom(\Theta)$$

We define $dom(\hat{\Theta})$ similarly: $dom(\hat{\Theta}, \hat{a} : \kappa{=}t) = dom(\hat{\Theta}) \cup \{\hat{a}\}$ and so on (including uvars a). We also define $dom(\Gamma)$ similarly (the set of program variables). We define $\hat{\Theta}(\hat{a})$ to be the sort assigned to $\hat{a}$ in $\hat{\Theta}$.

We might define things multiple times, like where they are introduced in the main text, and also in the appendix. We organize references to miscellaneous definitions in Fig. A.86 so if a reader does not know how something is defined it could maybe be found there. This figure is found at the end of Chapter A which organizes all the definitions pertaining to the refined system(s).

When metatheoretic statements are made in the main text, they will usually link to the same statements in the appendix where more details and proofs can be found.

Notation: We define the disjoint union $X \uplus Y$ of sets X and Y by $X \uplus Y = (\{1\} \times X) \cup (\{2\} \times Y)$ and define the injections $inj_k : X_k \to X_1 \uplus X_2$ by $inj_k(d) = (k, d)$. We write the projections of a (binary) product as $\pi_k : X_1 \times X_2 \to X_k$ (where $k \in \{1,2\}$): that is, $\pi_k(x_1, x_2) = x_k$. In the set-theoretic denotational semantics we sometimes use pattern-matching notation like $(V_1, V_2) \mapsto f(V_1, V_2)$ to stand for the function $V \mapsto f(\pi_1(V), \pi_2(V))$. Semantic values are usually named d, f, g, or V.

Chapter 4

Focusing on an Unrefined System

We begin with intuitionistic logic because it is the logical foundation of standard functional programming. In particular, we basically begin with a simple fragment of Gentzen's LJ, an intuitionistic sequent calculus. We focalize this logic with a view toward refinement typing of a (semi)standard functional programming language. We then bidirectionalize it (and in the next chapter add recursion and algebraic datatypes) to obtain our core unrefined system.

4.1 A Sequent Calculus for Intuitionistic Propositional Logic

Of course, we absolutely need arrow types (for functions (*function*al programming)). Looking forward to algebraic datatypes: we will construct ADTs with the basic building blocks of sums and products, including their units, zero and one. Figure 4.1 defines a grammar of logical formulas consisting of falsity (void type) 0, disjunction (sum type) $+$, truth (unit type) 1, conjunction (product type) $\times$, and implication (function type) $\to$. Figure 4.1 defines a standard intuitionistic sequent calculus for these formulas. In a setting without proof terms, we define a *context* Γ to be a multiset[1] having formulas A as elements.

[1] A *multiset* or *bag* is a set in which an element can appear more than once.

Gentzen [1935]'s left conjunction rules are not invertible, but Ketonen's left conjunction rule is (and, as we'll see, therefore corresponds to pattern matching, which is what we want[2]), so we use Ketonen [1944]'s.

$$\begin{array}{lrcl} \text{Formulas} & A,B,C,P,Q,R,N,M,L & ::= & 0 \mid A+B \mid 1 \mid A \times B \mid A \to B \\ \text{Contexts} & \Gamma & ::= & \cdot \mid \Gamma, A \end{array}$$

$\boxed{\Gamma \vdash A}$ Under hypotheses Γ it is the case that A is true

$$\frac{}{(\Gamma \ni A) \vdash A}\ \text{Hyp}$$

$$\frac{\Gamma \vdash A \qquad \Gamma, B \vdash C}{\Gamma \ni A \to B \vdash C}\ {\to}\text{L} \qquad\qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ {\to}\text{R}$$

$$\text{(no ``1L'')} \qquad\qquad \frac{}{\Gamma \vdash 1}\ 1\text{R}$$

$$\frac{\Gamma, A, B \vdash C}{\Gamma \ni A \times B \vdash C}\ {\times}\text{L} \qquad\qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \times B}\ {\times}\text{R}$$

$$\frac{}{\Gamma \ni 0 \vdash A}\ 0\text{L} \qquad\qquad \text{(no ``0R'')}$$

$$\frac{\Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \ni A + B \vdash C}\ {+}\text{L} \qquad\qquad \frac{k \in \{1,2\} \qquad \Gamma \vdash A_k}{\Gamma \vdash A_1 + A_2}\ {+}\text{R}$$

Figure 4.1: Sequent calculus for intuitionistic (propositional) logic

Definition 4.1 (Main and Secondary Premises)**.** Given a system of rules deriving mutually recursive judgments $\mathscr{J}_1, \ldots, \mathscr{J}_n$ (where $n > 0$), we call the premises of form $\mathscr{J}_1$ or $\ldots$ or $\mathscr{J}_n$ *main* or *primary* premises and the other premises *secondary* premises or *side conditions*.

[2]We omit negative product types in this thesis which focuses on modularly refining algebraic datatypes, which are positive. Negative products could be relevant for codata which would be negative; this is potential future work.

For example, in Fig. 4.1, the premise $k \in \{1,2\}$ is a side condition of rule $+$R, $\Gamma \vdash A_k$ is a main premise of $+$R, and $\Gamma \ni A$ is a side condition of rule Hyp. (The rule Hyp may also be written

$$\frac{A \in \Gamma}{\Gamma \vdash A}$$

and similarly for rules $\to$L and $\times$L and 0L and $+$L.)

Definition 4.2 (Admissible)**.** Given a logical system, a proposed inference rule is *admissible* with respect to the system if its premises imply its conclusion within the system.

Because we assume Γ is a (multi)*set*, the structural rule of exchange is admissible:

Lemma 4.1 (Exchange Admissible)**.** *The rule*

$$\frac{\Gamma_1, B, A, \Gamma_2 \vdash C}{\Gamma_1, A, B, \Gamma_2 \vdash C}\ \textit{(Exchange)}$$

is admissible in Fig. 4.1. That is, if $\Gamma_1, B, A, \Gamma_2 \vdash C$ *then* $\Gamma_1, A, B, \Gamma_2 \vdash C$.
Moreover, the proof of $\Gamma_1, A, B, \Gamma_2 \vdash C$ *has the same structure as* $\Gamma_1, B, A, \Gamma_2 \vdash C$.

Proof. By structural induction on the derivation of $\Gamma_1, B, A, \Gamma_2 \vdash C$. □

Hyp is the only *structural* rule given in Fig. 4.1, in that it is independent of any particular logical connective; the remaining rules there are *logical* rules in that they pertain to logical connectives. Because we defined the initial rules[3] (that is, Hyp, 0L, and 1R) to allow unused antecedents in Γ, the structural rule of weakening is admissible in Fig. 4.1.

[3] In this thesis, an *initial* rule is one with no main premises. Often the word initial is reserved only to describe the Hyp rule.

Lemma 4.2 (Weaken Admissible)**.** *The rule*

$$\frac{\Gamma \vdash A}{\Gamma, B \vdash A}\ \textit{(Weaken)}$$

is admissible in Fig. 4.1.

Moreover, the proof of $\Gamma, B \vdash A$ has the same structure as that of $\Gamma \vdash A$.

Proof. By structural induction on the derivation of $\Gamma \vdash A$. □

In Fig. 4.1, we have defined the left rules to be *persistent*, that is, hypotheses persist in a proof even after they are used, reflecting the way they are used in intuitionistic natural deduction: when a hypothesis is used it is not consumed and is still available to use (until discharged). However, in sequent calculus, all non-structural (that is, logical) rules (when read from premises to conclusion) introduce a *principal* formula. In particular, non-structural left rules introduce a principal formula on the left.

Therefore, in order to emphasize this aspect of sequent calculus while also maintaining persistence, we write, for example, $\Gamma \ni A \to B$ in the conclusion of $\to$L (which introduces the principal formula $A \to B$ to the left of the turnstile $\vdash$). In the conclusion of a rule, $\Gamma \ni B \vdash A$ is syntactic sugar for a rule concluding $\Gamma \vdash A$ using the side condition $\Gamma \ni B$.[4] Using Γ in the premises of $\to$L therefore means we still have $A \to B$ at our disposal: $A \to B \in \Gamma$. Alternatively, we could get a persistent rule by writing, say,

$$\frac{\Gamma, A \to B \vdash A \qquad \Gamma, A \to B, B \vdash C}{\Gamma, A \to B \vdash C}$$

[4] I got this idea from Scherer [2016].

(and similarly for $+$L and $\times$L) but this is much more verbose and is cluttered with formulas whose only purpose is to make contraction admissible and which don't necessarily have anything to do with introducing the principal logical connective $\to$ (or $+$ or $\times$).

Lemma 4.3 (Contract Admissible)**.** *The rule*

$$\frac{\Gamma, C, C \vdash A}{\Gamma, C \vdash A}\ \textit{(Contract)}$$

is admissible in Fig. 4.1.

Moreover, the proof of $\Gamma, C \vdash A$ has the same structure as that of $\Gamma, C, C \vdash A$.

Proof. By structural induction on the derivation of $\Gamma, C, C \vdash A$. □

Without persistent rules, we would need to add a contraction rule to Fig. 4.1 in order to prove that our final structural rule, cut, is admissible. Cut allows us to use succedents (formulas on the right of a turnstile $\vdash$) as hypotheses. After proving admissibility of cut, consistency of the system with cut is a straightforward corollary, because the system without cut clearly cannot derive 0 under no assumptions.

Admissibility of cut is crucial from a programming perspective because it means any proof in the system with cut can be *normalized* into a proof without cut. We will eventually make cut(s) explicit because they are useful to programmers. For example, let-binding the result of a computation in another computation is a cut, and so is pattern-matching: being able to eliminate these cuts basically means let-binding and pattern-matching behave computationally as expected (but recursion can yield divergence, that is, eliminating a cut can produce further cuts which will only reduce to more cuts and so on).

We prove the admissibility of cut in the structural style of Pfenning [2000], Simmons [2014], and (most closely) Espírito Santo [2017].

Lemma 4.4 (Cut Admissible)**.** *The rule*

$$\frac{\Gamma \vdash C \qquad \Gamma, C \vdash A}{\Gamma \vdash A}\ (Cut)$$

is admissible in Fig. 4.1.

Proof. It is easier to prove cut elimination in an equivalent system with proof terms. We introduce proof terms for Fig. 4.1 in Fig. 4.2 and leave proving its equivalence to the reader. The I used in match terms is an index set of size zero, one, or two and $\{r_i \Rightarrow e_i\}_{i \in I}$ is a list of $\#I$ clauses $r_i \Rightarrow e_i$. With proof terms added, we require a well-formed Γ to be a *set* of variable bindings $x : A$. That is, each variable (written x or y or z or x' or ...) can occur at most once in Γ.

In a style like Espírito Santo [2017] and Scherer [2016] we define proof terms for cuts, and cut elimination, in Fig. 4.3.

Here we organize cut elimination similarly to Section 4.2 of the PhD thesis of Scherer [2016]: there, $\rightsquigarrow_1$ is called a *principal* cut, $\rightsquigarrow_2$ an *initial* cut, and $\rightsquigarrow_3$ a *commutative* cut. The main differences are the notation, the polarity of product types, and there the right-hand side of "$\rightsquigarrow_1$" post-applies (rather than pre-applies as we do) the original cut, but otherwise it's about a similar (likely equivalent) sequent calculus. We find pre-applying leads to a simpler proof, where the induction metric (for admissibility of cut, the main technical lemma used by cut elimination, and cut elimination itself) is the same as that of Pfenning [2000]. Note that this notion of normalization is much clunkier than the one of natural

$$\begin{array}{lll} \text{Contexts} & \Gamma & ::= \cdot \mid \Gamma, x : A \\ \text{Terms} & e & ::= x \mid \lambda x.\, e \mid \mathsf{let}\ y = x(e);\ e' \mid \langle\rangle \mid \langle e_1, e_2\rangle \mid \mathsf{inj}_k\, e \mid \mathtt{match}\ x\ \{r_i \Rightarrow e_i\}_{i\in I} \\ \text{Patterns} & r & ::= \langle x_1, x_2\rangle \mid \mathsf{inj}_k\, x \end{array}$$

$\boxed{\Gamma \vdash e : A}$ Under hypotheses Γ we know e is a proof of A

$$\frac{}{(\Gamma \ni (x : A)) \vdash x : A}$$

$$\frac{\Gamma \vdash e : A \qquad \Gamma, y : B \vdash e' : C}{\Gamma \ni (x : A \to B) \vdash \mathsf{let}\ y = x(e);\ e' : C} \qquad \frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x.\, e : A \to B}$$

$$\frac{}{\Gamma \vdash \langle\rangle : 1}$$

$$\frac{\Gamma, y : A, z : B \vdash e : C}{\Gamma \ni (x : A \times B) \vdash \mathtt{match}\ x\ \{\langle y, z\rangle \Rightarrow e\} : C} \qquad \frac{\Gamma \vdash e_1 : A \qquad \Gamma \vdash e_2 : B}{\Gamma \vdash \langle e_1, e_2\rangle : A \times B}$$

$$\frac{}{\Gamma \ni (x : 0) \vdash \mathtt{match}\ x\ \{\} : A}$$

$$\frac{\Gamma, x_1 : A_1 \vdash e_1 : C \qquad \Gamma, x_2 : A_2 \vdash e_2 : C}{\Gamma \ni (x : A_1 + A_2) \vdash \mathtt{match}\ x\ \{\mathsf{inj}_1\, x_1 \Rightarrow e_1 \mid \mathsf{inj}_2\, x_2 \Rightarrow e_2\} : C}$$

$$\frac{k \in \{1, 2\} \qquad \Gamma \vdash e : A_k}{\Gamma \vdash \mathsf{inj}_k\, e : A_1 + A_2}$$

Figure 4.2: Proof terms for IPL

deduction, the presence of commutative cuts makes it non-confluent in an inconsistent context (consider for example $\mathsf{let}\ x = \mathtt{match}\ y\ \{\};\ \langle\rangle$ where $y : 0$), and it's very likely we can prove it is weakly normalizing [Scherer, 2016] but we are going to move on to designing and proving other results about a refinement type system.

By lexicographic induction, first, on cut formula C structure, second, on e structure, and third, on e' structure. Use (the proof term version of) Lemma 4.1 in some cases. □

We just proved we can eliminate cut in a standard intuitionistic propositional logic.

$$\frac{\Gamma \vdash e : C \qquad \Gamma, x : C \vdash e' : A}{\Gamma \vdash \mathsf{let}\ x = e;\ e' : A}$$

$$\begin{aligned}
\mathsf{let}\ x = \lambda z.\, e;\ \mathsf{let}\ y = x(e_0);\ e' &\rightsquigarrow_1 \\
&\mathsf{let}\ y = (\mathsf{let}\ z = (\mathsf{let}\ x = \lambda z.\, e;\ e_0);\ e);\ \mathsf{let}\ x = \lambda z.\, e;\ e' \\
\mathsf{let}\ x = \langle e_1, e_2 \rangle;\ \mathsf{match}\ x\ \{\langle x_1, x_2 \rangle \Rightarrow e'\} &\rightsquigarrow_1 \mathsf{let}\ x_1 = e_1;\ \mathsf{let}\ x_2 = e_2;\ \mathsf{let}\ x = \langle e_1, e_2 \rangle;\ e' \\
\mathsf{let}\ x = \mathsf{inj}_k\, e_k;\ \mathsf{match}\ x\ \{\mathsf{inj}_1\, x_1 \Rightarrow e'_1 \mid \mathsf{inj}_2\, x_2 \Rightarrow e'_2\} &\rightsquigarrow_1 \mathsf{let}\ x_k = e_k;\ \mathsf{let}\ x = \mathsf{inj}_k\, e_k;\ e'_k \\
\mathsf{let}\ x = e;\ x &\rightsquigarrow_2 e \\
\mathsf{let}\ x = e;\ y &\rightsquigarrow_2 y \quad \text{if } y \neq x \\
\mathsf{let}\ x = y;\ e &\rightsquigarrow_2 [y/x]e \\
\mathsf{let}\ x = e;\ \langle\rangle &\rightsquigarrow_3 \langle\rangle \\
\mathsf{let}\ x = e;\ \langle e'_1, e'_2 \rangle &\rightsquigarrow_3 \langle \mathsf{let}\ x = e;\ e'_1, \mathsf{let}\ x = e;\ e'_2 \rangle \\
\mathsf{let}\ x = e;\ \mathsf{inj}_k\, e' &\rightsquigarrow_3 \mathsf{inj}_k\, (\mathsf{let}\ x = e;\ e') \\
\mathsf{let}\ x = e;\ \lambda y.\, e' &\rightsquigarrow_3 \lambda y.\, \mathsf{let}\ x = e;\ e' \\
\mathsf{let}\ x = (\mathsf{let}\ y = z(e_0);\ e);\ e' &\rightsquigarrow_3 \mathsf{let}\ y = z(e_0);\ \mathsf{let}\ x = e;\ e' \\
\mathsf{let}\ x = (\mathsf{match}\ x'\ \{\langle y, z \rangle \Rightarrow e\});\ e' &\rightsquigarrow_3 \mathsf{match}\ x'\ \{\langle y, z \rangle \Rightarrow \mathsf{let}\ x = e;\ e'\} \\
\mathsf{let}\ x = (\mathsf{match}\ y\ \{\});\ e' &\rightsquigarrow_3 \mathsf{match}\ y\ \{\} \\
\mathsf{let}\ x = (\mathsf{match}\ z\ \{\mathsf{inj}_1\, z_1 \Rightarrow e_1 \mid \mathsf{inj}_2\, z_2 \Rightarrow e_2\});\ e' &\rightsquigarrow_3 \\
&\mathsf{match}\ z\ \{\mathsf{inj}_1\, z_1 \Rightarrow \mathsf{let}\ x = e_1;\ e' \mid \mathsf{inj}_2\, z_2 \Rightarrow \mathsf{let}\ x = e_2;\ e'\} \\
\mathsf{let}\ x = e;\ \mathsf{let}\ y = z(e_0);\ e' &\rightsquigarrow_3 \mathsf{let}\ y = z(\mathsf{let}\ x = e;\ e_0);\ \mathsf{let}\ x = e;\ e' \\
\mathsf{let}\ x = e;\ \mathsf{match}\ x'\ \{\langle y, z \rangle \Rightarrow e'\} &\rightsquigarrow_3 \mathsf{match}\ x'\ \{\langle y, z \rangle \Rightarrow \mathsf{let}\ x = e;\ e'\} \\
\mathsf{let}\ x = e;\ \mathsf{match}\ y\ \{\} &\rightsquigarrow_3 \mathsf{match}\ y\ \{\} \\
\mathsf{let}\ x = e;\ \mathsf{match}\ z\ \{\mathsf{inj}_1\, z_1 \Rightarrow e_1 \mid \mathsf{inj}_2\, z_2 \Rightarrow e_2\} &\rightsquigarrow_3 \\
&\mathsf{match}\ z\ \{\mathsf{inj}_1\, z_1 \Rightarrow \mathsf{let}\ x = e;\ e_1 \mid \mathsf{inj}_2\, z_2 \Rightarrow \mathsf{let}\ x = e;\ e_2\}
\end{aligned}$$

where $[y/x]e$ renames x to y in e and has the obvious definition by recursion on e.

Figure 4.3: Cut and cut elimination in IPL with proof terms

Let's focalize that logic. Focalizing a logical system is based on the study of the invertibility properties of the system's rules, and it helps to have cut for that. Conceptually, a rule is invertible if no information is lost in passing from conclusion to premises, so in bottom-up proof search for example invertible rules may be eagerly applied without losing provability.

Definition 4.3 (Invertible Rule)**.** Given a logical system and an inference rule

$$\frac{\mathscr{J}_1 \quad \cdots \quad \mathscr{J}_n}{\mathscr{J}}\ \text{RuleName}$$

in it (where $n \geq 0$ and $\mathscr{J}_1, \ldots, \mathscr{J}_n$ are all premises, main or secondary), RuleName is *invertible* if its conclusion $\mathscr{J}$ implies (within the given system) each premise $\mathscr{J}_k$.

For example, the rule $\to$R (Fig. 4.1) is invertible, which we prove by assuming its conclusion $\Gamma \vdash A \to B$ and using the system (and its admissible rules) to derive its premise $\Gamma, A \vdash B$:

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash A \to B}}{\Gamma, A \vdash A \to B}\ \text{(Weaken)} \qquad \dfrac{\dfrac{}{\Gamma, A, A \to B \vdash A}\ \text{Hyp} \qquad \dfrac{}{\Gamma, A, A \to B, B \vdash B}\ \text{Hyp}}{\Gamma, A, A \to B \vdash B}\ \to\text{L}}{\Gamma, A \vdash B}\ \text{(Cut)}$$

However, $+$R is not invertible, which we can prove with a counterexample:

$$\frac{}{A_1 + A_2 \vdash A_1 + A_2}\ \text{Hyp}$$

but $A_1 + A_2 \not\vdash A_1$ and $A_1 + A_2 \not\vdash A_2$.

Lemma 4.5 (Invertibility in Fig. 4.1)**.**

(1) The following rules are invertible in Fig. 4.1: $\to R$, $0L$, $+L$, $1R$, $\times L$, $\times R$.

(2) The following rules are not invertible in Fig. 4.1: $\to L$, $+R$.

Proof. (1) Rule $\to$R was shown above to be invertible. Rules 0L and 1R are vacuously invertible. Rule $+$L is shown to be invertible using Hyp, (Exchange), (Cut), $+$R. Rule $\times$L is shown to be invertible using Hyp, (Cut), and $\times$R. Rule $\times$R is shown to be invertible using Hyp, (Cut), and $\times$L.

(2) Above, we have given a counterexample to the invertibility of $+$R. A counterexample to the invertibility of $\to$L is $0 \to 1 \vdash 0 \to 1$. □

4.2 A Strongly Focused Sequent Calculus for Intuitionistic Proof Search

Andreoli [1992] discovered we can eagerly apply not only invertible rules, but also non-invertible rules, without losing provability. Based on this insight, and following the proof-search strategy of Espírito Santo [2017], in Fig. 4.4, we give a strongly focalized variant of Fig. 4.1. We call the stages in which invertible rules are eagerly applied, *inversion stages*, and the stages in which non-invertible rules are eagerly applied, *focusing stages*. We call the formulas with left-invertible rules, *positive*, and the formulas with right-invertible rules, *negative*. For the purpose of shifting between these stages, we add two new logical connectives, the upshift $\uparrow P$ of a positive formula and the downshift $\downarrow N$ of a negative formula. Focused proof-search only reduces the search space for proofs, it does not eliminate the need to make choices and possibly backtrack. The main stage where these choices are

made is called the stage of *stability*. We organize these five stages in judgments of Fig. 4.4. The main differences between Fig. 4.4 and the strongly focalized intuitionistic logic $\lambda_G^{\pm}$ of Espírito Santo [2017] are listed next.

(1) We omit atomic formulas.

(2) We omit negative conjunction.

(3) We add positive conjunction.

(4) We permit only positive formulas in contexts.

There are secondary differences involving notation. While Espírito Santo [2017] puts formulas under square brackets only when they are under focus (are the principal formula of a focusing stage), we put principal formulas that are to the left of the turnstile under square brackets. Since positive formulas, for example, are defined as those with left-invertible rules, $\Gamma; [P] \vdash A$ is the left-*inversion* stage, not the left-focusing stage, which is written $\Gamma; [N] \vdash A$. One can remember what stage we are in by looking at what side of the turnstile the principal formula is on, what the polarity of the formula is (based on its syntax), and recalling what the definition of polarity is (if positive then left-invertible else right-invertible).

At a high level, (bottom-up) proof search in Fig. 4.4 works as follows. Given a context Γ and a formula A as inputs, we always begin our search for a proof of A under Γ in the stable stage: $\Gamma \Vdash A$. We can either focus on the right (if $A = P$), or on the left (if there is a $\downarrow N$ in Γ). In either case, we may succeed or pass to an inversion stage, and then possibly back to stability. Besides focusing on the left or the right, there is exactly one other option in the stable stage, which is to *jump* to the right-inversion stage. Some backtracking may be needed, but only within certain stages, either right-focusing (due to disjunction) or stability (due to choices of what to focus on). Following (our understanding of) Espírito Santo

$$
\begin{array}{lrcl}
\text{Formulas} & A,B,C & ::= & P \mid N \\
\text{Positive formulas} & P,Q,R & ::= & 0 \mid P+P \mid 1 \mid P \times P \mid \downarrow N \\
\text{Negative formulas} & N,M,L & ::= & P \to N \mid \uparrow P \\
\text{Contexts} & \Gamma & ::= & \cdot \mid \Gamma, \downarrow N
\end{array}
$$

$\boxed{\Gamma \Vdash A}$ Stability

$$\frac{\Gamma \vdash N}{\Gamma \Vdash N} \qquad \frac{\Gamma \vdash P}{\Gamma \Vdash P} \qquad \frac{\Gamma; [M] \vdash A}{\Gamma \ni \downarrow M \Vdash A}$$

$\boxed{\Gamma \vdash P}$ Right focusing

$$\frac{\Gamma \ni P}{\Gamma \vdash P} \qquad \frac{}{\Gamma \vdash 1} \qquad \frac{\Gamma \vdash P_1 \qquad \Gamma \vdash P_2}{\Gamma \vdash P_1 \times P_2} \qquad \frac{k \in \{1,2\} \qquad \Gamma \vdash P_k}{\Gamma \vdash P_1 + P_2} \qquad \frac{\Gamma \vdash N}{\Gamma \vdash \downarrow N}$$

$\boxed{\Gamma; [N] \vdash A}$ Left focusing

$$\frac{\Gamma \vdash P \qquad \Gamma; [N] \vdash A}{\Gamma; [P \to N] \vdash A} \qquad \frac{\Gamma; [P] \vdash A}{\Gamma; [\uparrow P] \vdash A}$$

$\boxed{\Gamma \vdash N}$ Right inversion (strong)

$$\frac{\Gamma; [P] \vdash N}{\Gamma \vdash P \to N} \qquad \frac{\Gamma \Vdash P}{\Gamma \vdash \uparrow P}$$

$\boxed{\Gamma; [\overleftarrow{P}] \vdash A}$ Left inversion (strong)

$$\frac{\Gamma \Vdash A}{\Gamma; [\cdot] \vdash A} \qquad \frac{\Gamma, \downarrow N; [\overleftarrow{P}] \vdash A}{\Gamma; [\downarrow N, \overleftarrow{P}] \vdash A}$$

$$\frac{}{\Gamma; [0, \overleftarrow{P}] \vdash A} \qquad \frac{\Gamma; [P_1, \overleftarrow{P}] \vdash A \qquad \Gamma; [P_2, \overleftarrow{P}] \vdash A}{\Gamma; [P_1 + P_2, \overleftarrow{P}] \vdash A}$$

$$\frac{\Gamma; [\overleftarrow{P}] \vdash A}{\Gamma; [1, \overleftarrow{P}] \vdash A} \qquad \frac{\Gamma; [P_1, P_2, \overleftarrow{P}] \vdash A}{\Gamma; [P_1 \times P_2, \overleftarrow{P}] \vdash A}$$

Figure 4.4: Strongly focalized intuitionistic logic

Contexts	Γ	$::= \cdot \mid \Gamma, x : \downarrow N$
Stable expressions	t	$::= \mathtt{dlv}\, e \mid \mathtt{ret}\, v \mid x \,\$\, s$
Values	v	$::= x \mid \langle\rangle \mid \langle v, v\rangle \mid \mathtt{inj}_1\, v \mid \mathtt{inj}_2\, v \mid \{e\}$
Spines	s	$::= \mathtt{match}\, r \mid v\, s$
Expressions	e	$::= \lambda\, r \mid \lceil t \rceil$
Arms	r	$::= \Rightarrow t \mid y\, r \mid \mathtt{abort} \mid [r_1 \mid r_2] \mid \langle\rangle\, r \mid \times\, r$

$\boxed{\Gamma \Vdash t : A}$ Stability

$$\frac{\Gamma \vdash e : N}{\Gamma \Vdash \mathtt{dlv}\, e : N} \qquad \frac{\Gamma \vdash v : P}{\Gamma \Vdash \mathtt{ret}\, v : P} \qquad \frac{\Gamma; [M] \vdash s : A}{\Gamma \ni (x : \downarrow M) \Vdash x \,\$\, s : A}$$

$\boxed{\Gamma \vdash v : P}$ Right focusing

$$\frac{\Gamma \ni (x : P)}{\Gamma \vdash x : P} \qquad \frac{}{\Gamma \vdash \langle\rangle : 1} \qquad \frac{\Gamma \vdash v_1 : P_1 \qquad \Gamma \vdash v_2 : P_2}{\Gamma \vdash \langle v_1, v_2\rangle : P_1 \times P_2}$$

$$\frac{k \in \{1,2\} \qquad \Gamma \vdash v : P_k}{\Gamma \vdash \mathtt{inj}_k\, v : P_1 + P_2} \qquad \frac{\Gamma \vdash e : N}{\Gamma \vdash \{e\} : \downarrow N}$$

$\boxed{\Gamma; [N] \vdash s : A}$ Left focusing

$$\frac{\Gamma \vdash v : P \qquad \Gamma; [N] \vdash s : A}{\Gamma; [P \to N] \vdash v\, s : A} \qquad \frac{\Gamma; [P] \vdash r : A}{\Gamma; [\uparrow P] \vdash \mathtt{match}\, r : A}$$

$\boxed{\Gamma \vdash e : N}$ Right inversion (strong)

$$\frac{\Gamma; [P] \vdash r : N}{\Gamma \vdash \lambda\, r : P \to N} \qquad \frac{\Gamma \Vdash t : P}{\Gamma \vdash \lceil t \rceil : \uparrow P}$$

$\boxed{\Gamma; [\overleftarrow{P}] \vdash r : A}$ Left inversion (strong)

$$\frac{\Gamma \Vdash t : A}{\Gamma; [\cdot] \vdash \Rightarrow t : A} \qquad \frac{\Gamma, y : \downarrow N; [\overleftarrow{P}] \vdash r : A}{\Gamma; [\downarrow N, \overleftarrow{P}] \vdash y\, r : A}$$

$$\frac{}{\Gamma; [0, \overleftarrow{P}] \vdash \mathtt{abort} : A} \qquad \frac{\Gamma; [P_1, \overleftarrow{P}] \vdash r_1 : A \qquad \Gamma; [P_2, \overleftarrow{P}] \vdash r_2 : A}{\Gamma; [P_1 + P_2, \overleftarrow{P}] \vdash [r_1 \mid r_2] : A}$$

$$\frac{\Gamma; [\overleftarrow{P}] \vdash r : A}{\Gamma; [1, \overleftarrow{P}] \vdash \langle\rangle\, r : A} \qquad \frac{\Gamma; [P_1, P_2, \overleftarrow{P}] \vdash r : A}{\Gamma; [P_1 \times P_2, \overleftarrow{P}] \vdash \times\, r : A}$$

Figure 4.5: Strongly focalized intuitionistic logic with proof terms

Stable expressions $t ::= \cdots \mid \langle e \mid s \rangle \mid \langle v \mid r \rangle$

$$\frac{\Gamma \vdash e : N \qquad \Gamma ; [N] \vdash s : A}{\Gamma \Vdash \langle e \mid s \rangle : A} \qquad \frac{\Gamma \vdash v : P \qquad \Gamma ; [P] \vdash r : A}{\Gamma \Vdash \langle v \mid r \rangle : A}$$

$$\frac{\Gamma ; [\overleftarrow{P}] \vdash r : N \qquad \Gamma ; [N] \vdash s : A}{\Gamma ; [\overleftarrow{P}] \vdash r \mathbin{@} s : A} \qquad \frac{\Gamma \Vdash t : N \qquad \Gamma ; [N] \vdash s : A}{\Gamma \Vdash t \mathbin{@} s : A}$$

$$\frac{\Gamma ; [N'] \vdash s' : N \qquad \Gamma ; [N] \vdash s : A}{\Gamma ; [N'] \vdash s' \mathbin{@} s : A} \qquad \frac{\Gamma \vdash v : P \qquad \Gamma ; [P, \overleftarrow{P}] \vdash r : A}{\Gamma ; [\overleftarrow{P}] \vdash \langle v \mid r \rangle' : A}$$

$$\frac{\Gamma \vdash v : {\downarrow}N \qquad \Gamma, x : {\downarrow}N \Vdash t : A}{\Gamma \Vdash [v/x]t : A} \qquad \frac{\Gamma \Vdash t : P \qquad \Gamma ; [P] \vdash r : A}{\Gamma \Vdash t[\star \backslash r] : A} \quad \cdots$$

$$\begin{aligned}
\langle \lambda\, r \mid v\, s \rangle &\rightsquigarrow \langle v \mid r \mathbin{@} s \rangle \\
\langle \lceil t \rceil \mid \mathtt{match}\, r \rangle &\rightsquigarrow t[\star \backslash r] \\
\langle x \mid r \rangle &\rightsquigarrow x \,\$\, \mathtt{match}\, r \\
\langle \langle\rangle \mid \langle\rangle \Rightarrow t \rangle &\rightsquigarrow t \\
\langle \langle v_1, v_2 \rangle \mid \times\, r \rangle &\rightsquigarrow \langle v_1 \mid \langle v_2 \mid r \rangle' \rangle \\
\langle v \mid y \Rightarrow t \rangle' &\rightsquigarrow [v/y]t \\
\langle \mathtt{inj}_k\, v \mid [r_1 \mid r_2] \rangle &\rightsquigarrow \langle v \mid r_k \rangle \\
\langle \langle\rangle \mid \langle\rangle\ r \rangle' &\rightsquigarrow r \\
\langle \langle v_1, v_2 \rangle' \mid \times\, r \rangle &\rightsquigarrow \langle v_2 \mid \langle v_1 \mid r \rangle' \rangle' \\
\langle \mathtt{inj}_k\, v \mid [r_1 \mid r_2] \rangle' &\rightsquigarrow \langle v \mid r_k \rangle' \\
\langle x \mid \mathtt{abort} \rangle' &= \mathtt{abort} \\
\langle v \mid y\, r \rangle' &\rightsquigarrow [v/y]r
\end{aligned}$$

$$\begin{aligned}
(\Rightarrow t) \mathbin{@} s &= \Rightarrow (t \mathbin{@} s) \\
\mathtt{abort} \mathbin{@} s &= \mathtt{abort} \\
[r_1 \mid r_2] \mathbin{@} s &= [r_1 \mathbin{@} s \mid r_2 \mathbin{@} s] \\
(\langle\rangle\ r) \mathbin{@} s &= \langle\rangle\ (r \mathbin{@} s) \\
(\times\, r) \mathbin{@} s &= \times\, (r \mathbin{@} s) \\
(y\, r) \mathbin{@} s &= y\, (r \mathbin{@} s) \\
(\mathtt{dlv}\, e) \mathbin{@} s &= \langle e \mid s \rangle \\
(x \,\$\, s') \mathbin{@} s &= x \,\$\, (s' \mathbin{@} s) \\
\langle e \mid s' \rangle \mathbin{@} s &= \langle e \mid s' \mathbin{@} s \rangle \\
\langle v \mid r \rangle \mathbin{@} s &= \langle v \mid r \mathbin{@} s \rangle \\
(v\, s') \mathbin{@} s &= v\, (s' \mathbin{@} s) \\
(\mathtt{match}\, r) \mathbin{@} s &= \mathtt{match}\, (r \mathbin{@} s)
\end{aligned}$$

$$\begin{aligned}
[v/x](\Rightarrow t) &= (\Rightarrow [v/x]t) \\
[v/x]\mathtt{abort} &= \mathtt{abort} \\
[v/x][r_1 \mid r_2] &= [[v/x]r_1 \mid [v/x]r_2] \\
[v/x](\langle\rangle\ r) &= \langle\rangle\ [v/x]r \\
[v/x](\times\, r) &= \times\, ([v/x]r) \\
[v/x](y\, r) &= y\, ([v/x]r) \\
[v/x](\mathtt{dlv}\, e) &= \mathtt{dlv}\, [v/x]e \\
[v/x](\mathtt{ret}\, v') &= \mathtt{ret}\, [v/x]v' \\
[v/x](x \,\$\, s) &= \begin{cases} \langle e \mid [\{e\}/x]s \rangle & \text{if } v = \{e\} \\ y \,\$\, [y/x]s & \text{if } v = y \end{cases} \\
[v/x](y \,\$\, s) &= y \,\$\, [v/x]s \quad (\text{if } x \neq y) \\
[v/x]\langle e \mid s \rangle &= \langle [v/x]e \mid [v/x]s \rangle \\
[v/x]\langle v \mid r \rangle &= \langle [v/x]v \mid [v/x]r \rangle \\
&\vdots \\
(\mathtt{dlv}\, e)[\star \backslash r] &= \mathtt{dlv}\, e[\star \backslash r] \\
(\mathtt{ret}\, v)[\star \backslash r] &= \langle v \mid r \rangle \\
(x \,\$\, s)[\star \backslash r] &= x \,\$\, s[\star \backslash r] \\
\langle e \mid s \rangle[\star \backslash r] &= \langle e[\star \backslash r] \mid s[\star \backslash r] \rangle \\
\langle v \mid r' \rangle[\star \backslash r] &= \langle v[\star \backslash r] \mid r'[\star \backslash r] \rangle \\
&\vdots
\end{aligned}$$

Figure 4.6: Cut elimination in strong FIPL

[2017] we prioritize the focusing stage, and so we do not restrict the right-hand side of $\Gamma \Vdash A$ or $\Gamma; [N] \vdash A$ or $\Gamma; [P] \vdash A$ to positive formulas: right-introduction for implication can result in a negative on the right of the stable stage, so if we were to make this restriction then we would have to jump to right inversion even if there were a negative on which we would prefer to focus. We prioritize focusing but nondeterministically jumping to (right) inversion does not affect provability.

It is straightforward to prove Fig. 4.4 is sound relative to Fig. 4.1 in that erasing all polarity shifts from formulas in a (true) focalized judgment yields a (true) defocalized judgment. The proof uses the fact that weakening is admissible in Fig. 4.4.

Lemma 4.6 (Strong FIPL Weaken)**.** *Suppose B is $\downarrow M$.*

(1) If $\Gamma \Vdash A$ then $\Gamma, B \Vdash A$.

(2) If $\Gamma \vdash A$ then $\Gamma, B \vdash A$.

(3) If $\Gamma; [N] \vdash A$ then $\Gamma, B; [N] \vdash A$.

(4) If $\Gamma; [\overleftarrow{P}] \vdash A$ then $\Gamma, B; [\overleftarrow{P}] \vdash A$.

Moreover, none of the derivations change in structure or height.

Proof. By mutual induction on the structure of the given derivation. □

Definition 4.4 (Defocalize)**.** Given A in Fig. 4.4, define $|A|$ to erase all shifts in A and obtain a formula in Fig. 4.1. Given Γ in Fig. 4.4, define $|\Gamma|$ to erase all shifts from all formulas P in Γ and obtain a context in Fig. 4.1.

Theorem 4.1 (Strong FIPL Sound)**.**

(1) If $\Gamma \Vdash A$ in Fig. 4.4 then $|\Gamma| \vdash |A|$ in Fig. 4.1.

(2) If $\Gamma \vdash A$ in Fig. 4.4 then $|\Gamma| \vdash |A|$ in Fig. 4.1.

(3) If $\Gamma; [N] \vdash C$ in Fig. 4.4 and $|N| \in |\Gamma|$ then $|\Gamma| \vdash |C|$ in Fig. 4.1.

(4) If $\Gamma; [\overleftarrow{P}] \vdash C$ in Fig. 4.4 and $|\overleftarrow{P}| \subseteq |\Gamma|$ then $|\Gamma| \vdash |C|$ in Fig. 4.1.

Proof. By mutual induction on the structure of the given proof, using Lemma 4.6 (Strong FIPL Weaken), Lemma 4.1 (Exchange Admissible), and Lemma 4.3 (Contract Admissible) as needed. □

We want to ensure Fig. 4.4 is logically well-behaved in that cut is admissible in it. We provide proof terms in Fig. 4.5, as well as proof terms for cuts and rules and metaoperations for eliminating them in Fig. 4.6. In the cut elimination figure(s), we separate via a horizontal line admissible rules pertaining to proof terms for the main cuts on the one hand, and on the other, spine appending $- \mathbin{@} -$ and substitution $[-]-$ and $-[-]$ metaoperations where the latter namely $\mathscr{E}[\star \backslash r]$ substitutes r for what Espírito Santo [2017] calls the implicit positive covariables $\star$ in $\mathscr{E}$ (here, intuitively, the occurrences of $\mathtt{ret}\, v$ in $\mathscr{E}$ become $\langle v \mid \star \rangle$ and then $\star$ gets replaced by r). Our $[v/x]-$ (dual to $-[\star \backslash r]$) is similar to the negative substitution of Espírito Santo [2017] but it has to unthunk $v = \{e\}$ (remove the positive curly braces around the negative expression e) and intuitively produces cuts $\langle e \mid [v/x]s \rangle$ at occurrences of *coreturns* $x \,\$\, s$ in $-$. In addition to appending spines the operation $- \mathbin{@} s$ can perhaps also be viewed as substituting s for occurrences of an implicit negative variable (?)[5], or occurrences of $\mathtt{dlv}\, e$ in $-$ producing cuts $\langle e \mid s \rangle$. We avoid explaining all the (other) proof terms, saving that activity for explaining our programming language later. However, note that Γ is now a *set* of variable typings, not a *multi*set, because variables may only appear at most once in a context (in this thesis, this is the case for all contexts with variables).

[5] Espírito Santo [2017] does not say this but it seems in line with his terminology (at least to me).

Lemma 4.7 (Strong FIPL Cut Admissible)**.**

The cut rules in Fig. 4.6 are admissible in Fig. 4.5.

Proof. Fig. 4.6 provides cut elimination relations similar to (perhaps simpler than) those of Espírito Santo [2017]. Whenever inductively going from the left-hand side of the $\rightsquigarrow$ (or $=$) to the right-hand side of the $\rightsquigarrow$ (or $=$), either the structural size of the cut formula decreases; or else the cut formula stays the same but the part size decreases (a part here is one of the five kinds of cuts displayed in Fig. 4.6), where

$$[v/x]{-_1} \quad < \quad \langle _ \,|\, _ \rangle' \quad < \quad \langle _ \,|\, _ \rangle \quad < \quad -_1 \mathbin{@} s \quad = \quad -_1[\star \backslash r]$$

or else the cut formula stays the same, the part size stays the same, and the proof term in the $-_1$ position gets smaller. □

We prove that the strongly focused logic is complete by a particular polarization strategy, defined in Fig. 4.7.

Theorem 4.2 (Strong FIPL Complete)**.**

If $\Gamma \vdash A$ in Fig. 4.1 then $pol^+(\Gamma) \Vdash pol^+(A)$ in Fig. 4.4.

Proof. It suffices to prove (by structural induction on e) that

if $\Gamma \vdash e : A$ in Fig. 4.2 then $pol^+(\Gamma) \Vdash pol^+(e) : pol^+(A)$ in Fig. 4.5

where $pol^+(-)$ is defined in Fig. 4.7.

Use Lemma 4.6 (Strong FIPL Weaken) in the $\rightarrow$ right case. □

4.3 A Weakly Focused Intuition. Sequent Calculus, The Core of Our PL

That concludes our brief review of focusing for proof search by way of an example which we will now transform into our refinement type system. We consider a series of small

Define the positive and negative polarization and $pol^+(A)$ and $pol^-(A)$ mutually by recursion on A:

$$
\begin{aligned}
pol^-(0) &= \uparrow 0 \\
pol^-(A+B) &= \uparrow(pol^+(A) + pol^+(B)) \\
pol^-(1) &= \uparrow 1 \\
pol^-(A \times B) &= \uparrow(pol^+(A) \times pol^+(B)) \\
pol^-(A \to B) &= \downarrow\uparrow pol^+(A) \to \uparrow pol^+(B) \\
pol^+(A) &= \downarrow pol^-(A)
\end{aligned}
$$

$$
\begin{aligned}
pol^+(x) &= \mathtt{ret}\,\{\lceil x \rceil\} \\
pol^+(\langle\rangle) &= \mathtt{ret}\,\{\lceil \mathtt{ret}\,\langle\rangle \rceil\} \\
pol^+(\langle e_1, e_2\rangle) &= \mathtt{ret}\,\{\lceil \mathtt{ret}\,\langle \{\lceil pol^+(e_1)\rceil\}, \{\lceil pol^+(e_2)\rceil\}\rangle \rceil\} \\
pol^+(\mathtt{inj}_k\, e) &= \mathtt{ret}\,\{\lceil \mathtt{ret}\,(\mathtt{inj}_k\,\{\lceil pol^+(e)\rceil\}) \rceil\} \\
pol^+(\lambda x.e) &= \mathtt{ret}\,\{\lambda\, x' \Rightarrow x'\,\$\,\mathtt{match}\, x \Rightarrow \mathtt{dlv}\,\lceil pol^+(e)\rceil\} \\
&\qquad \text{where } x' \notin FV(e) \cup BV(e) \cup \{x\} \ (x' \text{ is fresh}) \\
pol^+(\mathtt{let}\, y{=}x(e);\, e') &= x\,\$\,\{\lceil pol^+(e)\rceil\}\,\mathtt{match}\, y \Rightarrow pol^+(e') \\
pol^+(\mathtt{match}\, x\, \{\langle y, z\rangle \Rightarrow e\}) &= x\,\$\,\mathtt{match} \times y\, z \Rightarrow pol^+(e) \\
pol^+(\mathtt{match}\, x\, \{\}) &= x\,\$\,\mathtt{match}\ \mathtt{abort} \\
pol^+(\mathtt{match}\, x\, \{\mathtt{inj}_1 x_1 \Rightarrow e_1 \mid \mathtt{inj}_2 x_2 \Rightarrow e_2\}) &= x\,\$\,\mathtt{match}\ [x_1 \Rightarrow pol^+(e_1) \mid x_2 \Rightarrow pol^+(e_2)]
\end{aligned}
$$

Figure 4.7: A polarization of IPL into strong FIPL

systemic transformations toward our unrefined type system for a standard functional programming language, on top of which we will build a modular index refinement layer.

As such, our aim now is to put the programmer, not the proof search engine, in a position of stability. (We give a series of transformations but only really care about the end result, so don't worry about understanding each step in detail.) The programmer chooses what to focus on and what to invert, that is, when to use pattern-matching, when to sequence a computation, and so on. The main thing programmers build are program *expressions*

(expressions of computations), which have negative type. We therefore restrict stability to negative formulas in our first step, Fig. 4.8.

Next, in Fig. 4.10, given near the end of this chapter, we weaken the inversion stages, as programmers tend to choose when to pattern-match. Since we are weakening left-inversion, and because stability is merely negative, for simplicity we might as well combine the negative stability and right-inversion stages (thereby weakening right-inversion). As mentioned in the background section, this makes the resulting system weakly focused.

Continuing, we add proof terms in Fig. 4.11. Already these proof terms should look more familiar to functional programmers. But we can still do better. So, next, in Fig. 4.12, we turn spines into mere lists of values, and transfer the old continuations of spines to let-binding (like pattern-matching, let-binding is usually handled by programmers) in the combined right-inversion/stability stage. We present cut elimination in Fig. 4.13: note that $\rightsquigarrow^+$ means a sequence of one or more $\rightsquigarrow$s. The former uses the operation $s@s'$ for composing spines, but in the latter this is replaced by $\mathtt{let}\ y = \{e\}\,(s);\ e'$ reductions. In both, substituting a thunk $\{e\}$ for the head x of a coreturn $x(s)$ looks kind of like a *hereditary substitution* in that the result of substitution is defined to "reduce", well, not really to reduce, but to be a cut term: $\langle e \mid [\{e\}/x]s \rangle$. To actually reduce the cut, the relation $\rightsquigarrow$ must be "applied" (defining $[\{e\}/x](x\$s) = e'$ if $\langle e \mid [\{e\}/x]s \rangle \rightsquigarrow^* e'$ would literally be a hereditary substitution as usually understood [Watkins et al., 2004, Pfenning, 2008]). (The operation $-^*$ takes the reflexive transitive closure of a relation; in this case the set of the relation is the language generated by the grammar for proof terms.)

In Fig. 4.13, whenever inductively going from the left-hand side of the $\rightsquigarrow$ (or $=$) to the right-hand side of the $\rightsquigarrow$ (or $=$), either the structural size of the cut formula decreases; or else the cut formula stays the same but the part size decreases (a part here is one of the

$$
\begin{array}{lrcl}
\text{Formulas} & A,B,C & ::= & P \mid N \\
\text{Positive formulas} & P,Q,R & ::= & 0 \mid P+P \mid 1 \mid P \times P \mid \downarrow N \\
\text{Negative formulas} & N,M,L & ::= & P \to N \mid \uparrow P \\
\text{Contexts} & \Gamma & ::= & \cdot \mid \Gamma, \downarrow N
\end{array}
$$

$\boxed{\Gamma \Vdash N}$ Negative stability

$$\frac{\Gamma \vdash N}{\Gamma \Vdash N} \qquad \frac{\Gamma; [M] \vdash N}{\Gamma \ni \downarrow M \Vdash N}$$

$\boxed{\Gamma \vdash P}$ Right focusing

$$\frac{\Gamma \ni P}{\Gamma \vdash P} \qquad \frac{}{\Gamma \vdash 1} \qquad \frac{\Gamma \vdash P_1 \qquad \Gamma \vdash P_2}{\Gamma \vdash P_1 \times P_2} \qquad \frac{k \in \{1,2\} \qquad \Gamma \vdash P_k}{\Gamma \vdash P_1 + P_2} \qquad \frac{\Gamma \vdash N}{\Gamma \vdash \downarrow N}$$

$\boxed{\Gamma; [N] \vdash M}$ Left focusing

$$\frac{\Gamma \vdash P \qquad \Gamma; [N] \vdash M}{\Gamma; [P \to N] \vdash M} \qquad \frac{\Gamma; [P] \vdash M}{\Gamma; [\uparrow P] \vdash M}$$

$\boxed{\Gamma \vdash N}$ Right inversion (strong)

$$\frac{\Gamma; [P] \vdash N}{\Gamma \vdash P \to N} \qquad \frac{\Gamma \vdash P}{\Gamma \vdash \uparrow P}$$

$\boxed{\Gamma; [\overleftarrow{P}] \vdash M}$ Left inversion (strong)

$$\frac{\Gamma \Vdash M}{\Gamma; [\cdot] \vdash M} \qquad \frac{\Gamma, \downarrow N; [\overleftarrow{P}] \vdash M}{\Gamma; [\downarrow N, \overleftarrow{P}] \vdash M}$$

$$\frac{}{\Gamma; [0, \overleftarrow{P}] \vdash M} \qquad \frac{\Gamma; [P_1, \overleftarrow{P}] \vdash M \qquad \Gamma; [P_2, \overleftarrow{P}] \vdash M}{\Gamma; [P_1 + P_2, \overleftarrow{P}] \vdash M}$$

$$\frac{\Gamma; [\overleftarrow{P}] \vdash M}{\Gamma; [1, \overleftarrow{P}] \vdash M} \qquad \frac{\Gamma; [P_1, P_2, \overleftarrow{P}] \vdash M}{\Gamma; [P_1 \times P_2, \overleftarrow{P}] \vdash M}$$

Figure 4.8: Strongly focalized intuitionistic logic with negative stability

Contexts	Γ	$::= \cdot \mid \Gamma, x : \downarrow N$
Stable expressions	t	$::= \mathtt{dlv}\, e \mid x \$ s$
Values	v	$::= x \mid \langle\rangle \mid \langle v, v\rangle \mid \mathtt{inj}_1\, v \mid \mathtt{inj}_2\, v \mid \{e\}$
Spines	s	$::= \mathtt{match}\, r \mid v\, s$
Expressions	e	$::= \lambda\, r \mid \mathtt{return}\, v$
Arms	r	$::= \Rightarrow t \mid x\, r \mid \mathtt{abort} \mid [r_1 \mid r_2] \mid \langle\rangle\, r \mid \times\, r$

$\boxed{\Gamma \Vdash t : N}$ Negative stability

$$\frac{\Gamma \vdash e : N}{\Gamma \Vdash \mathtt{dlv}\, e : N} \qquad \frac{\Gamma; [M] \vdash s : N}{\Gamma \ni s : \downarrow M \Vdash x \$ s : N}$$

$\boxed{\Gamma \vdash v : P}$ Right focusing

$$\frac{\Gamma \ni x : P}{\Gamma \vdash x : P} \qquad \frac{}{\Gamma \vdash \langle\rangle : 1} \qquad \frac{\Gamma \vdash v_1 : P_1 \qquad \Gamma \vdash v_2 : P_2}{\Gamma \vdash \langle v_1, v_2\rangle : P_1 \times P_2} \qquad \frac{k \in \{1,2\} \qquad \Gamma \vdash v : P_k}{\Gamma \vdash \mathtt{inj}_k\, v : P_1 + P_2}$$

$$\frac{\Gamma \vdash e : N}{\Gamma \vdash \{e\} : \downarrow N}$$

$\boxed{\Gamma; [N] \vdash s : M}$ Left focusing

$$\frac{\Gamma \vdash v : P \qquad \Gamma; [N] \vdash s : M}{\Gamma; [P \to N] \vdash v\, s : M} \qquad \frac{\Gamma; [P] \vdash r : M}{\Gamma; [\uparrow P] \vdash \mathtt{match}\, r : M}$$

$\boxed{\Gamma \vdash e : N}$ Right inversion (strong)

$$\frac{\Gamma; [P] \vdash r : N}{\Gamma \vdash \lambda\, r : P \to N} \qquad \frac{\Gamma \vdash v : P}{\Gamma \vdash \mathtt{return}\, v : \uparrow P}$$

$\boxed{\Gamma; [\overleftarrow{P}] \vdash r : M}$ Left inversion (strong)

$$\frac{\Gamma \Vdash t : M}{\Gamma; [\cdot] \vdash \Rightarrow t : M} \qquad \frac{\Gamma, \downarrow N; [\overleftarrow{P}] \vdash r : M}{\Gamma; [\downarrow N, \overleftarrow{P}] \vdash x\, r : M}$$

$$\frac{}{\Gamma; [0, \overleftarrow{P}] \vdash \mathtt{abort} : M} \qquad \frac{\Gamma; [P_1, \overleftarrow{P}] \vdash r_1 : M \qquad \Gamma; [P_2, \overleftarrow{P}] \vdash r_2 : M}{\Gamma; [P_1 + P_2, \overleftarrow{P}] \vdash [r_1 \mid r_2] : M}$$

$$\frac{\Gamma; [\overleftarrow{P}] \vdash r : M}{\Gamma; [1, \overleftarrow{P}] \vdash \langle\rangle\, r : M} \qquad \frac{\Gamma; [P_1, P_2, \overleftarrow{P}] \vdash r : M}{\Gamma; [P_1 \times P_2, \overleftarrow{P}] \vdash \times\, r : M}$$

Figure 4.9: Strongly focalized intuitionistic logic with negative stability (proof terms)

$$
\begin{array}{lrcl}
\text{Formulas} & A,B,C & ::= & P \mid N \\
\text{Positive formulas} & P,Q,R & ::= & 0 \mid P+P \mid 1 \mid P \times P \mid \downarrow N \\
\text{Negative formulas} & N,M,L & ::= & P \to N \mid \uparrow P \\
\text{Contexts} & \Gamma & ::= & \cdot \mid \Gamma, P
\end{array}
$$

$\boxed{\Gamma \vdash P}$ Right focusing

$$\frac{\Gamma \ni P}{\Gamma \vdash P} \qquad \frac{}{\Gamma \vdash 1} \qquad \frac{\Gamma \vdash P_1 \qquad \Gamma \vdash P_2}{\Gamma \vdash P_1 \times P_2} \qquad \frac{k \in \{1,2\} \qquad \Gamma \vdash P_k}{\Gamma \vdash P_1 + P_2} \qquad \frac{\Gamma \vdash N}{\Gamma \vdash \downarrow N}$$

$\boxed{\Gamma; [N] \vdash M}$ Left focusing

$$\frac{\Gamma \vdash P \qquad \Gamma; [N] \vdash M}{\Gamma; [P \to N] \vdash M} \qquad \frac{\Gamma, P \vdash M}{\Gamma; [\uparrow P] \vdash M}$$

$\boxed{\Gamma \vdash N}$ Right inversion (weak) with stable moments

$$\frac{\Gamma, P \vdash N}{\Gamma \vdash P \to N} \qquad \frac{\Gamma \vdash P}{\Gamma \vdash \uparrow P} \qquad \frac{\Gamma; [N] \vdash M}{\Gamma \ni \downarrow N \vdash M} \qquad \frac{\Gamma; [P] \vdash M}{\Gamma \ni P \vdash M}$$

$\boxed{\Gamma; [P] \vdash M}$ Left inversion (weak)

$$\frac{}{\Gamma; [0] \vdash M} \qquad \frac{\Gamma, P_1 \vdash M \qquad \Gamma, P_2 \vdash M}{\Gamma; [P_1 + P_2] \vdash M} \qquad \frac{\Gamma, P_1, P_2 \vdash M}{\Gamma; [P_1 \times P_2] \vdash M}$$

Figure 4.10: Weakly focalized intuitionistic logic with implicit negative stability

three kinds of cuts displayed in Fig. 4.13), where

$$\underline{\texttt{match}}\ v\ \{r_i \Rightarrow e_i\}_{i \in I} \quad < \quad [v/x]{-_1} \quad < \quad \texttt{let}\ y = \{-_1\}\,(s);\ e'$$

or else the cut formula stays the same, the part size stays the same, and the number of cuts in $-_1$ decreases; or else the cut formula stays the same, the part size stays the same, the number of cuts in $-_1$ stays the same, and the proof term in the $-_1$ position gets smaller.

$$
\begin{array}{lrcl}
\text{Variables} & x,y,z & & \\
\text{Contexts} & \Gamma & ::= & \cdot \mid \Gamma, x:P \\
\text{Values} & v & ::= & x \mid \langle\rangle \mid \langle v,v\rangle \mid \mathtt{inj}_1\, v \mid \mathtt{inj}_2\, v \mid \{e\} \\
\text{Spines} & s & ::= & z.e \mid v,s \\
\text{Expressions} & e & ::= & \lambda x.e \mid \mathtt{return}\, v \mid x\,\$\,s \mid \mathtt{match}\, x\, \{r_i \Rightarrow e_i\}_{i\in I} \\
\text{Patterns} & r & ::= & \mathtt{inj}_1\, x \mid \mathtt{inj}_2\, x \mid \langle\rangle \mid \langle x_1, x_2\rangle
\end{array}
$$

$\boxed{\Gamma \vdash v : P}$ Right focusing

$$\frac{\Gamma \ni (x:P)}{\Gamma \vdash x : P} \qquad \frac{}{\Gamma \vdash \langle\rangle : 1} \qquad \frac{\Gamma \vdash v_1 : P_1 \qquad \Gamma \vdash v_2 : P_2}{\Gamma \vdash \langle v_1, v_2\rangle : P_1 \times P_2}$$

$$\frac{k \in \{1,2\} \qquad \Gamma \vdash v : P_k}{\Gamma \vdash \mathtt{inj}_k\, v : P_1 + P_2} \qquad \frac{\Gamma \vdash e : N}{\Gamma \vdash \{e\} : \downarrow N}$$

$\boxed{\Gamma; [N] \vdash s : M}$ Left focusing

$$\frac{\Gamma \vdash v : P \qquad \Gamma; [N] \vdash s : M}{\Gamma; [P \to N] \vdash v, s : M} \qquad \frac{\Gamma, z : P \vdash e : M}{\Gamma; [\uparrow P] \vdash z.e : M}$$

$\boxed{\Gamma \vdash e : N}$ Right inversion/stability (weak)

$$\frac{\Gamma, x : P \vdash e : N}{\Gamma \vdash \lambda x.e : P \to N} \qquad \frac{\Gamma \vdash v : P}{\Gamma \vdash \mathtt{return}\, v : \uparrow P}$$

$$\frac{\Gamma; [N] \vdash s : M}{\Gamma \ni (x : \downarrow N) \vdash x\,\$\,s : M} \qquad \frac{\Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} : M}{\Gamma \ni (x:P) \vdash \mathtt{match}\, x\, \{r_i \Rightarrow e_i\}_{i\in I} : M}$$

$\boxed{\Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} : M}$ Left inversion (weak)

$$\frac{}{\Gamma; [0] \vdash \{\} : M} \qquad \frac{\Gamma, x_1 : P_1 \vdash e_1 : M \qquad \Gamma, x_2 : P_2 \vdash e_2 : M}{\Gamma; [P_1 + P_2] \vdash \{\mathtt{inj}_1\, x_1 \Rightarrow e_1 \mid \mathtt{inj}_2\, x_2 \Rightarrow e_2\} : M}$$

$$\frac{\Gamma \vdash e : M}{\Gamma; [1] \vdash \{\langle\rangle \Rightarrow e\} : M} \qquad \frac{\Gamma, x_1 : P_1, x_2 : P_2 \vdash e : M}{\Gamma; [P_1 \times P_2] \vdash \{\langle x_1, x_2\rangle \Rightarrow e\} : M}$$

Figure 4.11: Weakly focalized intuitionistic logic with implicit negative stability and proof terms

We gave a series of transformations, but don't intend to prove anything about most of them. They were my way of understanding how our system related to the strongly focused proof search. We have proven that IPL and strong FIPL are equivalent, and we have proven cut elimination in both. We now prove that our unrefined type assignment system (without recursion and ADTs) is equivalent to IPL. As a corollary, we can eliminate cuts in our unrefined assignment system (without recursion). We will extend this system with recursion and ADTs. The corresponding extension of IPL to recursion and ADTs is equivalent to this system, the unrefined type assignment (non-bidirectional) system underlying our refined system (the main object of study in this thesis). We will see in the next chapter that this type assignment system with recursion is inconsistent and we cannot prove cut in it, and so therefore IPL with recursion and ADTs is also inconsistent. By adding a refinement layer, we will be able to recover consistency.

Theorem 4.3. *The system defined in Fig. 4.12 + Fig. 4.13 is equivalent to the system defined in Fig. 4.2 + Fig. 4.3.*

Proof. We define functions φ (completeness) and ψ (soundness).

$$\varphi : Fig.\ 4.2 + Fig.\ 4.3 \leftrightarrow Fig.\ 4.12 + Fig.\ 4.13 : \psi$$

Soundness is stated:

(1) If $\Gamma \vdash e : N$ then $|\Gamma| \vdash \psi(e) : |N|$.

(2) If $\Gamma \vdash v : P$ then $|\Gamma| \vdash \psi(v) : |P|$.

The function ψ is defined as follows.

$$\begin{aligned}
\psi(x) &= x \\
\psi(\langle\rangle) &= \langle\rangle \\
\psi(\langle v_1, v_2\rangle) &= \langle \psi(v_1), \psi(v_2)\rangle \\
\psi(\mathsf{inj}_k\, v) &= \mathsf{inj}_k\, \psi(v) \\
\psi(\{e\}) &= \psi(e) \\
\psi(\lambda x.e) &= \lambda x.\, \psi(e) \\
\psi(\mathtt{return}\, v) &= \psi(v) \\
\psi(\mathtt{match}\; x\; \{r_i \Rightarrow e_i\}_{i\in I}) &= \begin{cases} \psi(e) & \text{if } \{r_i \Rightarrow e_i\}_{i\in I} = \{\langle\rangle \Rightarrow e\} \\ \mathtt{match}\; x\; \{r_i \Rightarrow \psi(e_i)\}_{i\in I} & \text{else} \end{cases} \\
\psi(\mathsf{let}\; z{=}x(\cdot);\, e) &= \mathsf{let}\; z{=}x;\; \psi(e) \\
\psi(\mathsf{let}\; z{=}x(v,s);\, e) &= \mathsf{let}\; y{=}x(\psi(v));\; \psi(\mathsf{let}\; z{=}y(s);\, e) \text{ where } y \text{ is fresh} \\
\psi(\mathsf{let}\; y{=}\{e\}\,(s);\, e') &= \mathsf{let}\; x{=}\psi(e);\; \psi(\mathsf{let}\; y{=}x(s);\, e') \text{ where } x \text{ is fresh} \\
\psi(\underline{\mathtt{match}}\; v\; \{\}) &= \mathsf{let}\; x{=}\psi(v);\; \mathtt{match}\; x\; \{\} \text{ where } x \text{ is fresh} \\
\psi(\underline{\mathtt{match}}\; v\; \{\mathsf{inj}_1 x_1 \Rightarrow e_1 \mid \mathsf{inj}_2 x_2 \Rightarrow e_2\}) &= \\
&\quad \mathsf{let}\; x{=}\psi(v);\; \mathtt{match}\; x\; \{\mathsf{inj}_1 x_1 \Rightarrow \psi(e_1) \mid \mathsf{inj}_2 x_2 \Rightarrow \psi(e_2)\} \text{ where } x \text{ fresh} \\
\psi(\underline{\mathtt{match}}\; v\; \{\langle\rangle \Rightarrow e\}) &= \psi(e) \\
\psi(\underline{\mathtt{match}}\; v\; \{\langle x_1, x_2\rangle \Rightarrow e\}) &= \mathsf{let}\; x{=}\psi(v);\; \mathtt{match}\; x\; \{\langle x_1, x_2\rangle \Rightarrow \psi(e)\} \text{ where } x \text{ fresh}
\end{aligned}$$

There is no clause for $\psi([v/x]e)$ because $[v/x]e$ is a metaoperation not a proof term. One can define a structural size function on proof terms and show it always decreases at the

inductive calls of ψ. Extending ψ to the systems extended with recursion and algebraic datatypes is straightforward (the linear reader can skip to completeness and come back near the end of the next chapter):

$$\psi(\mathtt{rec}\, x.\, e) = \mathtt{rec}\, x.\, \psi(e)$$

$$\psi(\mathtt{match}\, x\, \{\mathtt{into}(y) \Rightarrow e\}) = \mathtt{match}\, x\, \{\mathtt{into}(y) \Rightarrow \psi(e)\}$$

$$\psi(\mathtt{into}(v)) = \mathtt{into}(\psi(v))$$

However, we need to prove (straightforward) that
(polarized) $\vdash G[\mu F] \stackrel{\circ}{=} P$ implies (unpolarized) $\vdash |G|[\mu|F|] \stackrel{\circ}{=} |P|$ (where this $|-|$ merely erases polarity shifts and this unrolling judgment $\stackrel{\circ}{=}$ is identical to unrefined unrolling except it uses unpolarized types).

Completeness is stated:
If $\Gamma \vdash e : A$ then $pol^+(\Gamma) \vdash \varphi(e) : \uparrow pol^+(A)$
where φ and $pol^+(A)$ and $pol^-(A)$ are defined as follows.

$$pol^-(0) = \uparrow 0$$

$$pol^-(A+B) = \uparrow(pol^+(A) + pol^+(B))$$

$$pol^-(1) = \uparrow 1$$

$$pol^-(A \times B) = \uparrow(pol^+(A) \times pol^+(B))$$

$$pol^-(A \to B) = pol^+(A) \to \uparrow pol^+(B)$$

$$pol^+(A) = \downarrow pol^-(A)$$

$$\varphi(x) = \mathtt{return}\,\{\mathtt{return}\,x\}$$

$$\varphi(\mathtt{let}\ y{=}x(e);\ e') = \mathtt{let}\ z{=}\{\varphi(e)\}\,(\cdot);\ \mathtt{let}\ y{=}x(z);\ \varphi(e') \text{ where } z \text{ is fresh}$$

$$\varphi(\lambda x.e) = \mathtt{return}\,\{\lambda x.\,\varphi(e)\}$$

$$\varphi(\langle\rangle) = \mathtt{return}\,\{\mathtt{return}\,\langle\rangle\}$$

$$\varphi(\langle e_1, e_2\rangle) =$$

$$\mathtt{let}\ x_1{=}\{\varphi(e_1)\}\,(\cdot);\ \mathtt{let}\ x_2{=}\{\varphi(e_2)\}\,(\cdot);\ \mathtt{return}\,\{\mathtt{return}\,\langle x_1, x_2\rangle\}$$

where x_1, x_2 fresh

$$\varphi(\mathtt{inj}_k\, e) = \mathtt{let}\ x{=}\{\varphi(e)\};\ \mathtt{return}\,\{\mathtt{return}\,(\mathtt{inj}_k\, x)\} \text{ where } x \text{ fresh}$$

$$\varphi(\mathtt{match}\ x\ \{\}) = \mathtt{let}\ x'{=}x(\cdot);\ \mathtt{match}\ x'\ \{\} \text{ where } x' \text{ is fresh}$$

$$\varphi(\mathtt{match}\ x\ \{\mathtt{inj}_1\, x_1 \Rightarrow e_1 \mid \mathtt{inj}_2\, x_2 \Rightarrow e_2\}) =$$

$$\mathtt{let}\ x'{=}x(\cdot);\ \mathtt{match}\ x'\ \{\mathtt{inj}_1\, x_1 \Rightarrow \varphi(e_1) \mid \mathtt{inj}_2\, x_2 \Rightarrow \varphi(e_2)\} \text{ where } x' \text{ is fresh}$$

$$\varphi(\mathtt{match}\ x\ \{\langle x_1, x_2\rangle \Rightarrow e\}) = \mathtt{let}\ x'{=}x(\cdot);\ \mathtt{match}\ x'\ \{\langle x_1, x_2\rangle \Rightarrow \varphi(e)\} \text{ where } x' \text{ fresh}$$

$$\varphi(\mathtt{let}\ x{=}e;\ e') = \mathtt{let}\ x{=}\{\varphi(e)\}\,(\cdot);\ \varphi(e')$$

Use (unstated) weakening in the $\to$ right case. We extend φ to the system with recursion and ADTs (the reader can skip for now and come back near the end of the next chapter).

We also extend the polarization strategy to ADTs:

$$pol^-(\mu F) = \uparrow \mu(pol^+(F))$$

$$\begin{aligned} pol^+(F_1 \oplus F_2) &= pol^+(F_1) \oplus pol^+(F_2) \\ pol^+(\hat{B} \otimes \hat{P}) &= pol^+(\hat{B}) \otimes pol^+(\hat{P}) \\ pol^+(\hat{I}) &= \hat{I} \\ pol^+(\underline{A}) &= \underline{pol^+(A)} \\ pol^+(\mathrm{Id}) &= \mathrm{Id} \end{aligned}$$

$$\begin{aligned} \varphi(\mathtt{rec}\, x.\, e) &= \mathtt{rec}\, x'.\, \mathtt{let}\, x = x'(\cdot);\, \varphi(e) \text{ where } x' \text{ is fresh} \\ \varphi(\mathtt{match}\, x\, \{\mathtt{into}(y) \Rightarrow e\}) &= \mathtt{let}\, x' = x(\cdot);\, \mathtt{match}\, x'\, \{\mathtt{into}(y) \Rightarrow \varphi(e)\} \text{ where } x' \text{ fresh} \\ \varphi(\mathtt{into}(v)) &= \mathtt{return}\, \mathtt{into}(\varphi(v)) \end{aligned}$$

However, again, we need to prove (straightforward) a lemma:
If (unpolarized) $\vdash G[\mu F] \doteqdot A$ then $\vdash pol^+(G)[\mu(pol^+(F))] \doteqdot pol^+(A)$. □

4.4 A Bidirectionally Typed Weakly Focused Intuitionistic Sequent Calculus

Now we combine and bidirectionalize[6] Fig. 4.12 and Fig. 4.13 to obtain our core, unrefined type system, but without inductive types and recursion: Fig. 4.14 and Fig. 4.15. We combine the expressions $\mathtt{match}\, x\, \{r_i \Rightarrow e_i\}_{i \in I}$ and $\mathtt{let}\, x = y(s);\, e$ of Fig. 4.12 with the main cuts $\underline{\mathtt{match}}\, v\, \{r_i \Rightarrow e_i\}_{i \in I}$ and $\mathtt{let}\, x = \{e'\}\,(s);\, e$ of Fig. 4.13 to obtain let-binding $\mathtt{let}\, x = g;\, e$

[6] The result is mode-correct.

and pattern-matching $\texttt{match}\ h\ \{r_i \Rightarrow e_i\}_{i \in I}$. Next, we explain h and g; the explanation has something to do with our initial bidirectionalizing strategy, which is quite simple.

In our bidirectionalization, the idea is, first, to assign all the metavariables appearing in the conclusions of rules in Fig. 4.12 and Fig. 4.13 the mode of *input*. Second, we assign all the metavariables appearing "from nowhere" in going from conclusion to premise the mode of *output*, and introduce two new syntactic categories that (to abuse language slightly) output or synthesize these metavariables (types): heads h and bound expressions g. In order for the type system to be able to output these cut types/formulas always, based on non-variable inputs h and g, the cut type must appear somewhere in these inputs, so we introduce *type annotations* $(v : P)$ and $(e : \uparrow P)$ in the grammar for h and g. A head h is either a variable x, whose type can be synthesized from the context, or an annotated value $(v : P)$. A (let-)bound expression g is either a head applied to a spine $h(s)$, whose type can be synthesized ultimately from the head h, or an annotated value-returning expression $(e : \uparrow P)$.

An operational semantics of our (bidirectional) unrefined system (pre-recursion and pre-ADT), given in Fig. 4.13, is defined in terms of its type annotation erasure, given in Fig. 4.16, into the system of Fig. 4.12 together with the two main cuts at the top of Fig. 4.13. Compare the notion of normalization in Fig. 4.13 with the one we started with in the proof of Lemma 4.4 (Cut Admissible): Fig. 4.13 looks more like a CBPV operational semantics—which is to be expected because Lemma 4.4 (Cut Admissible) is unfocalized—and perhaps looks more familiar to programmers or computer scientists.

Theorem 4.4 (Bidirectionalization Sound)**.**

(1) If $\Gamma \vdash e \Leftarrow N$ *then* $\Gamma \vdash |e| : N$.

(2) If $\Gamma \vdash v \Leftarrow P$ *then* $\Gamma \vdash |v| : P$.

(3) *If* $\Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$ *then* $\Gamma; [P] \vdash |\{r_i \Rightarrow e_i\}_{i \in I}| : N$.

(4) *If* $\Gamma; [N] \vdash s \Leftarrow M$ *then* $\Gamma; [N] \vdash |s| : M$.

Proof. By structural induction on the program term. All parts are mutually recursive. Straightforward. The hardest case is the one where $e = \texttt{let}\ x\texttt{=}g; e_0$ which requires case-analyzing subderivations $\Gamma \vdash g \Rightarrow \uparrow P$. □

Theorem 4.5 (Bidirectionalization Complete)**.**

(1) *If* $\Gamma \vdash e : N$ *then there exists* e' *such that* $|e'| = e$ *and* $\Gamma \vdash e' \Leftarrow N$.

(2) *If* $\Gamma \vdash v : P$ *then there exists* v' *such that* $|v'| = v$ *and* $\Gamma \vdash v' \Leftarrow P$.

(3) *If* $\Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} : M$
then there exists $\{r'_i \Rightarrow e'_i\}_{i \in I}$ *such that* $|\{r'_i \Rightarrow e'_i\}_{i \in I}| = \{r_i \Rightarrow e_i\}_{i \in I}$
and $\Gamma; [P] \vdash \{r'_i \Rightarrow e'_i\}_{i \in I} \Leftarrow M$.

(4) *If* $\Gamma; [N] \vdash s : M$ *then there exists* s' *such that* $|s'| = s$ *and* $\Gamma; [N] \vdash s' \Rightarrow M$.

Proof. By structural induction on the given typing derivation. All parts are mutually recursive. Straightforward. □

Equivalence of the bidirectional typing system and the type assignment system (that is, the previous two lemmas) is straightforward to extend to recursive expressions and algebraic datatypes, introduced in the next chapter.

$$
\begin{array}{lrcl}
\text{Variables} & x,y,z & & \\
\text{Contexts} & \Gamma & ::= & \cdot \mid \Gamma, x:P \\
\text{Values} & v & ::= & x \mid \langle\rangle \mid \langle v,v\rangle \mid \mathtt{inj}_1\, v \mid \mathtt{inj}_2\, v \mid \{e\} \\
\text{Spines} & s & ::= & \cdot \mid v,s \\
\text{Expressions} & e & ::= & \lambda x.e \mid \mathtt{return}\, v \mid \mathtt{let}\, z{=}x(s);\, e \mid \mathtt{match}\, x\, \{r_i \Rightarrow e_i\}_{i\in I} \\
\text{Patterns} & r & ::= & \mathtt{inj}_1\, x \mid \mathtt{inj}_2\, x \mid \langle\rangle \mid \langle x_1, x_2\rangle
\end{array}
$$

$\boxed{\Gamma \vdash v : P}$ Right focusing

$$\frac{\Gamma \ni (x:P)}{\Gamma \vdash x : P} \qquad \frac{}{\Gamma \vdash \langle\rangle : 1} \qquad \frac{\Gamma \vdash v_1 : P_1 \qquad \Gamma \vdash v_2 : P_2}{\Gamma \vdash \langle v_1, v_2\rangle : P_1 \times P_2}$$

$$\frac{k \in \{1,2\} \qquad \Gamma \vdash v : P_k}{\Gamma \vdash \mathtt{inj}_k\, v : P_1 + P_2} \qquad \frac{\Gamma \vdash e : N}{\Gamma \vdash \{e\} : {\downarrow}N}$$

$\boxed{\Gamma; [N] \vdash s : {\uparrow}P}$ Left focusing

$$\frac{\Gamma \vdash v : Q \qquad \Gamma; [N] \vdash s : {\uparrow}P}{\Gamma; [Q \to N] \vdash v,s : {\uparrow}P} \qquad \frac{}{\Gamma; [{\uparrow}P] \vdash \cdot : {\uparrow}P}$$

$\boxed{\Gamma \vdash e : N}$ Right inversion (weak)

$$\frac{\Gamma, x:P \vdash e : N}{\Gamma \vdash \lambda x.e : P \to N} \qquad \frac{\Gamma \vdash v : P}{\Gamma \vdash \mathtt{return}\, v : {\uparrow}P}$$

$$\frac{\Gamma; [M] \vdash s : {\uparrow}Q \qquad \Gamma, z:Q \vdash e : N}{\Gamma \ni (x : {\downarrow}M) \vdash \mathtt{let}\, z{=}x(s);\, e : N} \qquad \frac{\Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} : N}{\Gamma \ni (x:P) \vdash \mathtt{match}\, x\, \{r_i \Rightarrow e_i\}_{i\in I} : N}$$

$\boxed{\Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} : N}$ Left inversion (weak)

$$\frac{}{\Gamma; [0] \vdash \{\} : N} \qquad \frac{\Gamma, x_1 : P_1 \vdash e_1 : N \qquad \Gamma, x_2 : P_2 \vdash e_2 : N}{\Gamma; [P_1 + P_2] \vdash \{\mathtt{inj}_1\, x_1 \Rightarrow e_1 \mid \mathtt{inj}_2\, x_2 \Rightarrow e_2\} : N}$$

$$\frac{\Gamma \vdash e : N}{\Gamma; [1] \vdash \{\langle\rangle \Rightarrow e\} : N} \qquad \frac{\Gamma, x_1 : P_1, x_2 : P_2 \vdash e : N}{\Gamma; [P_1 \times P_2] \vdash \{\langle x_1, x_2\rangle \Rightarrow e\} : N}$$

Figure 4.12: Weakly focalized intuitionistic logic with spines (lists of values not including a continuation) and implicit negative stability and proof terms

$$\text{Expressions} \quad e ::= \cdots \mid \mathsf{let}\, y = \{e\}\,(s);\, e' \mid \underline{\mathtt{match}}\; v\, \{r_i \Rightarrow e_i\}_{i \in I}$$

$$\frac{\Gamma \vdash e : N \qquad \Gamma;[N] \vdash s : \uparrow Q \qquad \Gamma, y : Q \vdash e' : M}{\Gamma \vdash \mathsf{let}\, y = \{e\}\,(s);\, e' : M}$$

$$\frac{\Gamma \vdash v : P \qquad \Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} : M}{\Gamma \vdash \underline{\mathtt{match}}\; v\, \{r_i \Rightarrow e_i\}_{i \in I} : M}$$

$$\frac{\Gamma \vdash v : P \qquad \Gamma, x : P \vdash e : N}{\Gamma \vdash [v/x]e : N} \quad \cdots$$

$$\begin{aligned}
\mathsf{let}\, y = \{e_0\}\,(s);\, e' &\rightsquigarrow \mathsf{let}\, y = \{e\}\,(s);\, e' \qquad \text{if } e_0 \rightsquigarrow^{+} e \text{ where } e \text{ is cut-free} \\
\mathsf{let}\, y = \{\lambda x.\, e\}\,(v,s);\, e' &\rightsquigarrow \mathsf{let}\, y = \{[v/x]e\}\,(s);\, e' \\
\mathsf{let}\, y = \{\mathtt{return}\, v\}\,(\cdot);\, e' &\rightsquigarrow [v/y]e' \\
\mathsf{let}\, y = \{\mathsf{let}\, y' = x(s');\, e\}\,(s);\, e' &\rightsquigarrow \mathsf{let}\, y' = x(s');\, (\mathsf{let}\, y = \{e\}\,(s);\, e') \\
\mathsf{let}\, y = \{\mathtt{match}\, x\, \{r_i \Rightarrow e_i\}_{i \in I}\}\,(s);\, e' &\rightsquigarrow \mathtt{match}\, x\, \{r_i \Rightarrow \mathsf{let}\, y = \{e_i\}\,(s);\, e'\}_{i \in I} \\
\underline{\mathtt{match}}\; \langle\rangle\; \{\langle\rangle \Rightarrow e\} &\rightsquigarrow e \\
\underline{\mathtt{match}}\; \langle v_1, v_2\rangle\; \{\langle x_1, x_2\rangle \Rightarrow e\} &\rightsquigarrow [v_2/x_2]([v_1/x_1]e) \\
\underline{\mathtt{match}}\; (\mathtt{inj}_k\, v)\; \{\mathtt{inj}_1\, x_1 \Rightarrow e_1 \mid \mathtt{inj}_2\, x_2 \Rightarrow e_2\} &\rightsquigarrow [v/x_k]e_k \\
\underline{\mathtt{match}}\; x\, \{r_i \Rightarrow e_i\}_{i \in I} &\rightsquigarrow \mathtt{match}\, x\, \{r_i \Rightarrow e_i\}_{i \in I}
\end{aligned}$$

$$\begin{aligned}
[v/x](\lambda y.\, e) &= \lambda y.\, [v/x]e \\
[v/x](\mathtt{return}\, v') &= \mathtt{return}\, [v/x]v' \\
[v/x](\mathsf{let}\, y = x(s);\, e') &= \mathsf{let}\, y = v([v/x]s);\, [v/x]e' \\
[v/x](\mathsf{let}\, y = z(s);\, e') &= \mathsf{let}\, y = z([v/x]s);\, [v/x]e' \quad \text{if } x \neq z \\
[v/x](\mathtt{match}\, x\, \{r_i \Rightarrow e_i\}_{i \in I}) &= \underline{\mathtt{match}}\; v\, \{r_i \Rightarrow [v/x]e_i\}_{i \in I} \\
[v/x]x &= v \\
[v/x]y &= y \quad (\text{if } x \neq y) \\
[v/x]\langle\rangle &= \langle\rangle \\
[v/x]\langle v_1, v_2\rangle &= \langle [v/x]v_1, [v/x]v_2\rangle \\
&\vdots
\end{aligned}$$

Figure 4.13: Cut elimination in weak FIPL with combined right inversion and negative stability

Variables	x, y, z	
Contexts	Γ	$::= \cdot \mid \Gamma, x : P$
Heads	h	$::= x \mid (v : P)$
Bound expressions	g	$::= h(s) \mid (e : \uparrow P)$
Values	v	$::= x \mid \langle\rangle \mid \langle v, v\rangle \mid \mathtt{inj}_1\, v \mid \mathtt{inj}_2\, v \mid \{e\}$
Spines	s	$::= \cdot \mid v, s$
Expressions	e	$::= \lambda x.\, e \mid \mathtt{return}\, v \mid \mathtt{let}\, x = g;\, e \mid \mathtt{match}\, h\, \{r_i \Rightarrow e_i\}_{i \in I}$
Patterns	r	$::= \mathtt{inj}_1\, x \mid \mathtt{inj}_2\, x \mid \langle\rangle \mid \langle x_1, x_2\rangle$

Figure 4.14: Unrefined programming grammar without inductive types and recursion

$\boxed{\Gamma \vdash h \Rightarrow P}$ Under input Γ, head h synthesizes output type P

$$\frac{(x : P) \in \Gamma}{\Gamma \vdash x \Rightarrow P} \qquad \frac{\Gamma \vdash v \Leftarrow P}{\Gamma \vdash (v : P) \Rightarrow P}$$

$\boxed{\Gamma \vdash g \Rightarrow \uparrow P}$ Under input Γ, input bound expression g synthesizes output type $\uparrow P$

$$\frac{\Gamma \vdash h \Rightarrow \downarrow N \qquad \Gamma ; [N] \vdash s \Rightarrow \uparrow P}{\Gamma \vdash h(s) \Rightarrow \uparrow P} \qquad \frac{\Gamma \vdash e \Leftarrow \uparrow P}{\Gamma \vdash (e : \uparrow P) \Rightarrow \uparrow P}$$

$\boxed{\Gamma \vdash v \Leftarrow P}$ Under input Γ, input value v checks against input type P

$$\frac{(x : P) \in \Gamma}{\Gamma \vdash x \Leftarrow P} \qquad \frac{}{\Gamma \vdash \langle\rangle \Leftarrow 1} \qquad \frac{\Gamma \vdash v_1 \Leftarrow P_1 \qquad \Gamma \vdash v_2 \Leftarrow P_2}{\Gamma \vdash \langle v_1, v_2 \rangle \Leftarrow P_1 \times P_2}$$

$$\frac{\Gamma \vdash v \Leftarrow P_k}{\Gamma \vdash \mathrm{inj}_k\, v \Leftarrow P_1 + P_2} \qquad \frac{\Gamma \vdash e \Leftarrow N}{\Gamma \vdash \{e\} \Leftarrow \downarrow N}$$

$\boxed{\Gamma ; [N] \vdash s \Rightarrow \uparrow P}$ Under input Γ, if a head of type $\downarrow N$ is applied to input spine s, then it will produce a result of type $\uparrow P$ (output)

$$\frac{\Gamma \vdash v \Leftarrow Q \qquad \Gamma ; [N] \vdash s \Rightarrow \uparrow P}{\Gamma ; [Q \to N] \vdash v, s \Rightarrow \uparrow P} \qquad \frac{}{\Gamma ; [\uparrow P] \vdash \cdot \Rightarrow \uparrow P}$$

$\boxed{\Gamma \vdash e \Leftarrow N}$ Under input Γ, input expression e checks against input type N

$$\frac{\Gamma, x : P \vdash e \Leftarrow N}{\Gamma \vdash \lambda x.\, e \Leftarrow P \to N} \qquad \frac{\Gamma \vdash v \Leftarrow P}{\Gamma \vdash \mathtt{return}\, v \Leftarrow \uparrow P} \qquad \frac{\Gamma \vdash g \Rightarrow \uparrow P \qquad \Gamma, x : P \vdash e \Leftarrow N}{\Gamma \vdash \mathtt{let}\, x = g;\, e \Leftarrow N}$$

$$\frac{\Gamma \vdash h \Rightarrow P \qquad \Gamma ; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N}{\Gamma \vdash \mathtt{match}\, h\, \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N}$$

$\boxed{\Gamma ; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N}$ Under input Γ, input patterns r_i match against (input) type P and input branch expressions e_i check against input type N

$$\frac{}{\Gamma ; [0] \vdash \{\} \Leftarrow N} \qquad \frac{\Gamma, x_1 : P_1 \vdash e_1 \Leftarrow N \qquad \Gamma, x_2 : P_2 \vdash e_2 \Leftarrow N}{\Gamma ; [P_1 + P_2] \vdash \{\mathrm{inj}_1\, x_1 \Rightarrow e_1 \mid \mathrm{inj}_2\, x_2 \Rightarrow e_2\} \Leftarrow N}$$

$$\frac{\Gamma \vdash e \Leftarrow N}{\Gamma ; [1] \vdash \{\langle\rangle \Rightarrow e\} \Leftarrow N} \qquad \frac{\Gamma, x_1 : P_1, x_2 : P_2 \vdash e \Leftarrow N}{\Gamma ; [P_1 \times P_2] \vdash \{\langle x_1, x_2 \rangle \Rightarrow e\} \Leftarrow N}$$

Figure 4.15: Unrefined system before adding inductive types and recursion

$$
\begin{aligned}
|\lambda x.\, e| &= \lambda x.\, |e| \\
|\mathtt{return}\, v| &= \mathtt{return}\, |v| \\
|\mathsf{let}\ x = y(s);\ e| &= \mathsf{let}\ x = y(|s|);\ |e| \\
|\mathsf{let}\ x = (\{e'\} : {\downarrow}N)(s);\ e| &= \mathsf{let}\ x = \{|e'|\}\,(|s|);\ |e| \\
|\mathsf{let}\ x = (e' : {\uparrow}P);\ e| &= \mathsf{let}\ x = \{|e'|\}\,(\cdot);\ |e| \\
|\mathtt{match}\ x\ \{r_i \Rightarrow e_i\}_{i\in I}| &= \mathtt{match}\ x\ \{r_i \Rightarrow |e_i|\}_{i\in I} \\
|\mathtt{match}\ (v : P)\ \{r_i \Rightarrow e_i\}_{i\in I}| &= \underline{\mathtt{match}}\ |v|\ |\{r_i \Rightarrow e_i\}_{i\in I}|
\end{aligned}
$$

$$
\begin{aligned}
|x| &= x \\
|\langle\rangle| &= \langle\rangle \\
|\langle v_1, v_2\rangle| &= \langle |v_1|, |v_2|\rangle \\
|\mathsf{inj}_k\, v| &= \mathsf{inj}_k\, |v| \\
|\{e\}| &= \{|e|\}
\end{aligned}
$$

$$
\begin{aligned}
|\cdot| &= \cdot \\
|v, s| &= |v|, |s|
\end{aligned}
$$

$$
|\{r_i \Rightarrow e_i\}_{i\in I}| = \{r_i \Rightarrow |e_i|\}_{i\in I}
$$

Figure 4.16: Erasure of type annotations, a transformation from Fig. 4.15 to Figs. 4.12 and 4.13

Chapter 5

Declarative Unrefined System

Let us finish creating our (unrefined) statically typed functional programming language by adding algebraic data types (the point of this thesis is to modularly refine algebraic data types) and recursion (without which, ADTs would be pointless). As we are designing a programming language, we have proof terms (program terms), so we can treat ADTs syntactically as *isorecursive* types[1] where the isomorphism between an ADT and its unrolling is witnessed explicitly by proof terms. We specify an algebraic datatype μF as the initial fixed point μF of a polynomial functor (intuitively, a tree-like type transformer) whose constants are value types. A syntactic value $\texttt{into}(v)$ has type μF if v has the type corresponding to the unrolling $F(\mu F)$ of μF. Dually, a pattern $\texttt{into}(x)$ of type μF deconstructs into an assumption that x has type corresponding to $F(\mu F)$.

5.1 Recursive Expressions and Algebraic Datatypes Added

The grammar is Fig. 5.1. Relative to Fig. 4.14, the extension goes like so: $v ::= \cdots \mid \texttt{into}(v)$ and $r ::= \cdots \mid \texttt{into}(x)$ and $e ::= \cdots \mid \texttt{rec}\, x.\, e$ and $P ::= \cdots \mid \mu F$ and the grammar(s) related

[1] Later, we treat ADTs *semantically* as *equirecursive*: the denotation ("meaning") of an ADT *equals* the denotation of its unrolling.

to functors F.

Program variables	x, y, z		
Expressions	e	$::=$	$\mathtt{return}\, v \mid \mathtt{let}\, x = g;\, e \mid \mathtt{match}\, h\, \{r_i \Rightarrow e_i\}_{i \in I} \mid \lambda x.\, e$ $\mid \mathtt{rec}\, x.\, e$
Values	v	$::=$	$x \mid \langle\rangle \mid \langle v, v\rangle \mid \mathtt{inj}_1\, v \mid \mathtt{inj}_2\, v \mid \mathtt{into}(v) \mid \{e\}$
Heads	h	$::=$	$x \mid (v : P)$
Bound expressions	g	$::=$	$h(s) \mid (e : \uparrow P)$
Spines	s	$::=$	$\cdot \mid v, s$
Patterns	r	$::=$	$\mathtt{into}(x) \mid \langle\rangle \mid \langle x, y\rangle \mid \mathtt{inj}_1\, x \mid \mathtt{inj}_2\, x$
Unrefined positive types	P, Q, R	$::=$	$1 \mid P \times Q \mid 0 \mid P + Q \mid \downarrow N \mid \mu F$
Unrefined negative types	N, M, L	$::=$	$P \to N \mid \uparrow P$
Types	A, B, C	$::=$	$P \mid N$
Unrefined functors	F, G, H	$::=$	$\hat{P} \mid F \oplus F$
Constant (product) functors	$\hat{P}$	$::=$	$\hat{I} \mid \underline{P} \otimes \hat{P}$
Identity (product) functors	$\hat{I}$	$::=$	$I \mid \mathrm{Id} \otimes \hat{I}$
Base functors	$\hat{B}$	$::=$	$\underline{P} \mid \mathrm{Id}$
	$\mathscr{F}$	$::=$	$F \mid \hat{B}$

Figure 5.1: Unrefined syntax

We specify algebraic data types (and measures on them) based on their standard categorical semantics [Goguen et al., 1977]. In the introduction (Chapter 1), to refine the type of A-lists by their length, we defined a recursive function len over the inductive structure of lists. Semantically, we characterize this structural recursion by algebraic folds over polynomial endofunctors; we design our system in line with this semantics. While this presentation may appear overly abstract for the user, it should be possible to allow the user to use the same or similar syntax as programs to express measures if they annotate them as measures in the style of Liquid Haskell.

We express inductive type structure without reference to constructor names by syntactic functors resembling the polynomial functors. For example (modulo the difference for

simplifying unrolling), we can specify the signature of the inductive type of lists of terms of type A syntactically as a functor $\underline{1} \oplus (\underline{A} \otimes \mathrm{Id})$, where $\underline{C}$ denotes the constant (set) functor (sending any set to the set denoted by type C), Id denotes the identity functor (sending any set to itself), the denotation of $F_1 \otimes F_2$ sends a set X to the product $(\llbracket F_1 \rrbracket X) \times (\llbracket F_2 \rrbracket X)$, and the denotation of $F_1 \oplus F_2$ sends a set X to the disjoint union $(\llbracket F_1 \rrbracket X) \uplus (\llbracket F_2 \rrbracket X)$. The idea is that each component of the sum functor $\oplus$ represents a data constructor, so that (for example) $\underline{1}$ represents the nullary constructor $[]$, and $\underline{A}$ represents the head element of a cons cell which is attached (via $\otimes$) to the recursive tail list represented by Id.

A functor F is a sum ($\oplus$) of products ($\hat{P}$), which multiply ($\otimes$) base functors ($\hat{B}$), which consist of constant functors ($\underline{P}$) at some positive type P and identity functors that represent recursive positions (Id). The rightmost factor in a product $\hat{P}$ is the (product) unit functor I, the leftmost factors are constant functors, and the factors in between are identity functors. By convention, $\otimes$ has higher precedence than $\oplus$. For convenience in specifying functor well-formedness and denotation $\mathscr{F}$ is a functor F or a base functor $\hat{B}$.

A direct grammar F for sums of products (of constant and identity functors) consists of $F ::= \hat{P} \mid F \oplus F$ and $\hat{P} ::= \hat{B} \mid \hat{B} \otimes \hat{P}$ and $\hat{B} ::= \underline{P} \mid \mathrm{Id}$. The grammar $F ::= F \oplus F \mid F \otimes F \mid \hat{B}$ is semantically equivalent to sums of products, but syntactically inconvenient, because it allows writing products of sums. We do not use either of these grammars, but rather Fig. 5.1, because it simplifies refined inductive type unrolling (Sec. 6.4). (In any case, a surface language where data types have named constructors would have to be elaborated to use one of these grammars.) These grammars have naturally isomorphic interpretations. For example, in our functor grammar (Fig. 5.1), we instead write $\mathsf{NatF} = I \oplus (\mathrm{Id} \otimes I)$ (note I is semantically equivalent to $\underline{1}$): notice that for any set X, we have $\llbracket I \oplus (\mathrm{Id} \otimes I) \rrbracket X = 1 \uplus (X \times 1) \cong 1 \uplus X = \llbracket \underline{1} \oplus \mathrm{Id} \rrbracket X$.

$\boxed{\vdash G[\mu F] \doteq P}$ Functor G applied to type μF corresponds to the (output) type P (inputs: G and F)

$$\frac{\vdash G[\mu F] \doteq P \qquad \vdash H[\mu F] \doteq Q}{\vdash (G \oplus H)[\mu F] \doteq P + Q}\ \text{UnrefUnroll}\oplus$$

$$\frac{\vdash \hat{P}[\mu F] \doteq P}{\vdash (\underline{Q} \otimes \hat{P})[\mu F] \doteq Q \times P}\ \text{UnrefUnrollConst} \qquad \frac{\vdash \hat{I}[\mu F] \doteq P}{\vdash (\mathrm{Id} \otimes \hat{I})[\mu F] \doteq \mu F \times P}\ \text{UnrefUnrollId}$$

$$\frac{}{\vdash I[\mu F] \doteq 1}\ \text{UnrefUnroll}I$$

Figure 5.2: Unrefined unrolling

The unrefined unrolling judgment $\vdash G[\mu F] \stackrel{\circ}{=} P$ is defined in Fig. 5.2. The functor G is called the *principal* functor of the unrolling judgment and we speak of "G-unrolling" (...output P, say). Rule UnrefUnroll$\oplus$ says $G \oplus H$ applied to μF outputs the sum type whose left summand is the output of G applied to μF and whose right summand is the output of H applied to μF. Rule UnrefUnrollConst says $\underline{P} \otimes \hat{P}$ applied to μF outputs the product type whose left factor is P and whose right factor is the result of applying functor $\hat{P}$ to type μF. Rule UnrefUnrollId says the functor $\mathrm{Id} \otimes \hat{I}$ applied to μF outputs the product type whose left factor is μF and whose right factor is the result of applying functor $\hat{I}$ to type μF. Rule UnrefUnroll$I$ says the unit functor I applied to μF outputs the unit type 1.

The typing judgments of the unrefined system are given in Fig. 5.3 and Fig. 5.4. Unrefined typing judgments presuppose the *well-formedness* Γ ctx of contexts Γ in that each variable x can occur at most once in Γ (and the types occurring in it are generated by the grammar for unrefined types, Fig. 5.1). Specifically, the unrefined context well-formedness judgment Γ ctx has two rules:[2]

$$\frac{}{\cdot \text{ ctx}} \qquad\qquad \frac{\Gamma \text{ ctx} \qquad x \notin dom(\Gamma)}{(\Gamma, x : P) \text{ ctx}}$$

[2]We define the *domain* $dom(\mathscr{O})$ of any context $\mathscr{O}$ of variable bindings to be the set of variables bound to a sort or type or such. In this case, $dom(\cdot) = \emptyset$ and $dom(\Gamma, x : P) = \{x\} \cup dom(\Gamma)$.

$\boxed{\Gamma \vdash h \Rightarrow P}$ Under input Γ, head h synthesizes (output) type P

$$\frac{(x : P) \in \Gamma}{\Gamma \vdash x \Rightarrow P}\ \text{Unref}{\Rightarrow}\text{Var} \qquad \frac{\Gamma \vdash v \Leftarrow P}{\Gamma \vdash (v : P) \Rightarrow P}\ \text{Unref}{\Rightarrow}\text{ValAnnot}$$

$\boxed{\Gamma \vdash g \Rightarrow \uparrow P}$ Under input Γ, input bound expression g synthesizes type $\uparrow P$ (output)

$$\frac{\Gamma \vdash h \Rightarrow \downarrow N \qquad \Gamma; [N] \vdash s \Rightarrow \uparrow P}{\Gamma \vdash h(s) \Rightarrow \uparrow P}\ \text{Unref}{\Rightarrow}\text{App} \qquad \frac{\Gamma \vdash e \Leftarrow \uparrow P}{\Gamma \vdash (e : \uparrow P) \Rightarrow \uparrow P}\ \text{Unref}{\Rightarrow}\text{ExpAnnot}$$

$\boxed{\Gamma \vdash v \Leftarrow P}$ Under input Γ, input value v checks against input type P

$$\frac{(x : P) \in \Gamma}{\Gamma \vdash x \Leftarrow P}\ \text{Unref}{\Leftarrow}\text{Var} \qquad \frac{}{\Gamma \vdash \langle\rangle \Leftarrow 1}\ \text{Unref}{\Leftarrow}1$$

$$\frac{\Gamma \vdash v_1 \Leftarrow P_1 \qquad \Gamma \vdash v_2 \Leftarrow P_2}{\Gamma \vdash \langle v_1, v_2 \rangle \Leftarrow P_1 \times P_2}\ \text{Unref}{\Leftarrow}\times \qquad \frac{\Gamma \vdash v \Leftarrow P_k}{\Gamma \vdash \text{inj}_k\, v \Leftarrow P_1 + P_2}\ \text{Unref}{\Leftarrow}+_k$$

$$\frac{\vdash F[\mu F] \doteq P \qquad \Gamma \vdash v \Leftarrow P}{\Gamma \vdash \texttt{into}(v) \Leftarrow \mu F}\ \text{Unref}{\Leftarrow}\mu \qquad \frac{\Gamma \vdash e \Leftarrow N}{\Gamma \vdash \{e\} \Leftarrow \downarrow N}\ \text{Unref}{\Leftarrow}\downarrow$$

$\boxed{\Gamma \vdash e \Leftarrow N}$ Under input Γ, input expression e checks against input type N

$$\frac{\Gamma \vdash v \Leftarrow P}{\Gamma \vdash \texttt{return}\, v \Leftarrow \uparrow P}\ \text{Unref}{\Leftarrow}\uparrow \qquad \frac{\Gamma \vdash g \Rightarrow \uparrow P \qquad \Gamma, x : P \vdash e \Leftarrow N}{\Gamma \vdash \texttt{let}\ x = g;\ e \Leftarrow N}\ \text{Unref}{\Leftarrow}\texttt{let}$$

$$\frac{\Gamma \vdash h \Rightarrow P \qquad \Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N}{\Gamma \vdash \texttt{match}\ h\ \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N}\ \text{Unref}{\Leftarrow}\texttt{match}$$

$$\frac{\Gamma, x : P \vdash e \Leftarrow N}{\Gamma \vdash \lambda x.\, e \Leftarrow P \to N}\ \text{Unref}{\Leftarrow}\lambda \qquad \frac{\Gamma, x : \downarrow N \vdash e \Leftarrow N}{\Gamma \vdash \texttt{rec}\, x.\, e \Leftarrow N}\ \text{Unref}{\Leftarrow}\texttt{rec}$$

Figure 5.3: Unrefined declarative typing

$\boxed{\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N}$ Under Γ, patterns r_i match against type P and branch expressions e_i check against type N (all inputs)

$$\frac{\Gamma \vdash e \Leftarrow N}{\Gamma;[1] \vdash \{\langle\rangle \Rightarrow e\} \Leftarrow N}\ \text{UnrefMatch1} \qquad \frac{\Gamma, x_1 : P_1, x_2 : P_2 \vdash e \Leftarrow N}{\Gamma;[P_1 \times P_2] \vdash \{\langle x_1, x_2\rangle \Rightarrow e\} \Leftarrow N}\ \text{UnrefMatch}\times$$

$$\frac{\Gamma, x_1 : P_1 \vdash e_1 \Leftarrow N \qquad \Gamma, x_2 : P_2 \vdash e_2 \Leftarrow N}{\Gamma;[P_1 + P_2] \vdash \{\mathtt{inj}_1\, x_1 \Rightarrow e_1 \mid \mathtt{inj}_2\, x_2 \Rightarrow e_2\} \Leftarrow N}\ \text{UnrefMatch}+$$

$$\frac{}{\Gamma;[0] \vdash \{\} \Leftarrow N}\ \text{UnrefMatch0} \qquad \frac{\vdash F[\mu F] \doteq P \qquad \Gamma, x : P \vdash e \Leftarrow N}{\Gamma;[\mu F] \vdash \{\mathtt{into}(x) \Rightarrow e\} \Leftarrow N}\ \text{UnrefMatch}\mu$$

$\boxed{\Gamma;[N] \vdash s \Rightarrow \uparrow P}$ Under input Γ, if input spine s is applied to a head of type $\downarrow N$ (input: N), then it will produce a result of type $\uparrow P$ (output)

$$\frac{\Gamma \vdash v \Leftarrow Q \qquad \Gamma;[N] \vdash s \Rightarrow \uparrow P}{\Gamma;[Q \to N] \vdash v, s \Rightarrow \uparrow P}\ \text{UnrefSpineApp} \qquad \frac{}{\Gamma;[\uparrow P] \vdash \cdot \Rightarrow \uparrow P}\ \text{UnrefSpineNil}$$

Figure 5.4: Unrefined matching and spines

Relative to Fig. 4.15, the new rules in Fig. A.35 and Fig. A.36 are Unref$\Leftarrow\mu$, UnrefMatchμ, and Unref$\Leftarrow$rec.

5.2 Operational and Denotational Semantics and Adequacy

We define the operational semantics by extending Fig. 4.12 and Fig. 4.13 and modifying the first rule. Relative to Fig. 4.12: $v ::= \cdots \mid \mathtt{into}(v)$ and $e ::= \cdots \mid \mathtt{rec}\, x.\, e$ and $r ::= \cdots \mid \mathtt{into}(x)$ and rules similar to Unref$\Leftarrow$rec, Unref$\Leftarrow\mu$, and UnrefMatchμ. Relative to Fig. 4.13 we extend $\rightsquigarrow$ by adding the two cases

$$\mathtt{rec}\, x.\, e \rightsquigarrow [\{\mathtt{rec}\, x.\, e\}/x]e$$

$$\underline{\mathtt{match}}\ \mathtt{into}(v)\ \{\mathtt{into}(x) \Rightarrow e\} \rightsquigarrow [v/x]e$$

and modifying the first case not to require reduction to a cut-free proof (reduction to a cut-free proof is for establishing cut elimination but cut elimination no longer holds with recursion):

$$\mathtt{let}\ y = \{e_0\}\,(s);\, e' \rightsquigarrow \mathtt{let}\ y = \{e\}\,(s);\, e' \qquad \text{if } e_0 \rightsquigarrow e$$

Strictly speaking, algebraic datatypes are not needed to express a diverging term.

Definition 5.1 (diverge and divfun)**.**

We define divfun to be the expression $\mathtt{rec}\, f.\, \lambda x.\mathtt{let}\ y = f(x);\ \mathtt{return}\, y$.

Given any unrefined type N, we define $\mathtt{diverge}_N$ to be the expression

$$\mathtt{let}\ f = (\mathtt{return}\,\{\mathtt{divfun}\} : \uparrow\downarrow(1 \to \uparrow\downarrow N));\ \mathtt{let}\ z = f\,\langle\rangle;\ \mathtt{return}\, z$$

We define $\mathtt{diverge}$ to be $\mathtt{diverge}_{\uparrow 0}$.

We may read the definition of $\mathtt{divfun}$ from left to right as follows: $\mathtt{divfun}$ is the recursive expression named f that takes an argument x and applies f to x binding the result to y and returning y. The expression $\mathtt{diverge}$ applies $\mathtt{divfun}$ to the unit value $\langle\rangle$. The expression $\mathtt{divfun}$ checks against many types in the unrefined system, like $1 \to \uparrow\downarrow N$ where N is an arbitrary (unrefined) negative type. In the following derivation, which we name $\mathscr{D}$, let Γ be $f : \downarrow(1 \to \uparrow\downarrow N), x : 1$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash f \Rightarrow \downarrow(1 \to \uparrow\downarrow N) \qquad \dfrac{\Gamma \vdash x \Leftarrow 1 \qquad \Gamma;[\uparrow\downarrow N] \vdash \cdot \Rightarrow \uparrow\downarrow N}{\Gamma;[1 \to \uparrow\downarrow N] \vdash x \Rightarrow \uparrow\downarrow N}}{\Gamma \vdash f(x) \Rightarrow \uparrow\downarrow N} \qquad \dfrac{\Gamma, y : \downarrow N \vdash y \Leftarrow \downarrow N}{\Gamma, y : \downarrow N \vdash \mathtt{return}\, y \Leftarrow \uparrow\downarrow N}}{f : \downarrow(1 \to \uparrow\downarrow N), x : 1 \vdash \mathtt{let}\ y = f(x); \mathtt{return}\, y \Leftarrow \uparrow\downarrow N}}{f : \downarrow(1 \to \uparrow\downarrow N) \vdash \lambda x.\mathtt{let}\ y = f(x); \mathtt{return}\, y \Leftarrow 1 \to \uparrow\downarrow N}}{\vdash \underbrace{\mathtt{rec}\, f.\, \lambda x.\mathtt{let}\ y = f(x); \mathtt{return}\, y}_{\mathtt{divfun}} \Leftarrow 1 \to \uparrow\downarrow N}$$

Let $\mathscr{D}'$ be the derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathscr{D}}{\vdash \mathtt{divfun} \Leftarrow 1 \to \uparrow\downarrow N}}{\vdash \{\mathtt{divfun}\} \Leftarrow \downarrow(1 \to \uparrow\downarrow N)}}{\vdash \mathtt{return}\,\{\mathtt{divfun}\} \Leftarrow \uparrow\downarrow(1 \to \uparrow\downarrow N)}}{\vdash (\mathtt{return}\,\{\mathtt{divfun}\} : \uparrow\downarrow(1 \to \uparrow\downarrow N)) \Rightarrow \uparrow\downarrow(1 \to \uparrow\downarrow N)}$$

Let Γ' be $f : \downarrow(1 \to \uparrow\downarrow N)$. Let $\mathscr{E}$ be the derivation

$$\dfrac{\dfrac{\Gamma' \vdash f \Rightarrow \downarrow(1 \to \uparrow\downarrow N) \qquad \dfrac{\Gamma' \vdash \langle\rangle \Leftarrow 1 \qquad \Gamma'; [\uparrow\downarrow N] \vdash \cdot \Rightarrow \uparrow\downarrow N}{\Gamma'; [1 \to \uparrow\downarrow N] \vdash \langle\rangle \Rightarrow \uparrow\downarrow N}}{\Gamma' \vdash f\,\langle\rangle \Rightarrow \uparrow\downarrow N} \qquad \dfrac{\Gamma', z : \downarrow N \vdash z \Leftarrow \downarrow N}{\Gamma', z : \downarrow N \vdash \mathtt{return}\, z \Leftarrow \uparrow\downarrow N}}{\Gamma' \vdash \mathtt{let}\, z = f\,\langle\rangle; \mathtt{return}\, z \Leftarrow \uparrow\downarrow N}$$

We apply one more rule to obtain $\uparrow\downarrow N$.

$$\dfrac{\vdash (\mathtt{return}\,\{\mathtt{divfun}\} : \uparrow\downarrow(1 \to \uparrow\downarrow N)) \Rightarrow \uparrow\downarrow(1 \to \uparrow\downarrow N) \qquad \Gamma' \vdash \mathtt{let}\, z = f\,\langle\rangle; \mathtt{return}\, z \Leftarrow \uparrow\downarrow N}{\vdash \underbrace{\mathtt{let}\, f = (\mathtt{return}\,\{\mathtt{divfun}\} : \uparrow\downarrow(1 \to \uparrow\downarrow N)); \mathtt{let}\, z = f\,\langle\rangle; \mathtt{return}\, z}_{\mathtt{diverge}_N} \Leftarrow \uparrow\downarrow N}$$

In particular, $\vdash \mathtt{diverge}_{\uparrow 0} \Leftarrow \uparrow\downarrow\uparrow 0$. That is, the corresponding logic of the system with recursion added is inconsistent. Further, as the name suggests, $\mathtt{diverge}_N$ diverges.

$$
\begin{aligned}
|\mathsf{diverge}_N| &= |\mathsf{let}\ f = (\mathsf{return}\,\{\mathsf{divfun}\} : \uparrow\downarrow(1 \to \uparrow\downarrow N));\ \mathsf{let}\ z = f\,\langle\rangle;\ \mathsf{return}\,z| \\
&= \mathsf{let}\ f = \{\mathsf{return}\,\{\mathsf{divfun}\}\}\,(\cdot);\ \mathsf{let}\ z = f\,\langle\rangle;\ \mathsf{return}\,z \\
&\rightsquigarrow [\{\mathsf{divfun}\}/f](\mathsf{let}\ z = f\,\langle\rangle;\ \mathsf{return}\,z) \\
&= \underbrace{\mathsf{let}\ z = \{\mathsf{divfun}\}\,\langle\rangle;\ \mathsf{return}\,z}_{e} \\
&= \mathsf{let}\ z = \{\mathsf{rec}\ f.\ \lambda x.\mathsf{let}\ y = f(x);\ \mathsf{return}\,y\}\,\langle\rangle;\ \mathsf{return}\,z \\
&\rightsquigarrow \mathsf{let}\ z = \{\lambda x.\mathsf{let}\ y = \{\mathsf{divfun}\}\,x;\ \mathsf{return}\,y\}\,\langle\rangle;\ \mathsf{return}\,z \\
&\rightsquigarrow \mathsf{let}\ z = \left\{\underbrace{\mathsf{let}\ y = \{\mathsf{divfun}\}\,\langle\rangle;\ \mathsf{return}\,y}_{=_\alpha e}\right\};\ \mathsf{return}\,z \\
&\rightsquigarrow \cdots
\end{aligned}
$$

That is, cut elimination no longer holds, which is what we should expect because otherwise the system would be consistent. This is a good thing, though, for our purposes, which is to design a refinement type system for a programming language and then prove it is logically consistent (and other, semantic and algorithmic properties) in a largely straightforward manner. Turing completeness means the language can express any (conventional) computation, which is a powerful place to start. One power of our refinement type system will be to verify that programs terminate.

The unrefined type assignment system is given in Fig. 5.5. To simplify syntax, we could drop the underline of $\mathsf{let}\ -=-;\ -$ since $\mathsf{let}\ z = x(s);\ e' \neq \mathsf{let}\ z = \{e\}\,(s);\ e'$. This simplified

$\boxed{\Gamma \vdash v : P}$ Under Γ value v has type P

$$\frac{\Gamma \ni (x : P)}{\Gamma \vdash x : P} \qquad \frac{}{\Gamma \vdash \langle\rangle : 1} \qquad \frac{\Gamma \vdash v_1 : P_1 \qquad \Gamma \vdash v_2 : P_2}{\Gamma \vdash \langle v_1, v_2 \rangle : P_1 \times P_2}$$

$$\frac{k \in \{1,2\} \qquad \Gamma \vdash v : P_k}{\Gamma \vdash \mathsf{inj}_k\, v : P_1 + P_2} \qquad \frac{\vdash F[\mu F] \triangleq P \qquad \Gamma \vdash v : P}{\Gamma \vdash \mathtt{into}(v) : \mu F} \qquad \frac{\Gamma \vdash e : N}{\Gamma \vdash \{e\} : \downarrow N}$$

$\boxed{\Gamma ; [N] \vdash s : \uparrow P}$ Under Γ spine s has type N returning $\uparrow P$

$$\frac{\Gamma \vdash v : Q \qquad \Gamma ; [N] \vdash s : \uparrow P}{\Gamma ; [Q \to N] \vdash v, s : \uparrow P} \qquad \frac{}{\Gamma ; [\uparrow P] \vdash \cdot : \uparrow P}$$

$\boxed{\Gamma \vdash e : N}$ Under Γ expression e has type N
(the types highlighted with a green box are the *main cut* types)

$$\frac{\Gamma, x : \downarrow N \vdash e : N}{\Gamma \vdash \mathtt{rec}\, x.\, e : N} \qquad \frac{\Gamma, x : P \vdash e : N}{\Gamma \vdash \lambda x.e : P \to N} \qquad \frac{\Gamma \vdash v : P}{\Gamma \vdash \mathtt{return}\, v : \uparrow P}$$

$$\frac{\Gamma ; [M] \vdash s : \uparrow Q \qquad \Gamma, z : Q \vdash e : N}{\Gamma \ni (x : \downarrow M) \vdash \mathsf{let}\ z = x(s);\ e : N} \qquad \frac{\Gamma ; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} : N}{\Gamma \ni (x : P) \vdash \mathtt{match}\, x\, \{r_i \Rightarrow e_i\}_{i \in I} : N}$$

$$\frac{\Gamma \vdash e : \boxed{M} \qquad \Gamma ; [M] \vdash s : \uparrow Q \qquad \Gamma, y : Q \vdash e' : N}{\Gamma \vdash \mathsf{let}\ y = \{e\}\,(s);\ e' : N}$$

$$\frac{\Gamma \vdash v : \boxed{P} \qquad \Gamma ; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} : N}{\Gamma \vdash \underline{\mathtt{match}}\, v\, \{r_i \Rightarrow e_i\}_{i \in I} : N}$$

$\boxed{\Gamma ; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} : N}$ Under Γ the pattern r_k and expression e_k of each clause $r_k \Rightarrow e_k$ has type P and N respectively

$$\frac{}{\Gamma ; [0] \vdash \{\} : N} \qquad \frac{\Gamma, x_1 : P_1 \vdash e_1 : N \qquad \Gamma, x_2 : P_2 \vdash e_2 : N}{\Gamma ; [P_1 + P_2] \vdash \{\mathsf{inj}_1\, x_1 \Rightarrow e_1 \mid \mathsf{inj}_2\, x_2 \Rightarrow e_2\} : N}$$

$$\frac{\Gamma \vdash e : N}{\Gamma ; [1] \vdash \{\langle\rangle \Rightarrow e\} : N} \qquad \frac{\Gamma, x_1 : P_1, x_2 : P_2 \vdash e : N}{\Gamma ; [P_1 \times P_2] \vdash \{\langle x_1, x_2 \rangle \Rightarrow e\} : N}$$

$$\frac{\vdash F[\mu F] \triangleq P \qquad \Gamma, x : P \vdash e : N}{\Gamma ; [\mu F] \vdash \{\mathtt{into}(x) \Rightarrow e\} : N}$$

Figure 5.5: Unrefined type assignment system

syntax simplifies the definition of forcing a thunk, which is often primitive in a CBPV.

$$\frac{\Gamma \vdash v : \downarrow N}{\Gamma \vdash \mathtt{force}\, v : N}$$

Given $\Gamma \vdash v : \downarrow N$ we can define $\mathtt{force}\, v$ such that $\Gamma \vdash \mathtt{force}\, v : N$ as follows.

$$\mathtt{force}\, v = \lambda \overleftarrow{x}.\mathtt{let}\ y = v(\overleftarrow{x});\ \mathtt{return}\, y$$

where y and $\overleftarrow{x} = x_1, \ldots, x_n$ are fresh (that is, not in $dom(\Gamma)$) and n is the arity of N (that is, the natural number of arguments of $N = P_1 \to \cdots \to P_n \to \uparrow R$). In the bidirectional system, we need to annotate the application of v in $\mathtt{force}\, v$ if v is not a variable.

$$\mathtt{force}\, v = \begin{cases} \lambda \overleftarrow{x}.\mathtt{let}\ y = v(\overleftarrow{x});\ \mathtt{return}\, y & \text{if } v \text{ is a variable} \\ \lambda \overleftarrow{x}.\mathtt{let}\ y = (v : \downarrow N)(\overleftarrow{x});\ \mathtt{return}\, y & \text{else} \end{cases}$$

The denotational semantics of the system defined by Fig. 4.12 + Fig. 4.13 + recursion ($\mathtt{rec}\ x.\ e$) + ADTs ($\mathtt{into}(v)$)—that is, the unrefined (non-bidirectional) type assignment system, given in Fig. 5.5—is similar to the denotational semantics of the unrefined system, discussed next chapter. The semantics of types is the same as that of the unrefined (bidirectional) system where positive types denote complete partial orders (cpos) and negative types denote pointed cpos (cppos) and functors denote endofunctors on the category of cpos and continuous functions: Fig. 5.9 and Fig. 5.10. The denotational semantics of type assignment derivations in the unrefined type assignment system is given in Fig. 5.6 (π_k are set projections $X_1 \times X_2 \to X_k$ defined by $\pi_k(x_1, x_2) = x_k$ and inj_k are set injections $X_k \to X_1 \uplus X_2$ defined by $inj_k(x) = (k, x)$). The statement and proof of denotational soundness of the

unrefined type assignment system is similar to that of the unrefined (bidirectional) typing system: Lemma 5.3 (Unrefined Typing Soundness).

The results in the remainder of this section are basically standard and unsurprising but presented in a slightly different way than usual because we are using a certain focused sequent calculus. We do not present most of the semantic definitions and metatheory of the type assignment system which compares to those of its bidirectionalization, presented in the next section.

It is straightforward to calculate $[\![\mathtt{diverge}_N]\!]_\cdot = \bot_{[\![\uparrow\downarrow N]\!]}$ by unfolding definitions. But does every (closed[3]) diverging expression denote the bottom (that is, least informative because never terminating to a result) element of its domain: is our denotational semantics operationally or computationally adequate? Let's discuss some basic operational semantics results before addressing this question.

We collect the operational semantics in Fig. 5.7 for convenience. We can generalize substitution to *parallel* substitution $[\sigma]-$ where σ is a list of substitutions v/x. The key clauses are

$$[\sigma](\mathtt{let}\ y{=}x(s);\ e') = \mathtt{let}\ y{=}\sigma(x)([\sigma]s);\ [\sigma]e'$$

$$[\sigma](\mathtt{match}\ x\ \{r_i \Rightarrow e_i\}_{i\in I}) = \underline{\mathtt{match}}\ \sigma(x)\ \{r_i \Rightarrow [\sigma]e_i\}_{i\in I}$$

where $\sigma(x)$ simply looks up the value v substituted for x: that is, $\sigma(x) = v$ if $v/x \in \sigma$. (In the product matching reduction, for example, the sequential substitution $[v_1/x_1][v_2/x_2]e$ is equivalent to the parallel substitution $[v_1/x_1, v_2/x_2]e$.) A σ is well-typed $\Gamma_0 \vdash \sigma : \Gamma$ in the unrefined assignment system if $\Gamma_0 \vdash \sigma(x) : \Gamma(x)$ for all $x \in dom(\Gamma)$ (where $\Gamma(x)$ looks up the type bound to x in Γ).

[3] A term is *closed* if it has no free variables.

$$
\begin{aligned}
[\![\Gamma \vdash x : P]\!]_\delta &= \delta(x) \\
[\![\Gamma \vdash \langle\rangle : 1]\!]_\delta &= \bullet \\
[\![\Gamma \vdash \langle v_1, v_2\rangle : P_1 \times P_2]\!]_\delta &= ([\![\Gamma \vdash v_1 : P_1]\!]_\delta, [\![\Gamma \vdash v_2 : P_2]\!]_\delta) \\
[\![\mathtt{inj}_k\, v]\!]_\delta &= inj_k\, [\![v]\!]_\delta \\
[\![\{e\}]\!]_\delta &= [\![e]\!]_\delta \\
[\![\lambda x.\, e]\!]_\delta &= d \mapsto [\![e]\!]_{\delta, d/x} \\
[\![\mathtt{return}\, v]\!]_\delta &= inj_1\, [\![v]\!]_\delta \\
[\![\Gamma \vdash \mathtt{let}\ z = x(s);\ e : N]\!]_\delta &= \begin{cases} [\![e]\!]_{\delta, d/z} & \text{if } [\![s]\!]_\delta\, \delta(x) = inj_1\, d \\ \bot_{[\![N]\!]} & \text{else} \end{cases} \\
[\![\mathtt{match}\ x\ \{r_i \Rightarrow e_i\}_{i \in I}]\!]_\delta &= [\![\{r_i \Rightarrow e_i\}_{i \in I}]\!]_\delta\, \delta(x) \\
[\![\Gamma \vdash \mathtt{rec}\, x.\, e : N]\!]_\delta &= \bigsqcup_{k \in \mathbb{N}} [\![\Gamma, x : \downarrow N \vdash e : N]\!]^k_{\delta, -/x} \bot_{[\![N]\!]} \\
[\![\{\}]\!]_\delta &= \text{empty function} \\
[\![\{\langle\rangle \Rightarrow e\}]\!]_\delta &= d \mapsto [\![e]\!]_\delta \\
[\![\{\langle x_1, x_2\rangle \Rightarrow e\}]\!]_\delta &= d \mapsto [\![e]\!]_{\delta, \pi_1(d)/x_1, \pi_2(d)/x_2} \\
[\![\{\mathtt{inj}_1\, x_1 \Rightarrow e_1 \mid \mathtt{inj}_2\, x_2 \Rightarrow e_2\}]\!]_\delta &= d \mapsto \begin{cases} [\![e_1]\!]_{\delta, d_1/x_1} & \text{if } d = inj_1\, d_1 \\ [\![e_2]\!]_{\delta, d_2/x_2} & \text{if } d = inj_2\, d_2 \end{cases} \\
[\![\Gamma \vdash \mathtt{let}\ y = \{e\}\,(s);\ e' : N]\!]_\delta &= \begin{cases} [\![e']\!]_{\delta, d/y} & \text{if } [\![s]\!]_\delta\, [\![e]\!]_\delta = inj_1\, d \\ \bot_{[\![N]\!]} & \text{else} \end{cases} \\
[\![\underline{\mathtt{match}}\ v\ \{r_i \Rightarrow e_i\}_{i \in I}]\!]_\delta &= [\![\{r_i \Rightarrow e_i\}_{i \in I}]\!]_\delta\, [\![v]\!]_\delta
\end{aligned}
$$

Figure 5.6: Denotational semantics of unrefined type assignment derivations where $\delta \in [\![\Gamma]\!]$ is a list of semantic substitutions d/x where $x : P$ and $d \in [\![P]\!]$

The standard origin story surrounding progress and preservation is that structural operational semantic type soundness begins with Wright and Felleisen [1994] and was simplified into "progress and preservation" by Harper [2016].

Theorem 5.1 (Progress and Preservation)**.**

(1) (Progress 1) *If* $\cdot \vdash e : \uparrow P$ *then* $e \rightsquigarrow e'$ *for some* e' *or* $e = \mathtt{return}\, v$ *for some* v*.*

$$
\begin{aligned}
\mathsf{let}\ y = \{e_0\}\,(s);\ e' &\rightsquigarrow \mathsf{let}\ y = \{e\}\,(s);\ e' \qquad \text{if } e_0 \rightsquigarrow e \\
\mathsf{let}\ y = \{\lambda x.\, e\}\,(v,s);\ e' &\rightsquigarrow \mathsf{let}\ y = \{[v/x]e\}\,(s);\ e' \\
\mathsf{let}\ y = \{\mathsf{return}\, v\}\,(\cdot);\ e' &\rightsquigarrow [v/y]e'
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{let}\ y = \left\{\mathsf{let}\ y' = x(s');\ e\right\}(s);\ e' &\rightsquigarrow \mathsf{let}\ y' = x(s');\ (\mathsf{let}\ y = \{e\}\,(s);\ e') \\
\mathsf{let}\ y = \{\mathsf{match}\ x\ \{r_i \Rightarrow e_i\}_{i\in I}\}\,(s);\ e' &\rightsquigarrow \mathsf{match}\ x\ \left\{r_i \Rightarrow \mathsf{let}\ y = \{e_i\}\,(s);\ e'\right\}_{i\in I}
\end{aligned}
$$

$$
\mathsf{rec}\, x.\ e \rightsquigarrow [\{\mathsf{rec}\, x.\ e\}/x]e
$$

$$
\begin{aligned}
\underline{\mathsf{match}}\ \mathsf{into}(v)\ \{\mathsf{into}(x) \Rightarrow e\} &\rightsquigarrow [v/x]e \\
\underline{\mathsf{match}}\ \langle\rangle\ \{\langle\rangle \Rightarrow e\} &\rightsquigarrow e \\
\underline{\mathsf{match}}\ \langle v_1, v_2\rangle\ \{\langle x_1, x_2\rangle \Rightarrow e\} &\rightsquigarrow [v_2/x_2]([v_1/x_1]e) \\
\underline{\mathsf{match}}\ (\mathsf{inj}_k\, v)\ \{\mathsf{inj}_1\, x_1 \Rightarrow e_1 \mid \mathsf{inj}_2\, x_2 \Rightarrow e_2\} &\rightsquigarrow [v/x_k]e_k \\
\underline{\mathsf{match}}\ x\ \{r_i \Rightarrow e_i\}_{i\in I} &\rightsquigarrow \mathsf{match}\ x\ \{r_i \Rightarrow e_i\}_{i\in I}
\end{aligned}
$$

$$
\begin{aligned}
[v/x](\lambda y.\, e) &= \lambda y.\, [v/x]e \\
[v/x](\mathsf{return}\, v') &= \mathsf{return}\, [v/x]v' \\
[v/x](\mathsf{let}\ y = x(s);\ e') &= \begin{cases} \mathsf{let}\ y = \{e\}\,([\{e\}/x]s);\ [\{e\}/x]e' & \text{if } v = \{e\} \\ \mathsf{let}\ y = z([z/x]s);\ [z/x]e' & \text{if } v = z \end{cases} \\
[v/x](\mathsf{let}\ y = z(s);\ e') &= \mathsf{let}\ y = z([v/x]s);\ [v/x]e' \quad \text{if } x \neq z \\
[v/x](\mathsf{match}\ x\ \{r_i \Rightarrow e_i\}_{i\in I}) &= \underline{\mathsf{match}}\ v\ \{r_i \Rightarrow [v/x]e_i\}_{i\in I}
\end{aligned}
$$

$$
\begin{aligned}
[v/x]x &= v \\
[v/x]y &= y \quad (\text{if } x \neq y) \\
[v/x]\langle\rangle &= \langle\rangle \\
[v/x]\langle v_1, v_2\rangle &= \langle [v/x]v_1, [v/x]v_2\rangle \\
&\vdots
\end{aligned}
$$

Figure 5.7: Operational semantics of unrefined type assignment system

(2) (Progress 2) *If* $\cdot \vdash e : P \to N$ *then* $e \rightsquigarrow e'$ *for some* e' *or* $e = \lambda x.e'$ *for some* e'.

(3) (Preservation) *If* $\cdot \vdash e : N$ *and* $e \rightsquigarrow e'$ *then* $\cdot \vdash e' : N$.

Proof. Progress is proved by structural induction on the given type assignment derivation. Preservation is proved by structural induction on e, using the substitution lemma (the bottom admissible rule in Fig. 4.13 + (Fig. 4.12 + recursion + ADTs) = Fig. 5.5). □

To prove the soundness of the operational semantics it helps to define value substitution typings $\Gamma \vdash \sigma : \Gamma'$ in the unrefined type assignment system and parallel value substitution $[\sigma]-$ in it. It is straightforward to prove that the single sequential value substitutions in the operational semantics (in product matching) are the same as parallel substitution of both values at once: $[v_1/x_1][v_2/x_2]- = [v_1/x_1, v_2/x_2]-$.

Lemma 5.1 (Syntactic and Semantic Substitution Commute)**.**

(1) *If* $\delta \in [\![\Gamma']\!]$ *and* $\Gamma' \vdash \sigma : \Gamma$ *and* $\Gamma \vdash e : N$ *then* $[\![[\sigma]e]\!]_\delta = [\![e]\!]_{[\![\sigma]\!]_\delta}$.

(2) *If* $\delta \in [\![\Gamma']\!]$ *and* $\Gamma' \vdash \sigma : \Gamma$ *and* $\Gamma \vdash v' : P'$ *then* $[\![[\sigma]v']\!]_\delta = [\![v']\!]_{[\![\sigma]\!]_\delta}$.

(3) *If* $\delta \in [\![\Gamma']\!]$ *and* $\Gamma' \vdash \sigma : \Gamma$ *and* $\Gamma; [M] \vdash s : \uparrow Q$ *then* $[\![[\sigma]s]\!]_\delta = [\![s]\!]_{[\![\sigma]\!]_\delta}$.

(4) *If* $\delta \in [\![\Gamma']\!]$ *and* $\Gamma' \vdash \sigma : \Gamma$ *and* $\Gamma; [Q] \vdash \{r_i \Rightarrow e_i\}_{i \in I} : N$
then $[\![[\sigma]\{r_i \Rightarrow e_i\}_{i \in I}]\!]_\delta = [\![\{r_i \Rightarrow e_i\}_{i \in I}]\!]_{[\![\sigma]\!]_\delta}$.

Proof. By mutual induction on the structure of the program term. □

The preceding lemma helps us prove our operational semantics is semantically sound.

Theorem 5.2 (Soundness of Operational Semantics)**.**
If $e \rightsquigarrow e'$ *and* $\cdot \vdash e : N$ *then* $[\![e]\!]_\cdot = [\![e']\!]_\cdot$.

Proof. By structural induction on e. Use Lemma 5.1 (Syntactic and Semantic Substitution Commute) and the type assignment version of Lemma D.22 (Continuous Maps). □

A closed expression e is said to *diverge* (operationally/computationally speaking) if there is no sequence $e \rightsquigarrow e' \rightsquigarrow^* \texttt{return}\, v$ or $\lambda x.\, e''$. A denotational semantics is computationally or operationally *adequate* if the denotation of a diverging expression is the bottom element of the denotation of its type.

Theorem 5.3 (Completeness of Operational Semantics: Computational Adequacy)**.**
Suppose $\cdot \vdash e : \uparrow P$ *in the unrefined type assignment system.*
If $[\![e]\!]_\cdot \neq \bot_{[\![\uparrow P]\!]}$ *then* $e \rightsquigarrow^* \texttt{return}\, v$ *for some* v*.*

Proof. This is a corollary of the following technical lemma, similar to Lemma 6.11 of Gunter [1993], inspired by the work of Tait [1967] and Plotkin [1976], the proof of which is similar but simpler perhaps in some ways due to the use of a *focused* sequent calculus, but a bit more complicated in other ways.

We define $d \lesssim_A \mathscr{O}$ (similar to the relation $d \lesssim_t M$ in Gunter's textbook where M is a term and t is an unpolarized type) which is a Tait/Plotkin-esque logical relation for relating the operational and denotational semantics in a way suitable to the main technical lemma. The judgment $d \lesssim_A \mathscr{O}$ presupposes $d \in [\![A]\!]$ and $\cdot \vdash \mathscr{O} : A$ and $d = [\![\mathscr{O}]\!]_\cdot$ and (further) relates d,

$\mathscr{O}$, and A via the following definition by eight rules.

$$\frac{}{\bot \lesssim_A _} \qquad \frac{e \rightsquigarrow^* \mathtt{return}\, v \qquad d \lesssim_P v}{inj_1\, d \lesssim_{\uparrow P} e} \qquad \frac{e \rightsquigarrow^* \lambda x.e' \qquad f(d) \lesssim_N [v/x]e' \text{ for all } v \text{ and } d \text{ such that } d \lesssim_P v}{f \lesssim_{P\to N} e}$$

$$\frac{}{\bullet \lesssim_1 \langle\rangle} \qquad \frac{d_1 \lesssim_{P_1} v_1 \qquad d_2 \lesssim_{P_2} v_2}{(d_1,d_2) \lesssim_{P_1\times P_2} \langle v_1, v_2\rangle} \qquad \frac{d \lesssim_{P_k} v}{inj_k\, d \lesssim_{P_1+P_2} \mathtt{inj}_k\, v} \qquad \frac{d \lesssim_N e}{d \lesssim_{\downarrow N} \{e\}}$$

$$\frac{\vdash F[\mu F] \doteq P \qquad d \lesssim_P v}{d \lesssim_{\mu F} \mathtt{into}(v)}$$

Judgment $\delta \lesssim_\Gamma \sigma$ presupposes $\vdash \delta : \Gamma$ and $\cdot \vdash \sigma : \Gamma$ and $\delta = [\![\sigma]\!]_\cdot$ and has two rules:

$$\frac{}{\cdot \lesssim_\cdot \cdot} \qquad \frac{\delta \lesssim_\Gamma \sigma \qquad d \lesssim_P v}{\delta, d/x \lesssim_{\Gamma,x:P} \sigma, v/x}$$

The main technical lemma is stated: If $\Gamma \vdash \mathscr{O} : A$ and $\delta \lesssim_\Gamma \sigma$ then $[\![\mathscr{O}]\!]_\delta \lesssim_A [\sigma]\mathscr{O}$.

The induction metric is lexicographic, first, on the sum of the structural sizes[4] of the type A and the main cut types of $\Gamma \vdash e : A$, and second, on the structure of $\mathscr{O}$.

- **Case**
$$\frac{\Gamma, x : P \vdash e_0 : N_0}{\Gamma \vdash \lambda x.\, e_0 : P \to N_0}$$

[4] That is, $sz(0) = 0$ and $sz(P_1 + P_2) = sz(P_1) + sz(P_2) + 1$ and so on.

$d \lesssim_P v$	Assume
$\delta \lesssim_\Gamma \sigma$	Given
$\vdash \delta : \Gamma$	Presupposition
$\vdash \delta, d/x : \Gamma, x : P$	By rule with presupposition
$\cdot \vdash \sigma : \Gamma$	Presupposition
$\cdot \vdash \sigma, v/x : \Gamma, x : P$	By rule with presupposition
$\delta = [\![\sigma]\!].$	Presupposition
$\delta, d/x = [\![\sigma, v/x]\!].$	By def. and by presupposition
$\delta, d/x \lesssim_{\Gamma, x:P} \sigma, v/x$	By rule
$[\![e_0]\!]_{\delta, d/x} \lesssim_{N_0} [\sigma, v/x]e_0$	By induction hypothesis (i.h.)
$[\![\lambda x.\, e_0]\!]_\delta\, d \lesssim_{N_0} [\sigma, v/x]e_0$	By def.
$[\![\lambda x.\, e_0]\!]_\delta\, d \lesssim_{N_0} [v/x][\sigma]e_0$	By substitution property
$x : P \vdash [\sigma]e_0 : N_0$	By admissible rule (substitution lemma)
$\vdash \lambda x.\, [\sigma]e_0 : P \to N_0$	By rule
$\lambda x.\, [\sigma]e_0 \rightsquigarrow^* \lambda x.\, [\sigma]e_0$	By reflexivity
$[\![\lambda x.\, e_0]\!]_\delta \lesssim_{P \to N_0} \lambda x.\, [\sigma]e_0$	By rule with typing soundness and substitution soundness (Lemma 5.1)
$[\![\lambda x.\, e_0]\!]_\delta \lesssim_{P \to N_0} [\sigma](\lambda x.\, e_0)$	By def.

- **Case**

$$\frac{\Gamma \vdash v : P}{\Gamma \vdash \mathtt{return}\, v : \uparrow P}$$

$$\begin{array}{rl}
[\![v]\!]_\delta \lesssim_P [\sigma]v & \text{By i.h.} \\
\mathtt{return}\,[\sigma]v \rightsquigarrow^* \mathtt{return}\,[\sigma]v & \text{By reflexivity} \\
inj_1\,[\![v]\!]_\delta \lesssim_{\uparrow P} \mathtt{return}\,[\sigma]v & \text{By rule} \\
inj_1\,[\![v]\!]_\delta \lesssim_{\uparrow P} [\sigma](\mathtt{return}\,v) & \text{By def.} \\
[\![\mathtt{return}\,v]\!]_\delta \lesssim_{\uparrow P} [\sigma](\mathtt{return}\,v) & \text{By def.}
\end{array}$$

- **Case**

$$\frac{\Gamma;[M] \vdash s : \uparrow Q \qquad \Gamma, z : Q \vdash e_0 : N}{\Gamma \ni (x : \downarrow M) \vdash \mathsf{let}\ z{=}x(s);\ e_0 : N}$$

If $[\![s]\!]_\delta\,\delta(x) = \bot$ then apply the first rule to get the goal. Assume $[\![s]\!]_\delta\,\delta(x) \neq \bot$.

Because $\cdot \vdash \sigma(x) : \downarrow M$ by inversion[5] $\sigma(x)$ must be a thunk $\{e'\}$ and $\cdot \vdash e' : M$.

$$\begin{array}{rl}
\delta \lesssim_\Gamma \sigma & \text{Given} \\
\delta(x) \lesssim_{\Gamma(x)} \sigma(x) & \text{By inversion} \\
\delta(x) \lesssim_{\downarrow M} \{e'\} & \text{By equalities} \\
\delta(x) \neq \bot & \text{Otherwise } [\![s]\!]_\delta\,\delta(x) = \bot \\
\delta(x) \lesssim_M e' & \text{By inversion} \\
M = \overleftarrow{P} \to \uparrow Q & \text{By inversion} \\
s = \overleftarrow{v} & \text{''}
\end{array}$$

($\overleftarrow{P} \to N$ is defined by $\cdot \to N = N$ and $(P_k, \overleftarrow{P}) \to N = P_k \to (\overleftarrow{P} \to N)$)

By inversion and repeated i.h. ($e'_{\#\overleftarrow{v}-1}$ is not a subterm of $\mathsf{let}\ z{=}x(s);\ e_0$ but the first part of the lexicographic induction is strictly smaller), $e' \rightsquigarrow^* e'_0$

where *either* $e'_0 = \mathtt{return}\,v'$ and $\delta(x) = inj_1\,d$ and $d \lesssim_Q v'$

[5] Justification "by inversion" means "from what went in to the given".

or $e_0' = \lambda x_{\#\overleftarrow{v}-1}.e'_{\#\overleftarrow{v}-1}$

and $[[\sigma]v_k/x_k]e_k' \rightsquigarrow^* \lambda x_{k-1}.e'_{k-1}$ for all $k \in \{1,2,\ldots,\#\overleftarrow{v}-1\}$

and $[[\sigma]v_0/x_0]e_0' \rightsquigarrow^* \texttt{return}\, v'$

and $\delta(x)(\llbracket[\sigma]v_{\#\overleftarrow{v}-1}\rrbracket_.)\cdots(\llbracket[\sigma]v_0\rrbracket_.) = inj_1\, d$ and $d \lesssim_Q v'$.

$$\begin{array}{rll}
\delta, d/z \lesssim_{\Gamma,z:Q} \sigma, v'/z & & \text{By rule} \\
\Gamma, z : Q \vdash e_0 : N & & \text{Subderivation} \\
\llbracket e_0 \rrbracket_{\delta,d/z} \lesssim_N [\sigma, v'/z]e_0 & & \text{By i.h.} \\
\llbracket e_0 \rrbracket_{\delta,\llbracket s\rrbracket_\delta \delta(x)/z} \lesssim_N [\sigma, v'/z]e_0 & & \text{By def., equality} \\
[\sigma](\texttt{let}\ z\texttt{=}x(s);\, e_0) = \texttt{let}\ z\texttt{=}\{e'\}\,([\sigma]s);\, [\sigma]e_0 & & \text{By def.} \\
= \texttt{let}\ z\texttt{=}\{e'\}\,([\sigma]^{\overleftarrow{v}});\, [\sigma]e_0 & & \text{By equality} \\
\rightsquigarrow^* [v'/z][\sigma]e_0 & & \text{By def.} \\
= [\sigma, v'/z]e_0 & & \text{Property of substitution}
\end{array}$$

$$\begin{array}{ll}
\llbracket \texttt{let}\ z\texttt{=}x(s);\, e_0 \rrbracket_\delta \lesssim_N [\sigma, v'/z]e_0 & \text{By def.} \\
\llbracket \texttt{let}\ z\texttt{=}x(s);\, e_0 \rrbracket_\delta \lesssim_N [\sigma](\texttt{let}\ z\texttt{=}x(s);\, e_0) & \text{If } d \lesssim_N e' \text{ and } e \rightsquigarrow^* e' \\
 & \text{then } d \lesssim_N e \\
 & \text{(uses Thm. 5.2)}
\end{array}$$

- **Case**

$$\frac{\Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} : N}{\Gamma \ni (x : P) \vdash \texttt{match}\, x\, \{r_i \Rightarrow e_i\}_{i\in I} : N}$$

$$\begin{array}{rl}
\cdot \vdash \sigma(x) : P & \text{By inversion} \\
\delta(x) = \llbracket \sigma(x) \rrbracket_. & \text{By inversion}
\end{array}$$

– **Case**

$$\frac{}{\Gamma;[0] \vdash \{\} : N}$$

Impossible because otherwise $\delta(x) \in \emptyset$.

– **Case**

$$\frac{\vdash F[\mu F] \stackrel{\circ}{=} Q \qquad \Gamma, y : Q \vdash e_1 : N}{\Gamma;[\mu F] \vdash \{\mathtt{into}(y) \Rightarrow e_1\} : N}$$

$\sigma(x) = \mathtt{into}(v')$ By inversion

$\cdot \vdash v' : Q$ $''$

$$[\sigma](\mathtt{match}\ x\ \{\mathtt{into}(y) \Rightarrow e_1\}) = \underline{\mathtt{match}}\ \mathtt{into}(v')\ \{\mathtt{into}(y) \Rightarrow [\sigma]e_1\} \quad \text{By def.}$$

$$\rightsquigarrow [v'/y][\sigma]e_1 \quad \text{By def.}$$

$$= [\sigma, v'/y]e_1 \quad \text{Subst. prop.}$$

$$\delta \lesssim_\Gamma \sigma \quad \text{Given}$$

$$\delta(x) \lesssim_{\Gamma(x)} \sigma(x) \quad \text{By inversion}$$

$$[\![v']\!]_\cdot \lesssim_{\mu F} \mathtt{into}(v') \quad \text{By equalities, def.}$$

$$[\![v']\!]_\cdot \lesssim_Q v' \quad \text{By inversion}$$

$$\delta, [\![v']\!]_\cdot/y \lesssim_{\Gamma, y:Q} \sigma, v'/y \quad \text{By rule}$$

$$[\![e_1]\!]_{\delta, [\![v']\!]_\cdot/y} \lesssim_N [\sigma, v'/y]e_1 \quad \text{By i.h.}$$

$$[\![\underline{\mathtt{match}}\ v\ \{\mathtt{into}(y) \Rightarrow e_1\}]\!]_\delta \lesssim_N [\sigma, v'/y]e_1 \quad \text{By def., equality}$$

$[\![\underline{\texttt{match}}\ v\ \{\texttt{into}(y) \Rightarrow e_1\}]\!]_\delta \lesssim_N [\sigma](\underline{\texttt{match}}\ v\ \{\texttt{into}(y) \Rightarrow e_1\})$ If $d \lesssim_N e'$ and $e \rightsquigarrow^* e'$ then $d \lesssim_N e$ (uses Thm. 5.2)

– **Case**

$$\frac{\Gamma, x_1 : P_1 \vdash e_1 : N \qquad \Gamma, x_2 : P_2 \vdash e_2 : N}{\Gamma; [P_1 + P_2] \vdash \{\texttt{inj}_1\, x_1 \Rightarrow e_1 \mid \texttt{inj}_2\, x_2 \Rightarrow e_2\} : N}$$

Similar to previous subcase.

– **Case**

$$\frac{\Gamma \vdash e_1 : N}{\Gamma; [1] \vdash \{\langle\rangle \Rightarrow e_1\} : N}$$

Similar to previous subcase.

– **Case**

$$\frac{\Gamma, x_1 : P_1, x_2 : P_2 \vdash e_1 : N}{\Gamma; [P_1 \times P_2] \vdash \{\langle x_1, x_2\rangle \Rightarrow e_1\} : N}$$

Similar to previous subcase.

• **Case**

$$\frac{\Gamma, x : \downarrow N \vdash e_0 : N}{\Gamma \vdash \texttt{rec}\, x.\, e_0 : N}$$

$\bot \lesssim_N \mathtt{rec}\, x.\, [\sigma]e_0$	By rule
$\bot \lesssim_N [\sigma](\mathtt{rec}\, x.\, e_0)$	By def.
$\bot \lesssim_{\downarrow N} \{\mathtt{rec}\, x.\, [\sigma]e_0\}$	By rule
$[\![e_0]\!]^k_{\delta,-/x} \bot \lesssim_N \mathtt{rec}\, x.\, [\sigma]e_0$	Assume
$[\![e_0]\!]^k_{\delta,-/x} \bot \lesssim_{\downarrow N} \{\mathtt{rec}\, x.\, [\sigma]e_0\}$	By rule
$\delta \lesssim_\Gamma \sigma$	Given
$\delta, [\![e_0]\!]^k_{\delta,-/x} \bot /x \lesssim_{\Gamma,x:\downarrow N} \sigma, \{\mathtt{rec}\, x.\, [\sigma]e_0\}/x$	By rule
$\Gamma, x:\downarrow N \vdash e_0 : N$	Subderivation
$[\![e_0]\!]_{\delta,[\![e_0]\!]^k_{\delta,-/x}\bot/x} \lesssim_N [\sigma, \{\mathtt{rec}\, x.\, [\sigma]e_0\}/x]e_0$	By i.h.
$[\![e_0]\!]^{k+1}_{\delta,-/x} \bot \lesssim_N [\sigma, \{\mathtt{rec}\, x.\, [\sigma]e_0\}/x]e_0$	By def.
$[\![e_0]\!]^{k+1}_{\delta,-/x} \bot \lesssim_N [\{\mathtt{rec}\, x.\, [\sigma]e_0\}/x][\sigma]e_0$	By property of substitution
$[\![e_0]\!]^{k+1}_{\delta,-/x} \bot \lesssim_N \mathtt{rec}\, x.\, [\sigma]e_0$	If $d \lesssim_N e'$ and $e \rightsquigarrow^* e'$ then $d \lesssim_N e$ (uses Thm. 5.2)
$[\![e_0]\!]^{k+1}_{\delta,-/x} \bot \lesssim_N [\sigma](\mathtt{rec}\, x.\, e_0)$	By def.

The goal follows from the following sublemma,

provable by induction on A structure, using Lemma D.20 (App. lub Distributes):

If $d_0 \sqsubseteq d_1 \sqsubseteq \cdots$ is a chain in $[\![A]\!]_.$ and $d_k \lesssim_A \mathscr{O}$ for all $k \in \mathbb{N}$ then $\sqcup_{k\in\mathbb{N}} d_k \lesssim_A \mathscr{O}$.

- **Case**

$$\frac{\Gamma \vdash e' : M \qquad \Gamma; [M] \vdash s : \uparrow Q \qquad \Gamma, y : Q \vdash e_0 : N}{\Gamma \vdash \mathtt{let}\ y = \{e'\}(s);\ e_0 : N}$$

Similar to case where $e = \mathtt{let}\ z = x(s);\ e_0$ but using the i.h. on the first premise rather than using the given $\delta \lesssim_\Gamma \sigma$.

- **Case**

$$\frac{\Gamma \vdash v : P \qquad \Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} : N}{\Gamma \vdash \underline{\texttt{match}}\ v\ \{r_i \Rightarrow e_i\}_{i\in I} : N}$$

  Similar to case where $e = \texttt{match}\ x\ \{r_i \Rightarrow e_i\}_{i\in I}$ but use denotational soundness of type assignment in the case where $P = 0$ and in some cases the i.h. rather than the given $\delta \lesssim_\Gamma \sigma$.

- The remaining cases are straightforward. □

In this thesis we care about proving the metatheory of the refinement type system. We prove syntactic substitution, typing soundness, and substitution soundness; these together with the relatively standard foundation allow us to be fairly confident our system is well-behaved operationally speaking, but this thesis does not emphasize operational semantics. Instead, it emphasizes denotational semantics. Operational semantics will have a brief mention again in Chapter 7. We then provide a typing algorithm and prove it is decidable, sound, and complete.

5.3 Denotational Semantics of and Substitution in the Unrefined System

From now on, we mostly forget about previous notions of substitution for proof terms, or rather, we move on from all the systems other than the unrefined system and the refined system, and don't discuss operational semantics much further. We consider a notion of substitution appropriate to the study of the refined system and its underlying unrefined system. To this end, in Fig. 5.8 we define the parallel substitution of values for variables, add annotations to substitutions to make substitution analytic in the sense of producing cut terms with type annotations, and introduce a judgment for typing such substitutions. Where

previously substitution could change a term to a primary cut term, substitution $[\sigma]-$ now simply produces an annotation.

$$\text{Unrefined syntactic substitutions} \quad \sigma ::= \cdot \mid \sigma, v : P/x$$

$\boxed{\Gamma_0 \vdash \sigma : \Gamma}$ Under Γ_0, we know σ is a well-typed syntactic substitution for variables in Γ

$$\frac{}{\Gamma_0 \vdash \cdot : \cdot} \qquad \frac{\Gamma_0 \vdash \sigma : \Gamma \qquad \Gamma_0 \vdash v \Leftarrow P \qquad x \notin dom(\Gamma)}{\Gamma_0 \vdash \sigma, v : P/x : \Gamma, x : P}$$

$$[\sigma]^{\mathsf{h}} x = \begin{cases} x & \text{if } x \notin dom(\sigma) \text{ or } \sigma(x) = (x : P) \\ \sigma(x) & \text{else} \end{cases}$$
$$[\sigma]^{\mathsf{h}}(v : P) = ([\sigma]v : P)$$

$$[\sigma](h(s)) = ([\sigma]^{\mathsf{h}} h)([\sigma]s)$$
$$[\sigma](e : \uparrow P) = ([\sigma]e : \uparrow P)$$

$$[\sigma]x = \begin{cases} x & \text{if } x \notin dom(\sigma) \\ v & \text{if } \sigma(x) = (v : P) \end{cases}$$
$$[\sigma]\langle v_1, v_2 \rangle = \langle [\sigma]v_1, [\sigma]v_2 \rangle$$
$$\vdots$$
$$[\sigma](\mathtt{match}\ h\ \{r_i \Rightarrow e_i\}_{i \in I}) = \mathtt{match}\ \left([\sigma]^{\mathsf{h}} h\right)\ ([\sigma]\{r_i \Rightarrow e_i\}_{i \in I})$$
$$\vdots$$
$$[\sigma](\lambda x.e) = \lambda x.[\sigma]e$$
$$\vdots$$

Figure 5.8: Syntactic substitution in unrefined system

We prove syntactic properties of the unrefined system which are needed to prove the desired properties of the refined system.

Lemma 5.2 (Unrefined Syntactic Substitution)**.** *(Lemma D.1)*

Assume $\Gamma_0 \vdash \sigma : \Gamma$*. Then:*

(1) If $\Gamma \vdash h \Rightarrow P$*, then* $\Gamma_0 \vdash [\sigma]h \Rightarrow P$*.*

(2) If $\Gamma \vdash g \Rightarrow \uparrow P$*, then* $\Gamma_0 \vdash [\sigma]g \Rightarrow \uparrow P$*.*

(3) If $\Gamma \vdash v \Leftarrow P$*, then* $\Gamma_0 \vdash [\sigma]v \Leftarrow P$*.*

(4) If $\Gamma \vdash e \Leftarrow N$*, then* $\Gamma_0 \vdash [\sigma]e \Leftarrow N$*.*

(5) If $\Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$*, then* $\Gamma_0; [P] \vdash [\sigma]\{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$*.*

(6) If $\Gamma; [N] \vdash s \Rightarrow \uparrow P$*, then* $\Gamma_0; [N] \vdash [\sigma]s \Rightarrow \uparrow P$*.*

Similarly, we prove semantic properties of the unrefined system which are needed to prove the desired properties of the refined system. First, we define the semantics of the unrefined system in Figs. 5.9, 5.10, 5.11, and 5.12.

It is a standard CBPV semantics where the only effect is nontermination, modelled by the bottom element of *pointed* complete partial orders (cppos). That is, negative types are modelled by cppos. Values can't diverge, so we only need complete partial orders (cpos) to model value types.

The unrefined system is basically just the refined system with everything pertaining to indexes erased. The program terms of the unrefined system have almost the same syntax as those of the refined system, but an unrefined, recursive expression has no type annotation, and we replace the expression `unreachable` (at L) by $\texttt{diverge}_{|L|}$, which stands for an inexhaustive pattern-matching error (checking against L). The unrefined system satisfies a substitution lemma similar to that of the refined system, but its proof is simpler and does not rely on subsumption admissibility, because the unrefined system has no subtyping.

$$
\begin{aligned}
&[\![P]\!] : \mathbf{Cpo}\\
&[\![1]\!] = (\{\bullet\}, \sqsubseteq_{\{\bullet\}})\\
&[\![P \times Q]\!] = ([\![P]\!] \times [\![Q]\!], \sqsubseteq_{[\![P]\!]\times[\![Q]\!]})\\
&\qquad \text{where } (V_{11}, V_{12}) \sqsubseteq_{D_1\times D_2} (V_{21}, V_{22}) \text{ iff } V_{11} \sqsubseteq_{D_1} V_{12} \text{ and } V_{21} \sqsubseteq_{D_2} V_{22}\\
&[\![0]\!] = (\emptyset, \sqsubseteq_{\emptyset})\\
&[\![P + Q]\!] = ([\![P]\!] \uplus [\![Q]\!], \sqsubseteq_{[\![P]\!]\uplus[\![Q]\!]})\\
&\qquad \text{where } (j, V_{1j}) \sqsubseteq_{D_1\uplus D_2} (j, V_{2j}) \text{ iff } V_{1j} \sqsubseteq_{D_j} V_{2j}\\
&[\![\downarrow N]\!] = ([\![N]\!], \sqsubseteq_{[\![N]\!]})\\
&[\![\mu F]\!] = (\cup_{k\in\mathbb{N}} [\![F]\!]^k \emptyset, \sqsubseteq_{\mu[\![F]\!]})\\
&\qquad \text{where } V_1 \sqsubseteq_{\mu[\![F]\!]} V_2 \text{ iff there exists } k \in \mathbb{N} \text{ such that } V_1 \sqsubseteq_{[\![F]\!]^{k+1}\emptyset} V_2\\
&\qquad \text{and } \sqsubseteq_{[\![F_1\oplus F_2]\!]X} = \sqsubseteq_{[\![F_1]\!]X\uplus[\![F_2]\!]X}\\
&\qquad \text{and } \sqsubseteq_{[\![\hat{B}\otimes\hat{P}]\!]X} = \sqsubseteq_{[\![\hat{B}]\!]X\times[\![\hat{P}]\!]X}\\
&\qquad \text{and } \sqsubseteq_{[\![\mathrm{Id}]\!]X} = \sqsubseteq_X\\
&\qquad \text{and } \sqsubseteq_{[\![I]\!]X} = \sqsubseteq_{\{\bullet\}}\\
&\qquad \text{and } \sqsubseteq_{[\![\underline{Q}]\!]X} = \sqsubseteq_{[\![Q]\!]}
\end{aligned}
$$

$$
\begin{aligned}
&[\![N]\!] : \mathbf{Cppo}\\
&[\![P \to N]\!] = (\{f : [\![P]\!] \to [\![N]\!] \mid f \text{ is continuous}\}, \sqsubseteq_{[\![P]\!]\Rightarrow[\![N]\!]}, d \mapsto \bot_{[\![N]\!]})\\
&\qquad \text{where } f \sqsubseteq_{D\Rightarrow E} g \text{ iff } f(d) \sqsubseteq_E g(d) \text{ for all } d \in D\\
&[\![\uparrow P]\!] = ([\![P]\!] \uplus \{\bot_\uparrow\}, \left\{((1,d),(1,d')) \;\middle|\; d \sqsubseteq_{[\![P]\!]} d'\right\} \cup \left\{((2,\bot_\uparrow),d) \;\middle|\; d \in [\![\uparrow P]\!]\right\}, (2,\bot_\uparrow))
\end{aligned}
$$

Figure 5.9: Denotational semantics of unrefined types

$$
\begin{aligned}
[\![\mathscr{F}]\!] &: \mathbf{Cpo} \to \mathbf{Cpo} \\
[\![F \oplus G]\!] &= X \mapsto [\![F]\!]\, X \uplus [\![G]\!]\, X \\
[\![I]\!] &= X \mapsto \{\bullet\} \\
[\![\hat{B} \otimes \hat{P}]\!] &= X \mapsto [\![\hat{B}]\!]\, X \times [\![\hat{P}]\!]\, X \\
[\![\underline{P}]\!] &= X \mapsto [\![P]\!] \\
[\![\mathrm{Id}]\!] &= X \mapsto X
\end{aligned}
$$

$$
\begin{aligned}
\mathit{fmap}\ [\![F_1 \oplus F_2]\!]\ f &= d \mapsto \begin{cases} (1, (\mathit{fmap}\ [\![F_1]\!]\ f)\ d') & \text{if } d = (1, d') \\ (2, (\mathit{fmap}\ [\![F_2]\!]\ f)\ d') & \text{if } d = (2, d') \end{cases} \\
\mathit{fmap}\ [\![I]\!]\ f &= \mathit{id}_{\{\bullet\}} \\
\mathit{fmap}\ [\![\hat{B} \otimes \hat{P}]\!]\ f &= (d_1, d_2) \mapsto \big((\mathit{fmap}\ [\![\hat{B}]\!]\ f)\ d_1, (\mathit{fmap}\ [\![\hat{P}]\!]\ f)\ d_2\big) \\
\mathit{fmap}\ [\![\underline{P}]\!]\ f &= \mathit{id}_{[\![P]\!]} \\
\mathit{fmap}\ [\![\mathrm{Id}]\!]\ f &= f
\end{aligned}
$$

Figure 5.10: Denotational semantics of unrefined functors

$$
\begin{aligned}
&[\![\Gamma \vdash h \Rightarrow P]\!] : [\![\Gamma]\!] \to [\![P]\!] \\
&[\![x]\!]_\delta = \delta(x) \\
&[\![(v : P)]\!]_\delta = [\![v]\!]_\delta
\end{aligned}
$$

$$
\begin{aligned}
&[\![\Gamma \vdash g \Rightarrow \uparrow P]\!] : [\![\Gamma]\!] \to [\![\uparrow P]\!] \\
&[\![h(s)]\!]_\delta = [\![s]\!]_\delta\ [\![h]\!]_\delta \\
&[\![(e : \uparrow P)]\!]_\delta = [\![e]\!]_\delta
\end{aligned}
$$

Figure 5.11: Denotational semantics of unrefined heads h and bound expressions g

$$[\![\Gamma \vdash v \Leftarrow P]\!] : [\![\Gamma]\!] \to [\![P]\!]$$
$$[\![x]\!]_\delta = \delta(x)$$
$$[\![\langle\rangle]\!]_\delta = \bullet$$
$$[\![\langle v_1, v_2\rangle]\!]_\delta = ([\![v_1]\!]_\delta, [\![v_2]\!]_\delta)$$
$$[\![\mathtt{inj}_k\, v]\!]_\delta = (k, [\![v]\!]_\delta)$$
$$[\![\mathtt{into}(v)]\!]_\delta = [\![v]\!]_\delta$$
$$[\![\{e\}]\!]_\delta = [\![e]\!]_\delta$$

$$[\![\Gamma \vdash e \Leftarrow N]\!] : [\![\Gamma]\!] \to [\![N]\!]$$
$$[\![\mathtt{return}\, v]\!]_\delta = (1, [\![v]\!]_\delta)$$
$$[\![\Gamma \vdash \mathtt{let}\ x = g;\ e \Leftarrow N]\!]_\delta = \begin{cases} [\![e]\!]_{(\delta, V/x)} & \text{if } [\![g]\!]_\delta = (1, V) \\ \bot_{[\![N]\!]} & \text{if } [\![g]\!]_\delta = (2, \bot_\uparrow) \end{cases}$$
$$[\![\lambda x.\, e]\!]_\delta = V \mapsto [\![e]\!]_{(\delta, V/x)}$$
$$[\![\Gamma \vdash \mathtt{rec}\, x.\ e \Leftarrow N]\!]_\delta = \bigsqcup_{k \in \mathbb{N}} \left(V \mapsto [\![\Gamma, x : {\downarrow}N \vdash e \Leftarrow N]\!]_{\delta, V/x}\right)^k \bot_{[\![N]\!]}$$
$$[\![\mathtt{match}\ h\ \{r_i \Rightarrow e_i\}_{i \in I}]\!]_\delta = [\![\{r_i \Rightarrow e_i\}_{i \in I}]\!]_\delta\, [\![h]\!]_\delta$$

Figure 5.12: Denotational semantics of unrefined values v and expressions e

In CBPV, nontermination is regarded as an effect, so value and computation types denote different kinds of mathematical things: predomains and domains, respectively [Levy, 2004], which are both sets with some structure. Because we have recursive expressions, we must model nontermination, an effect. We use domain theory. For our (unrefined) system, we interpret (unrefined) positive types as predomains and (unrefined) negative types as domains. The only effect we consider in this thesis is nontermination (though we simulate nonexhaustive pattern-matching errors with it); we take (chain-)complete partial orders (cpos) as predomains, and pointed ((ω-)chain-)complete partial orders (cppos) as domains.

$$[\![\Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N]\!] : [\![\Gamma]\!] \to [\![P]\!] \to [\![N]\!]$$
$$[\![\{\langle\rangle \Rightarrow e\}]\!]_\delta = V \mapsto [\![e]\!]_\delta$$
$$[\![\{\langle x_1, x_2\rangle \Rightarrow e\}]\!]_\delta = (V_1, V_2) \mapsto [\![e]\!]_{(\delta, V_1/x_1, V_2/x_2)}$$
$$[\![\{\mathrm{inj}_1\, x_1 \Rightarrow e_1 \mid \mathrm{inj}_2\, x_2 \Rightarrow e_2\}]\!]_\delta = V \mapsto \begin{cases} [\![e_1]\!]_{\delta, V_1/x_1} & \text{if } V = (1, V_1) \\ [\![e_2]\!]_{\delta, V_2/x_2} & \text{if } V = (2, V_2) \end{cases}$$
$$[\![\{\}]\!]_\delta = \text{empty function}$$
$$[\![\{\mathtt{into}(x) \Rightarrow e\}]\!]_\delta = V \mapsto [\![e]\!]_{\delta, V/x}$$

$$[\![\Gamma; [N] \vdash s \Rightarrow M]\!] : [\![\Gamma]\!] \to [\![N]\!] \to [\![M]\!]$$
$$[\![v, s]\!]_\delta = f \mapsto [\![s]\!]_\delta\,(f([\![v]\!]_\delta))$$
$$[\![\cdot]\!]_\delta = V \mapsto V$$

Figure 5.13: Denotational semantics of unrefined match expressions and spines

Positive types and functors The grammar for unrefined positive types is similar to that for refined positive types (introduced later in Ch. 6), but lacks asserting and existential types, and unrefined inductive types μF are not refined by measurements. Unrefined inductive types use the unrefined functor grammar, which is the same as the refined functor grammar but uses unrefined types in constant functors.

$$P, Q \quad ::= \quad 1 \mid P \times Q \mid 0 \mid P + Q \mid {\downarrow}N \mid \mu F$$

The denotations of unrefined positive types are standard: Fig. 5.9. We briefly describe their partial orders, then describe the denotation of functors, and lastly return to the denotation of inductive types (which involve functors).

We give (the denotation of) 1 (denoting the distinguished terminal object $\{\bullet\}$) the discrete order $\{(\bullet, \bullet)\}$. For $P \times Q$ (denoting product) we use component-wise order (that is,

$(d_1,d_2) \sqsubseteq_{D_1\times D_2} (d_1',d_2')$ if $d_1 \sqsubseteq_{D_1} d_1'$ and $d_2 \sqsubseteq_{D_2} d_2'$), for 0 (denoting the initial object) we use the empty order, and for $P+Q$ (denoting coproduct, that is, disjoint union $\uplus$) we use injection-wise order ($inj_j\, d \sqsubseteq_{D_1\uplus D_2} inj_k\, d'$ if $j=k$ and $d \sqsubseteq_{D_j} d'$). We give $\downarrow N$ the order of N, that is, $\downarrow$ denotes the forgetful functor from the category **Cppo** of cppos and continuous functions to the category **Cpo** of cpos and continuous functions. Finally, $V_1 \sqsubseteq_{[\![\mu F]\!]} V_2$ if $V_1 \sqsubseteq_{[\![F]\!]^{k+1}\emptyset} V_2$ for some $k \in \mathbb{N}$, inheriting the type denotation orders as the functor is applied.

The denotations of unrefined functors are standard **Cpo** endofunctors: Fig. 5.10. We briefly describe them here. The sum functor $\oplus$ denotes a functor that sends a cpo to the disjoint union $\uplus$ of its component applications (with usual injection-wise order), and its functorial action is injection-wise. The product functor $\otimes$ denotes a functor that sends a cpo to the product $\times$ of its component applications (with usual component-wise order), and its functorial action is component-wise. The unit functor I denotes a functor sending any cpo to $1 = \{\bullet\}$ (discrete order), and its functorial action sends all morphisms to $id_{\{\bullet\}}$. The constant (type) functor $\underline{P}$ denotes a functor sending any cpo to the cpo $[\![P]\!]$, and its functorial action sends all morphisms to the identity $id_{[\![P]\!]}$ on $[\![P]\!]$. The identity functor Id denotes the identity endofunctor on **Cpo**. (Forgetting the order structure, functors also denote endofunctors on the category **Set** of sets and functions.)

We now explain the denotational semantics of our algebraic datatypes. Semantically, we build an inductive type (such as $[\![\mathsf{List}\, A]\!]$), by repeatedly applying (the denotation of) its functor specification (such as $[\![\mathsf{ListF}_A]\!]$) to the initial object $[\![0]\!] = \emptyset$. For example,

$$[\![\mathsf{List}\, A]\!] = \bigcup_{k\in\mathbb{N}} [\![\underline{1} \oplus (A \otimes \mathrm{Id})]\!]^k\, \emptyset = 1 \uplus \Big([\![A]\!] \times \big(1 \uplus \big([\![A]\!] \times \cdots \big)\big)\Big)$$

where $1 = \{\bullet\}$ (using the relatively direct functors with more complicated unrolling, discussed in Chapter 5). We denote the nil list [] by $inj_1 \bullet$, a list $x :: []$ with one term x by $inj_2(\llbracket x \rrbracket, inj_1 \bullet)$, and so on. In general, given a (polynomial) **Set** (category of sets and functions) endofunctor F (which, for this thesis, will always be the denotation of a well-formed (syntactic) functor, refined or otherwise), we define the operator $\mu -$, which takes polynomial **Set** endofunctors to sets, by $\mu F = \cup_{k \in \mathbb{N}} F^k \emptyset$. We then define $\llbracket \mu F \rrbracket = \mu \llbracket F \rrbracket$. In our system, for every well-formed (unrefined) functor F, the set $\mu \llbracket F \rrbracket$ is a *fixed point* of $\llbracket F \rrbracket$ (appendix Lemma D.8): that is, $\llbracket F \rrbracket (\mu \llbracket F \rrbracket) = \mu \llbracket F \rrbracket$ (and similarly for refined functors: appendix Lemma E.10). One can further prove it's the *least* fixed point. Strictly speaking, if we ever say "foo denotes a cpo", for example, we mean "we prove foo denotes a cpo": see Lemma D.15 (Unref. Type Denotations).

Negative types The grammar for unrefined negative types has unrefined function types $P \to N$ and unrefined upshifts $\uparrow P$, with no guarding or universal types. We give the denotational semantics of unrefined negative types in Fig. 5.9. We prove (Lemma D.15) unrefined negative types denote cppos.

$$N \quad ::= \quad P \to N \mid \uparrow P$$

Function types $P \to N$ denote continuous functions from $\llbracket P \rrbracket$ to $\llbracket N \rrbracket$ (which we sometimes write as $\llbracket P \rrbracket \Rightarrow \llbracket N \rrbracket$), where its order is defined pointwise, together with the bottom element (the "point" of "*pointed* cpo") $\bot_{\llbracket P \to N \rrbracket}$ that maps every $V \in \llbracket P \rrbracket$ to the bottom element $\bot_{\llbracket N \rrbracket}$ of $\llbracket N \rrbracket$ (that is, $\uparrow$ denotes the lift functor from **Cpo** to **Cppo**). For our purposes, this is equivalent to lifting $\llbracket P \rrbracket \in$ **Cpo** to **Cppo** and denoting arrow types by *strict* ($\bot$ goes to $\bot$) continuous functions so that function types denote **Cppo** *exponentials*.

Upshifts $\uparrow P$ denote $[\![P]\!] \uplus \{\bot_\uparrow\}$ with the lift order

$$\sqsubseteq_{[\![\uparrow P]\!]} = \left\{ (inj_1\, d, inj_1\, d') \;\middle|\; d \sqsubseteq_{[\![P]\!]} d' \right\} \cup \left\{ (inj_2\, \bot_\uparrow, d) \;\middle|\; d \in [\![\uparrow P]\!] \right\}$$

and bottom element $\bot_{[\![\uparrow P]\!]} = inj_2\, \bot_\uparrow$. We could put, say, $\bullet$ rather than $\bot_\uparrow$, but I think the latter is clearer in associating it with the bottom element of upshifts; or $\bot$ rather than $\bot_\uparrow$ but we often elide the "$[\![A]\!]$" subscript in $\bot_{[\![A]\!]}$ when clear from context.

Well-typed program terms We write $\Gamma \vdash \mathscr{O} \cdots A$ and $\Gamma; [B] \vdash \mathscr{O} \cdots A$ to stand for all six unrefined program typing judgments: $\Gamma \vdash h \Rightarrow P$ and $\Gamma \vdash g \Rightarrow \uparrow P$ and $\Gamma \vdash v \Leftarrow P$ and $\Gamma \vdash e \Leftarrow N$ and $\Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$ and $\Gamma; [N] \vdash s \Rightarrow \uparrow P$.

We define a semantic substitution δ for unrefined program variables Γ as follows

$$\frac{}{\vdash \cdot : \cdot} \qquad\qquad \frac{\vdash \delta : \Gamma}{\vdash \delta, V/x : \Gamma, x : P}$$

The denotational semantics (Figs. 5.11 and 5.12) of (unrefined) well-typed program terms of judgmental form $\Gamma \vdash \mathscr{O} \cdots A$ or $\Gamma; [B] \vdash \mathscr{O} \cdots A$ are (proven to be) continuous functions $[\![\Gamma]\!] \to [\![A]\!]$ and $[\![\Gamma]\!] \to [\![B]\!] \to [\![A]\!]$ respectively, where $[\![\Gamma]\!]$ is the set of all semantic substitutions $\vdash \delta : \Gamma$ together with component-wise order. Similarly to function type denotations, the bottom element of a $[\![\Gamma]\!] \to [\![N]\!]$ sends every $\delta \in [\![\Gamma]\!]$ to $\bot_{[\![N]\!]}$ (equivalently, we can lift source predomains and consider strict continuous functions). We only interpret typing derivations, but we often only mention the program term in semantic brackets $[\![-]\!]$. For example, if $\Gamma \vdash x \Rightarrow P$, then $[\![x]\!] = (\delta \in [\![\Gamma]\!]) \mapsto \delta(x)$. We write the application of the denotation $[\![E]\!]$ of a program term E (typed under Γ) to a semantic substitution $\delta \in [\![\Gamma]\!]$

as $[\![E]\!]_\delta$. We only mention a few of the more interesting cases of the definition of typing denotations. If $\Gamma;[N] \vdash v,s \Rightarrow M$, then

$$[\![v,s]\!] = (\delta \in [\![\Gamma]\!]) \mapsto (f \mapsto [\![s]\!]_\delta\,(f([\![v]\!]_\delta)))$$

Returner expressions denote monadic returns:

$$[\![\mathtt{return}\,v]\!]_\delta = inj_1\,[\![v]\!]_\delta$$

Let-binding denotes monadic binding:

$$[\![\mathtt{let}\,x\mathtt{=}g;\,e]\!]_\delta = \begin{cases} [\![e]\!]_{(\delta,V/x)} & \text{if } [\![g]\!]_\delta = inj_1\,V \\ \bot_{[\![N]\!]} & \text{if } [\![g]\!]_\delta = inj_2\,\bot_\uparrow \end{cases}$$

A recursive expression denotes a fixed point obtained by taking the least upper bound ($\sqcup$) of all its successive approximations:

$$[\![\Gamma \vdash \mathtt{rec}\,x.\,e \Leftarrow N]\!]_\delta = \bigsqcup_{k\in\mathbb{N}} \left(V \mapsto [\![\Gamma,x:{\downarrow}N \vdash e \Leftarrow N]\!]_{\delta,V/x}\right)^k \bot_{[\![N]\!]}$$

In the unrefined system, we have defined $\mathtt{diverge}_N$ for any unrefined N. Unfolding definitions, one can calculate

$$[\![\cdot \vdash \mathtt{diverge}_N \Leftarrow {\uparrow}{\downarrow}N]\!]_\cdot = \bot_{[\![\uparrow\downarrow N]\!]}$$

In the *refined* system we have `unreachable` expressions which check against any (simple[6])

[6]Discussed when we introduced the refined system.

refined negative type L under an inconsistent SMT context. We erase `unreachable` expressions checking against L to $\texttt{diverge}_{|L|}$ where $|L|$ erases indices. In this way, by showing a well-typed refined expression is never a bottom element, we also show that the refined system prevents nonexhaustive pattern match errors.

We will say more about the semantics of folds in Chapter 7, but note that the action of rolling and unrolling syntactic values is essentially denoted by $d \mapsto d$:

$$[\![\texttt{into}(v)]\!]_\delta = [\![v]\!]_\delta$$
$$[\![\{\texttt{into}(x) \Rightarrow e\}]\!]_\delta = V \mapsto [\![e]\!]_{\delta, V/x}$$

This works due to the fact that unrolling is sound (roughly, the denotations of each side of "$\mathrel{\mathring{=}}$" in the unrolling judgment are *equal*: if $\vdash G[\mu F] \mathrel{\mathring{=}} P$ then $[\![G]\!]\,(\mu\,[\![F]\!]) = [\![P]\!]$) and the fact that $[\![F]\!]\,(\mu\,[\![F]\!]) = \mu\,[\![F]\!]$ (and similarly for the refined system). See Lemma D.16 (Unref. Unroll Sound) and Lemma D.8 (Mu is Fixed Point).

Unrefined soundness Our proofs of (appendix) Lemma D.23 (Unrefined Typing Soundness) and (appendix) Lemma D.25 (Unrefined Substitution Soundness) use standard domain theory [Gunter, 1993].

We interpret an unrefined syntactic substitution (typing derivation) $\Gamma_0 \vdash \sigma : \Gamma$ as a (proven continuous) function $[\![\Gamma_0]\!] \to [\![\Gamma]\!]$ that takes a $\delta \in [\![\Gamma_0]\!]$ and uses δ to interpret each of the entries in σ, resulting in a semantic substitution for Γ.

$$[\![\Gamma_0 \vdash \cdot : \cdot]\!] = (\delta \in [\![\Gamma_0]\!]) \mapsto \cdot$$
$$[\![\Gamma_0 \vdash (\sigma, (v : P/x)) : (\Gamma, x : P)]\!] = (\delta \in [\![\Gamma_0]\!]) \mapsto ((([\![\sigma]\!]_\delta), [\![v]\!]_\delta / x)$$

Similarly to typing derivations, we only consider denotations of typing derivations $\Gamma_0 \vdash \sigma : \Gamma$ of substitutions, but often simply write $[\![\sigma]\!]$.

The key semantic properties of the unrefined system that we prove in the appendix are as follows.

Unrefined typing soundness says that a term typed A under Γ denotes a continuous function $[\![\Gamma]\!] \to [\![A]\!]$.

Lemma 5.3 (Unrefined Typing Soundness)**.** *(Lemma D.23)*

Assume $\vdash \delta : \Gamma$.

(1) If $\Gamma \vdash h \Rightarrow P$, *then* $[\![\Gamma \vdash h \Rightarrow P]\!]_\delta \in [\![P]\!]$.

(2) If $\Gamma \vdash g \Rightarrow \uparrow P$, *then* $[\![\Gamma \vdash g \Rightarrow \uparrow P]\!]_\delta \in [\![\uparrow P]\!]$.

(3) If $\Gamma \vdash v \Leftarrow P$, *then* $[\![\Gamma \vdash v \Leftarrow P]\!]_\delta \in [\![P]\!]$.

(4) If $\Gamma \vdash e \Leftarrow N$, *then* $[\![\Gamma \vdash e \Leftarrow N]\!]_\delta \in [\![N]\!]$.

(5) If $\Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$, *then* $[\![\Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N]\!]_\delta \in [\![P]\!] \Rightarrow [\![N]\!]$.

(6) If $\Gamma; [N] \vdash s \Rightarrow \uparrow P$, *then* $[\![\Gamma; [N] \vdash s \Rightarrow \uparrow P]\!]_\delta \in [\![N]\!] \Rightarrow [\![\uparrow P]\!]$.

Proof. By mutual induction on the structure of the given typing derivation. This lemma is mutually recursive with the succeeding lemma.[7] □

The proof of unrefined typing soundness is standard, and uses the well-known fact that a continuous function in **Cppo** has a least fixed point. Among other things, we also use the fact that $\mu [\![F]\!]$ is a fixed point of $[\![F]\!]$ (appendix Lemma D.8). We also use the soundness of unrefined unrolling.

[7] This is the style of proof found in the textbook of Gunter [1993], but it is the only place in the thesis where I have two separate mutually recursive lemma statements standing for one lemma.

Lemma 5.4 (Continuous Maps)**.** *(Lemma D.22)*

Suppose $\vdash \delta_1 : \Gamma_1$ *and* $\vdash \delta_2 : \Gamma_2$ *and* $(\Gamma_1, y : Q, \Gamma_2)$ ctx.

(1) If $\Gamma_1, y : Q, \Gamma_2 \vdash h \Rightarrow P$*, then the function* $[\![h]\!]_{\delta_1,-/y,\delta_2}$ *is continuous.*

(2) If $\Gamma_1, y : Q, \Gamma_2 \vdash g \Rightarrow \uparrow P$*, then the function* $[\![g]\!]_{\delta_1,-/y,\delta_2}$ *is continuous.*

(3) If $\Gamma_1, y : Q, \Gamma_2 \vdash v \Leftarrow P$*, then the function* $[\![v]\!]_{\delta_1,-/y,\delta_2}$ *is continuous.*

(4) If $\Gamma_1, y : Q, \Gamma_2 \vdash e \Leftarrow N$*, then the function* $[\![e]\!]_{\delta_1,-/y,\delta_2}$ *is continuous.*

(5) If $\Gamma_1, y : Q, \Gamma_2; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$*, then* $[\![\{r_i \Rightarrow e_i\}_{i \in I}]\!]_{\delta_1,-/y,\delta_2}$ *is continuous.*

(6) If $\Gamma_1, y : Q, \Gamma_2; [N] \vdash s \Rightarrow \uparrow P$*, then the function* $[\![s]\!]_{\delta_1,-/y,\delta_2}$ *is continuous.*

Proof. By mutual induction on the structure of the given typing derivation. This Lemma is mutually recursive with the preceding Lemma (see footnote there). □

Lemma 5.5 (Unrefined Substitution Typing Soundness)**.** *(Lemma D.24)*

If $\Gamma_0 \vdash \sigma : \Gamma$*, then* $\vdash [\![\sigma]\!]_\delta : \Gamma$ *for all* $\vdash \delta : \Gamma_0$*.*

Unrefined substitution soundness says that semantic and syntactic substitution commute: if E is a program term typed under Γ and $\Gamma_0 \vdash \sigma : \Gamma$ is a substitution, then $[\![[\sigma]E]\!] = [\![E]\!] \circ [\![\sigma]\!]$.

Lemma 5.6 (Unrefined Substitution Soundness)**.** *(Lemma D.25)*

Assume $\Gamma_0 \vdash \sigma : \Gamma$ *and* $\vdash \delta : \Gamma_0$*.*

(1) If $\Gamma \vdash h \Rightarrow P$*, then* $[\![\Gamma_0 \vdash [\sigma]^{\mathsf{h}} h \Rightarrow P]\!]_\delta = [\![\Gamma \vdash h \Rightarrow P]\!]_{[\![\sigma]\!]_\delta}$*.*

(2) If $\Gamma \vdash g \Rightarrow \uparrow P$*, then* $[\![\Gamma_0 \vdash [\sigma]g \Rightarrow \uparrow P]\!]_\delta = [\![\Gamma \vdash g \Rightarrow \uparrow P]\!]_{[\![\sigma]\!]_\delta}$*.*

(3) If $\Gamma \vdash v \Leftarrow P$*, then* $[\![\Gamma_0 \vdash [\sigma]v \Leftarrow P]\!]_\delta = [\![\Gamma \vdash v \Leftarrow P]\!]_{[\![\sigma]\!]_\delta}$*.*

(4) If $\Gamma \vdash e \Leftarrow N$, *then* $[\![\Gamma_0 \vdash [\sigma]e \Leftarrow N]\!]_\delta = [\![\Gamma \vdash e \Leftarrow N]\!]_{[\![\sigma]\!]_\delta}$.

(5) If $\Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$,

then $[\![\Gamma_0; [P] \vdash [\sigma]\{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N]\!]_\delta = [\![\Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N]\!]_{[\![\sigma]\!]_\delta}$.

(6) If $\Gamma; [N] \vdash s \Rightarrow \uparrow P$, *then* $[\![\Gamma_0; [N] \vdash [\sigma]s \Rightarrow \uparrow P]\!]_\delta = [\![\Gamma; [N] \vdash s \Rightarrow \uparrow P]\!]_{[\![\sigma]\!]_\delta}$.

We use unrefined type/substitution soundness to prove *refined* type/substitution soundness, discussed in Ch. 7.

Warning! While we use P, Q, R to represent positive types of any form in the *unrefined* system, we abandon this grammar in the *refined* system: there, P will be $\exists$ or Q, and Q will be $\wedge$ or R, and R will be simple in the sense of invariant under index extraction. Dually, in the refined system, N will be $\forall$ or M, and M will be $\supset$ or L, and L will be simple in the sense of invariant under index extraction.

Chapter 6

Declarative Refined System

We now present our core declarative calculus and type system.

First, we briefly discuss declarative typing judgments and their presuppositions, the syntax of program types/functors, index terms/propositions/spines, index sorts, algebras, program terms, and (logical and program) contexts. In Sec. 6.1, we discuss the mutually recursive index sorting judgments $\Xi \vdash t : \tau\ [\xi]$ and $\Xi; [\tau] \vdash \boldsymbol{t} : \kappa$, the well-formedness of (logical and program) contexts ($\Theta\ \mathsf{ctx}$ and $\Theta \vdash \Gamma\ \mathsf{ctx}$), types ($\Xi \vdash A\ \mathsf{type}[\xi]$), functors ($\Xi \vdash \mathscr{F}\ \mathsf{functor}[\xi]$), and algebras ($\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$). In Sec. 6.2 we discuss the propositional validity (or truth) judgment $\Theta \vdash \varphi\ \mathsf{true}$, index-level (hereditary) substitution $[\sigma]-$ where $\Theta_0 \vdash \sigma : \Theta$, and logical properties required of the index domain. In Sec. 6.3, we discuss subtyping/submeasuring ($\Theta \vdash A \leq B$ and $\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)$ and $\Xi \vdash \alpha; F \leq_\tau \beta; G$). In Sec. 6.4, we discuss the unrolling judgment for refined inductive types. In Sec. 6.5, we discuss the refined typing system. In Sec. 6.6, we extend substitution to that of program values for program variables, and prove a substitution lemma stating that typing is stable under substitution (a key operational correctness result).

In Fig. 6.1, we summarize the judgments defining the declarative system (only the last two judgments are used only in the metatheory), as well as their presuppositions. In the

second column, we refer to where in the chapter we introduce the judgment. In the figure, "pre." (in the third column) abbreviates "presupposes", which basically lists the judgments (fourth column) we tend to leave implicit in rules defining the given judgment (first column). Presuppositions also indicate the (input or output) moding of judgments. For example, on the one hand $\Theta;\Gamma \vdash v \Leftarrow P$ presupposes the well-formedness of P; but on the other hand, $\Theta;\Gamma \vdash h \Rightarrow P$ does not presuppose the well-formedness of P but rather we must prove that the output-moded P is well-formed (which is straightforward). The presupposition

$$\Xi \vdash \overrightarrow{\beta} : G(\mathscr{M}(F)) \Rightarrow \mathscr{M}(F)$$

is syntactic sugar for it being the case that

$$\Xi \vdash \beta_k : G(\tau_k) \Rightarrow \tau_k$$

for all $(\beta_k, (\mathsf{fold}_F _)\, v _ =_{\tau_k} _) \in zip(\overrightarrow{\beta})(\mathscr{M}(F))$[1].

Groups of mutually defined judgments are separated by blank lines.

Types Ultimately, we would like to refine algebraic datatypes μF by list measurements $\mathscr{M}$ on μF. A *measurement* is an equality of the form $\mathsf{measure}_k\ v\ \boldsymbol{t} =_\tau t$ where $\mathsf{measure}_k : \mu F \to \tau$ and $v : \mu F$ (our ADT comprehensions have the form $\{v : \mu F \mid \mathscr{M}\}$) and $\boldsymbol{t}$ is a list of index terms and projections (in other words, an index spine) of sort τ and t is an index of the result sort of τ. More precisely, $\Xi;[\tau] \vdash \boldsymbol{t} : \kappa$ and $\Xi \vdash t : \kappa$ (the indices $\boldsymbol{t}$ and t may

[1] We use the notation $\overrightarrow{\mathscr{O}}$ for lists growing rightward, and $\overleftarrow{\mathscr{O}}$ for lists growing leftward. We define the metaoperation *zip* taking n lists of the same length as inputs (where $n \geq 2$), and returning a list of n-tuples: $zip(\cdot)(\cdot) = \cdot$ and $zip(\overrightarrow{\mathscr{O}}, \mathscr{O}_k)(\overrightarrow{\mathscr{O}'}, \mathscr{O}'_k) = zip(\overrightarrow{\mathscr{O}})(\overrightarrow{\mathscr{O}'}), (\mathscr{O}_k, \mathscr{O}'_k)$ and similarly for three or more lists. We write $\mathscr{O}$ to stand for an arbitrary metavariable. We often (not always) write "_" for metavariables we "don't care about", that is, don't use.

Θ ctx	(Sec. 6.1)	pre.	no judgment
$\Xi \vdash t : \tau\ [\xi_t]$	(Sec. 6.1)	pre.	Ξ ctx
$\Xi; [\tau] \vdash \boldsymbol{t} : \kappa$	(Sec. 6.1)	pre.	Ξ ctx
$\xi \vdash \mathfrak{D}$ det	(Sec. 6.1)	pre.	no judgment
$\alpha \circ \mathsf{inj}_k \mathrel{\mathring{=}} \alpha_k$	(Sec. 6.1)	pre.	no judgment
$\Xi \vdash A$ type$[\xi_A]$	(Sec. 6.1)	pre.	Ξ ctx
$\Xi \vdash \mathscr{M}(F)$ msmts$[\xi]$	(Sec. 6.1)	pre.	Ξ ctx
$\Xi \vdash \mathscr{F}$ functor$[\xi_{\mathscr{F}}]$	(Sec. 6.1)	pre.	Ξ ctx
$\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$	(Sec. 6.1)	pre.	$\Xi \vdash F$ functor$[\xi_F]$
$\Xi \vdash \Gamma$ ctx	(Sec. 6.1)	pre.	Ξ ctx
$\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma$	(Sec. 6.2, 6.6)	pre.	Θ_0 ctx and Θ ctx and $\overline{\Theta_0} \vdash \Gamma_0$ ctx and $\overline{\Theta} \vdash \Gamma$ ctx
$\vdash \delta : \Theta; \Gamma$	(Sec. 6.2, Ch. 7)	pre.	$\overline{\Theta} \vdash \Gamma$ ctx and Θ ctx
$\Theta \vdash \varphi$ true	(Sec. 6.2)	pre.	$\overline{\Theta} \vdash \varphi : \mathbb{B}$ and Θ ctx
$\Theta \vdash u \equiv t : \tau$	(Sec. 6.3)	pre.	$\overline{\Theta} \vdash u : \tau$ and $\overline{\Theta} \vdash t : \tau$ and Θ ctx
$\Theta; [\tau] \vdash \boldsymbol{u} \equiv \boldsymbol{t} : \kappa$	(Sec. 6.3)	pre.	$\overline{\Theta}; [\tau] \vdash \boldsymbol{u} : \kappa$ and $\overline{\Theta}; [\tau] \vdash \boldsymbol{t} : \kappa$ and Θ ctx
$\Theta \vdash A \leq^{\pm} B$	(Sec. 6.3)	pre.	$\overline{\Theta} \vdash A$ type$[\xi_A]$ and $\overline{\Theta} \vdash B$ type$[\xi_B]$ and Θ ctx
$\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)$	(Sec. 6.3)	pre.	$\overline{\Theta} \vdash \mathscr{M}'(F')$ msmts$[\xi']$ and $\overline{\Theta} \vdash \mathscr{M}(F)$ msmts$[\xi]$ and Θ ctx
$\Xi \vdash \alpha; F \leq_\tau \beta; G$	(Sec. 6.3)	pre.	$\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$ and $\Xi \vdash \beta : G(\tau) \Rightarrow \tau$
$\Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \wr \mathrel{\mathring{=}} {}^{\mathsf{d}}\Theta; R$	(Sec. 6.4)	pre.	$\Xi \vdash \mathscr{M}(F)$ msmts$[\xi]$ and $\Xi \vdash \overrightarrow{\beta} : G(\mathscr{M}(F)) \Rightarrow \mathscr{M}(F)$
$\Theta; \Gamma \vdash h \Rightarrow P$	(Sec. 6.5)	pre.	Θ ctx and $\overline{\Theta} \vdash \Gamma$ ctx
$\Theta; \Gamma \vdash g \Rightarrow \uparrow P$	(Sec. 6.5)	pre.	Θ ctx and $\overline{\Theta} \vdash \Gamma$ ctx
$\Theta; \Gamma \vdash v \Leftarrow P$	(Sec. 6.5)	pre.	Θ ctx and $\overline{\Theta} \vdash \Gamma$ ctx and $\overline{\Theta} \vdash P$ type$[\xi_P]$
$\Theta; \Gamma \vdash e \Leftarrow N$	(Sec. 6.5)	pre.	Θ ctx and $\overline{\Theta} \vdash \Gamma$ ctx and $\overline{\Theta} \vdash N$ type$[\xi_N]$
$\Theta; \Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$	(Sec. 6.5)	pre.	Θ ctx and $\overline{\Theta} \vdash \Gamma$ ctx and $\overline{\Theta} \vdash P$ type$[\xi_P]$ and $\overline{\Theta} \vdash N$ type$[\xi_N]$
$\Theta; \Gamma; [N] \vdash s \Rightarrow \uparrow P$	(Sec. 6.5)	pre.	Θ ctx and $\overline{\Theta} \vdash \Gamma$ ctx and $\overline{\Theta} \vdash N$ type$[\xi_N]$
$\Theta \subseteq \Theta'$	(Sec. 6.2)	pre.	Θ ctx and Θ' ctx
$\Theta \vdash \Theta_1 \equiv \Theta_2$ ctx	(Sec. 6.4)	pre.	(Θ, Θ_1) ctx and (Θ, Θ_2) ctx

Every judgment except submeasuring, unrolling, and algebra well-formedness presupposes that no Id hypothesis occurs in its input context.

Figure 6.1: Declarative judgments and their presuppositions

mention free variables $dom(\Xi)$, but F and $\mathsf{measure}_k$ may not and indeed are closed) where κ is a *first-order* (result) sort, that is, a sort in which an arrow sort $\Rightarrow$ does not occur. It is important in our system that t, the index on the right-hand side of a measurement, be first-order: unrolling generates equalities with this t occurring in them, and we build asserting types with these equalities and must typecheck values against them, which must verify the asserted equality with this t. However, we cannot expect an SMT solver to succeed at this, so to maintain decidability we require this t to be first-order.

So, to handle unrolling of refined ADTs (which, by the way, are positive), which generates asserted equalities, we also need to refine positive types P by index propositions φ which should hold: these are called *asserting types* $Q \wedge \varphi$. We also need *existential types* $\exists a : \kappa. P$ to express the existence of indices making assertions hold. Logically, dually, in the negative polarity, we should have *guarding types* $\varphi \supset M$ and *universal types* $\forall a : \kappa. N$.

However, after working out a system for first-order single-measurements of algebraic data [Economou et al., 2023] we discovered we often needed to extract index information. For example, it simplifies the metatheory to assume types in program contexts don't have any extractable index information, so we extract such index information from types before adding them to program contexts (this is sound because adding them to program contexts means we are assuming them anyway). In that paper, we handled this using an extraction judgment, but we later discovered it is much simpler to bake extraction directly into the syntax of types.

Basically the point of the extraction judgment $\Theta \vdash A \rightsquigarrow^{\pm} A' \ [\Theta']$ was to extract from A the indices Θ' we can logically assume to hold in assuming A, resulting in a type A' whose only difference from A is its lack of the extracted indices: see Fig. 6.2. It extracted quantified variables and $\wedge$ and $\supset$ propositions from a type, outputting the same type but

$\boxed{\Theta \vdash A \rightsquigarrow^{\pm} A' \ [\Theta_A]}$ Under Θ, type A extracts to A' and Θ_A

$$\frac{P \neq \exists \text{ or } \wedge \text{ or } \times}{\Theta \vdash P \rightsquigarrow^{+} P \ [\cdot]} \qquad \frac{\Theta \vdash Q \rightsquigarrow^{+} Q' \ [\Theta_Q]}{\Theta \vdash Q \wedge \varphi \rightsquigarrow^{+} Q' \ [\varphi, \Theta_Q]} \qquad \frac{\Theta, a : \tau \vdash P \rightsquigarrow^{+} P' \ [\Theta_P]}{\Theta \vdash \exists a : \tau . P \rightsquigarrow^{+} P' \ [a : \tau, \Theta_P]}$$

$$\frac{\Theta \vdash R_1 \rightsquigarrow^{+} R_1' \ [\Theta_{R_1}] \qquad \Theta \vdash R_2 \rightsquigarrow^{+} R_2' \ [\Theta_{R_2}]}{\Theta \vdash R_1 \times R_2 \rightsquigarrow^{+} R_1' \times R_2' \ [\Theta_{R_1}, \Theta_{R_2}]}$$

$$\frac{}{\Theta \vdash \uparrow P \rightsquigarrow^{-} \uparrow P \ [\cdot]} \qquad \frac{\Theta \vdash M \rightsquigarrow^{-} M' \ [\Theta_M]}{\Theta \vdash \varphi \supset M \rightsquigarrow^{-} M' \ [\varphi, \Theta_M]}$$

$$\frac{\Theta, a : \tau \vdash N \rightsquigarrow^{-} N' \ [\Theta_N]}{\Theta \vdash \forall a : \tau . N \rightsquigarrow^{-} N' \ [a : \tau, \Theta_N]} \qquad \frac{\Theta \vdash R \rightsquigarrow^{+} R' \ [\Theta_R] \qquad \Theta \vdash L \rightsquigarrow^{-} L' \ [\Theta_L]}{\Theta \vdash R \to L \rightsquigarrow^{-} R' \to L' \ [\Theta_R, \Theta_L]}$$

Figure 6.2: Declarative extraction

without these propositions, and the context Θ_A with them. We call A' and Θ_A the type and context *extracted* from A. For negative A, everything is extracted up to an upshift. For positive A, everything is extracted up to any connective that is not $\exists$ or $\wedge$ or $\times$. If $\Theta \vdash A \rightsquigarrow^{\pm} A \ [\cdot]$, then we say A is *simple*.

However, in this thesis, we take a simpler, grammatical approach, by stratifying type syntax. First, we name the *simple* positive types as R and R' and R'' and R_0 and R_1 and so on and (dually) we name the *simple* negative types as L and L' and L'' and L_0 and L_1 and so on. Second, asserting types Q (and Q' and so on) can be formed around simple positive types R, and (dually) guarding types M (and M' and so on) can be formed around simple negative types L. Finally, existential types P (and P' and so on) can be formed around (simple positive or) asserting types Q, and (dually) universal types N (and N' and so on) can be formed around (simple negative or) guarding types M.

Types and functors are defined in Fig. 6.3. The only difference from the syntax of unrefined types and functors (Fig. 5.1) is that types in constant functors are refined and we

add $\wedge$ and $\exists$ and $\supset$ and $\forall$ and measurements $\mathscr{M}$ for μF. Types are polarized into positive (value) types P and negative (computation) types N. We write A, B and C for types of either polarity.

Types	A,B,C	$::=$	$P \mid N$
Positive types	P	$::=$	$Q \mid \exists a : \kappa. P$
	Q	$::=$	$R \mid Q \wedge \varphi$
	R	$::=$	$1 \mid R \times R \mid 0 \mid P + P \mid \downarrow N \mid \{\nu : \mu F \mid \mathscr{M}(F)\}$
Measurements on $\nu : \mu F$	$\mathscr{M}(F)$	$::=$	$\cdot_F \mid \mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\, \nu\, \boldsymbol{u} =_\tau t$
Negative types	N	$::=$	$M \mid \forall a : \kappa. N$
	M	$::=$	$L \mid \varphi \supset M$
	L	$::=$	$R \to L \mid \uparrow P$
Functors	F,G,H	$::=$	$\hat{P} \mid F \oplus F$
Constant (product) functors	$\hat{P}$	$::=$	$\hat{I} \mid \underline{P} \otimes \hat{P}$
Identity (product) functors	$\hat{I}$	$::=$	$I \mid \mathrm{Id} \otimes \hat{I}$
Base functors	$\hat{B}$	$::=$	$\underline{P} \mid \mathrm{Id}$
	$\mathscr{F}$	$::=$	$F \mid \hat{B}$

Figure 6.3: Types

After meditating on index extraction, we stratify positive types P so that existential types $\exists a : \kappa. P$ (written P) are on the outside, followed by asserting types $Q \wedge \varphi$ (written Q), with the innermost types being simple (written R) in the sense of invariance under index extraction. Positive types consist of the unit type 1 (which is simple), simple products $R_1 \times R_2$, the void type 0 (which is simple), sums $P_1 + P_2$ (which are simple) (P_1 and P_2 are not simple, that is, are not written R_1 and R_2, because we cannot extract indices from either summand even if we assume the sum holds[2]), downshifts (of negative types; *thunk* types) $\downarrow N$ (which are simple), *asserting* types $Q \wedge \varphi$ (read "Q with φ"), index-level existential quantifications $\exists a : \kappa. P$, and refined algebraic datatypes (or refined inductive types

[2]This reflects the fact that our extraction judgment did not extract under sums, which is done to minimize the amount of SMT logic in the system itself.

or comprehensions) $\{v : \mu F \mid \mathscr{M}(F)\}$ where $\mathscr{M}(F)$ is a list of measurements of the form $(\mathsf{fold}_F\ \alpha)\, v\, \boldsymbol{t} =_\tau t$. We read $\{v : \mu F \mid \mathscr{M}(F)\}$ as comprising values v of type μF (with algebraic signature F) such that every (index-level) *measurement* $(\mathsf{fold}_F\ \alpha)\, v\, \boldsymbol{t} =_\tau t$ in $\mathscr{M}(F)$ holds; in Sec. 6 and Ch. 7, we explain the metavariables F, α, τ, and t, as well as what these and the syntactic parts μ and fold mean or denote. Briefly, $\mu-$ denotes "initial fixed point of $-$" and a fold over F with α (having index carrier sort τ) denotes a semantic measure on the inductive type μF into τ. We write $\mathscr{M}(F)$ with the F in parentheses because it is convenient to have easy access to the underlying functor of a list of measurements, though we sometimes omit it.

We stratify negative types N so that universal types $\forall a : \kappa. N$ (written N) are on the outside, followed by guarding types $\varphi \supset M$ (written M), with the innermost types being simple (written L) in the sense of invariance under index extraction. Negative types consist of simple function types $R \to L$, upshifts (of positive types; *lift* or *returning* types) $\uparrow P$ (dual to $\downarrow N$), which are simple, *guarding* types $\varphi \supset M$ (read "φ implies M"; dual to $Q \wedge \varphi$), and index-level universal quantifications $\forall a : \kappa. N$ (dual to $\exists a : \kappa. P$).

In $Q \wedge \varphi$ and $\varphi \supset M$, the index proposition φ has no run-time content. Neither does the a in $\exists a : \kappa. P$ and $\forall a : \kappa. N$, nor the measurements $\mathscr{M}(F)$ in $\{v : \mu F \mid \mathscr{M}(F)\}$.

Index language: sorts, terms, and propositions Our type system is parametric in the index domain, provided the latter has certain (basic) properties. For our system to be decidable, the index domain must be decidable. We nonetheless work with a certain index domain: Fig. 6.4 includes a quantifier-free logic (ignoring λ terms: by using hereditary substutition we pass only first-order indices to "the SMT solver", but we use lambda terms to express multi-argument measures) of linear equality, inequality, arithmetic, and uninterpreted functions, which is decidable [Barrett et al., 2009].

Index variables	a, b, c, d	
Index terms	t, u, φ, ψ	$::= a \mid n \mid t + t \mid t - t \mid (t, t) \mid \lambda a.t \mid a(\boldsymbol{t})$ $\mid t = t \mid t \leq t \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \mathsf{tt} \mid \mathsf{ff}$
Index spines	$\boldsymbol{t}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$	$::= \cdot \mid t, \boldsymbol{t} \mid .1, \boldsymbol{t} \mid .2, \boldsymbol{t}$
Index sorts	τ, ω	$::= \mathbb{B} \mid \mathbb{N} \mid \mathbb{Z} \mid \tau \times \tau \mid \kappa \Rightarrow \tau$
First-order index sorts	κ	$::= \mathbb{B} \mid \mathbb{N} \mid \mathbb{Z} \mid \kappa \times \kappa$

Figure 6.4: Index domain

Sorts τ consist of booleans $\mathbb{B}$, natural numbers $\mathbb{N}$, integers $\mathbb{Z}$, products $\tau_1 \times \tau_2$, and functions $\kappa \Rightarrow \tau$. The sorts $\mathbb{B}$ and $\mathbb{N}$ and $\mathbb{Z}$ are base sorts and their set of constants are written $\mathscr{K}_\kappa$ for $\kappa \in \{\mathbb{B}, \mathbb{N}, \mathbb{Z}\}$. First-order sorts are sorts with no $\Rightarrow$ occurring in them.

Non-propositional index terms t consist of variables a, function abstractions $\lambda a.t$, applications $a(\boldsymbol{t})$ of an uninterpreted function a to an index spine $\boldsymbol{t}$, integer constants n, addition $t_1 + t_2$, subtraction $t_1 - t_2$, and pairs (t_1, t_2). An index spine is a list of indices and projections.

Propositional index terms φ are built over non-propositional index terms, and consist of equality $t_1 = t_2$, inequality $t_1 \leq t_2$, conjunction $\varphi_1 \wedge \varphi_2$, disjunction $\varphi_1 \vee \varphi_2$, negation $\neg\varphi$, trivial truth tt, and trivial falsity ff.

Measurements As we will discuss in Chapter 7, every polynomial endofunctor F has an initial fixed point μF satisfying a recursion principle for defining measures (on μF) by folds with algebras. We define algebras in Fig. 6.5. An algebra α is a list (with separator $\mid$) of clauses $p \Rightarrow t$ which pattern match on algebraic structure (p, q, and o are patterns) and bind variables in index bodies t. Sum algebra patterns p consist of $\mathtt{inj}_1\, p$ and $\mathtt{inj}_2\, p$ (which match on sum functors $\oplus$). Product algebra patterns q consist of tuples (o, q) (which match on $\otimes$) ending in the unit pattern $()$ (which matches on I). Base algebra patterns o consist of wildcard patterns $\top$ (which match on constant functors $\underline{P}$), variable patterns a (which

match on the identity functor Id), and pack patterns $\mathtt{pk}(a,o)$ (which match on *existential* constant functors $\underline{\exists a : \tau.\, P}$, where a is also bound in the body t of the algebra clause).

Algebras	α,β	$::= \cdot \mid (p \Rightarrow t \mid \alpha)$
Sum algebra patterns	p	$::= \mathtt{inj}_1\, p \mid \mathtt{inj}_2\, p \mid q$
Product algebra patterns	q	$::= () \mid (o,q)$
Base algebra patterns	o	$::= \top \mid a \mid \mathtt{pk}(a,o)$

Figure 6.5: Algebras

For example, given a type P, consider the functor $I \oplus (\underline{P} \otimes \mathrm{Id} \otimes I)$. To specify the function $\mathsf{length} : \mathsf{List}\, P \to \mathbb{N}$ computing the length of a list of values of type P, we write the algebra $\mathtt{inj}_1\, () \Rightarrow 0 \mid \mathtt{inj}_2\, (\top, (a, ())) \Rightarrow 1 + a$ with which to fold $\mathsf{List}\, P$.

With the pack algebra pattern, we can use indexes of an inductive type in our measures. For example, given $a : \mathbb{N}$, and defining the singleton type $\mathsf{Nat}(a)$ as

$$\{v : \mu \mathsf{NatF} \mid (\mathsf{fold}_{\mathsf{NatF}}\ \mathsf{ixnat})\, v = a\}$$

where $\mathsf{NatF} = I \oplus \mathrm{Id} \otimes I$ and ixnat is

$$\mathtt{inj}_1\, () \Rightarrow 0 \mid \mathtt{inj}_2\, (a, ()) \Rightarrow 1 + a$$

consider lists of natural numbers, specified by $I \oplus \underline{\exists b : \mathbb{N}.\, \mathsf{Nat}(b)} \otimes \mathrm{Id} \otimes I$. Folding such a list with the algebra

$$\mathtt{inj}_1\, () \Rightarrow 0 \mid \mathtt{inj}_2\, (\mathtt{pk}(b, \top), a, ()) \Rightarrow a + b$$

sums all the numbers in the list. (For clarity, we updated the definitions in Chapter 5 to agree with our grammars as presented in Fig. 6.3 and Fig. 6.5.) As another example, to

define the algebra for computing whether a list of natural numbers is in increasing order, we can write

$$\mathtt{inj}_1\,() \Rightarrow \lambda b.\,\mathtt{tt} \mid \mathtt{inj}_2\,(\mathtt{pk}(c,\top),(a,()) \Rightarrow \lambda b.\,c \geq b \wedge a(c)$$

Program terms Program terms are defined in Fig. 6.6. We polarize terms into two main syntactic categories: expressions (which have negative type) and values (which have positive type). Program terms are further distinguished according to whether their principal types are synthesized (heads and bound expressions) or checked (spines and patterns). There is no difference between the program terms of the refined system and the program terms of the unrefined system, except for the type annotations (in the former they are refinement types).

Program variables	x, y, z		
Expressions	e	$::=$	$\mathtt{return}\,v \mid \lambda x.\,e \mid \mathtt{rec}\,x : N.\,e \mid \mathtt{unreachable}$
			$\mid \mathtt{let}\,x = g;\,e \mid \mathtt{match}\,h\,\{r_i \Rightarrow e_i\}_{i \in I}$
Values	v	$::=$	$x \mid \langle\rangle \mid \langle v, v\rangle \mid \mathtt{inj}_1\,v \mid \mathtt{inj}_2\,v \mid \mathtt{into}(v) \mid \{e\}$
Heads	h	$::=$	$x \mid (v : P)$
Bound expressions	g	$::=$	$h(s) \mid (e : \uparrow P)$
Spines	s	$::=$	$\cdot \mid v, s$
Patterns	r	$::=$	$\mathtt{into}(x) \mid \langle\rangle \mid \langle x, y\rangle \mid \mathtt{inj}_1\,x \mid \mathtt{inj}_2\,x$

Figure 6.6: Program terms

Expressions e consist of functions $\lambda x.\,e$, recursive expressions $\mathtt{rec}\,x : N.\,e$, let-bindings $\mathtt{let}\,x = g;\,e$, match expressions $\mathtt{match}\,h\,\{r_i \Rightarrow e_i\}_{i \in I}$, value returners (or producers) $\mathtt{return}\,v$, and an unreachable expression `unreachable` (such as an impossible pattern match). Bound expressions g, which can be let-bound, consist of expressions annotated with a returner type $(e : \uparrow P)$ and applications $h(s)$ of a head h to a spine s. Heads h, which can be applied to a

spine or pattern-matched, consist of variables x and positive-type-annotated values $(v : P)$. Spines s are lists of values; we often omit the empty spine $\cdot$, writing (for example) v_1, v_2 instead of $v_1, v_2, \cdot$. In match expressions, heads are matched against patterns r.

Values consist of variables x, the unit value $\langle\rangle$, pairs $\langle v_1, v_2 \rangle$, injections into sum type $\mathtt{inj}_k\, v$ where k is 1 or 2, rollings into inductive type $\mathtt{into}(v)$, and thunks (suspended computations) $\{e\}$.

Contexts A *logical context* $\Theta ::= \cdot \mid \Theta, a \stackrel{\mathsf{d}\cdot}{\div} \tau \mid \Theta, a \div \tau \mid \Theta, a\,\mathrm{Id} \mid \Theta, \varphi$ is an ordered list of index propositions φ, Id variable hypotheses[3] $a\,\mathrm{Id}$, and (index) variable sortings $a : \tau$, (which may be used in propositions). In order to mark whether an index variable is value-determined, we split[4] the colons : of index variable sortings in two: $\div$ and $\stackrel{\mathsf{d}\cdot}{\div}$. If $a \stackrel{\mathsf{d}\cdot}{\div} \kappa$ then a is value-determined; else, if $a \div \kappa$, then a is not known to be value-determined. We write logical contexts without propositions as Ξ, and the operation $\overline{}$ merely removes propositions from logical contexts. So, for any Θ we know $\overline{\Theta}$ is a Ξ. We sometimes put a d (for "(value-)det.") superscript on a logical context: this means it is a logical context in which every : is $\stackrel{\mathsf{d}\cdot}{\div}$ (that is, in which every index variable sorting in it is value-determined). The superscript is a part of the name of the logical context: for example, ${}^{\mathsf{d}}\Xi$ (pronounced "Ξ det.") and Ξ are distinct context names, and ${}^{\mathsf{d}}\Xi$ is not to be understood as "the logical context named Ξ such that every sorting is value-determined". A *program variable context* (or *program context*) $\Gamma ::= \cdot \mid \Gamma, x : P$ is a set of (program) variable typings $x : P$.

$\boxed{\Theta\ \mathsf{ctx}}$ Input logical context Θ is well-formed

$$\frac{}{\cdot\ \mathsf{ctx}}\ \text{LogCtxEmpty} \qquad \frac{\Theta\ \mathsf{ctx} \qquad a \notin dom(\Theta)}{\begin{array}{c}(\Theta, a \div \tau)\ \mathsf{ctx} \\ (\Theta, a \overset{\mathrm{d}}{\div} \tau)\ \mathsf{ctx} \\ (\Theta, a \overset{\mathrm{d}}{\div} \tau, a\,\mathrm{Id})\ \mathsf{ctx}\end{array}}\ \text{LogCtxVar[Det][Id]}$$

$$\frac{\Theta\ \mathsf{ctx} \qquad \overline{\Theta} \vdash \varphi : \mathbb{B}}{(\Theta, \varphi)\ \mathsf{ctx}}\ \text{LogCtxProp}$$

Figure 6.7: Declarative logical context well-formedness

6.1 Context Well-Formedness, Index Sorting, and Type Well-Formedness

Figure 6.7 defines the well-formedness of logical contexts. It is defined mutually with the sorting judgment of indices: index terms in well-formed logical contexts must have boolean sort under a well-formed Ξ. Id variable hypotheses, discussed later when we introduce algebra well-formedness, in well-formed logical contexts must occur immediately after the sorting of its variable which must be value-determined.

In well-formed program variable contexts $\Xi \vdash \Gamma\ \mathsf{ctx}$, the types (of program variables) must be well-formed under (presupposed well-formed) Ξ; further, we must not be able to extract index information from these types. For example, $x : 1 \wedge \mathsf{ff}$ is an ill-formed program context because ff can be extracted, but $x : {\downarrow}{\uparrow}(1 \wedge \mathsf{ff})$ is well-formed because nothing under a shift type can be extracted.

In well-formed logical contexts and well-formed program contexts, each variable can be declared at most once.

[3] To be explained later, when we get to algebra well-formedness.

[4] Alternatively, we could have gone with $\Theta ::= \cdot \mid \Theta, a : \tau \mid \Theta, a\ \mathsf{det} \mid \Theta, a\,\mathrm{Id} \mid \Theta, \varphi$ but I didn't.

$\boxed{\Xi \vdash t : \tau\ [\xi_t]}$ Under input Ξ, input index t has input sort τ and output value-determined dependencies ξ_t
Note that $\Xi \vdash t : \tau$ abbreviates $\Xi \vdash t : \tau\ [_]$ where "_" means "don't care"

$$\frac{(a : \tau) \in \Xi}{\Xi \vdash a : \tau\ [\cdot]}\ \text{IxVar} \qquad \frac{\Xi \vdash t : \tau\ [\xi'] \qquad \xi \subsetneq \xi'}{\Xi \vdash t : \tau\ [\xi]}\ \text{IxSub} \qquad \frac{t \in \mathscr{K}_\kappa}{\Xi \vdash t : \kappa\ [\cdot]}\ \text{IxConst}$$

$$\frac{\kappa \in \{\mathbb{N}, \mathbb{Z}\} \qquad \Xi \vdash t_1 : \kappa\ [\xi_1] \qquad \Xi \vdash t_2 : \kappa\ [\xi_2]}{\Xi \vdash t_1 + t_2 : \kappa\ [\cdot] \qquad \Xi \vdash t_1 - t_2 : \kappa\ [\cdot]}\ \text{Ix}+ \text{ and Ix}-$$

$$\frac{\Xi \vdash t_1 : \tau_1\ [\xi_1] \qquad \Xi \vdash t_2 : \tau_2\ [\xi_2]}{\Xi \vdash (t_1, t_2) : \tau_1 \times \tau_2\ [\cdot]}\ \text{Ix}\times \qquad \frac{\Xi, a \div \kappa \vdash t : \tau\ [\xi_t]}{\Xi \vdash \lambda a.t : \kappa \Rightarrow \tau\ [\cdot]}\ \text{Ix}\lambda$$

$$\frac{(a : \tau) \in \Xi \qquad \Xi; [\tau] \vdash \boldsymbol{t} : \kappa}{\Xi \vdash a(\boldsymbol{t}) : \kappa\ [\cdot]}\ \text{IxApp}$$

$$\frac{(a \overset{\mathsf{d}}{\div} \kappa) \in \Xi \qquad (t \overset{\mathsf{d}}{\div} \kappa) \notin \Xi \qquad \overset{\mathsf{d}}{\div} \Xi \vdash t : \kappa\ [\xi_t]}{\Xi \vdash a = t : \mathbb{B}\ [FV(t) \to a] \qquad \Xi \vdash t = a : \mathbb{B}\ [FV(t) \to a]}\ \text{Ix=L and Ix=R}$$

$$\frac{(a \overset{\mathsf{d}}{\div} \kappa) \in \Xi \qquad (b \overset{\mathsf{d}}{\div} \kappa) \in \Xi}{\Xi \vdash a = b : \mathbb{B}\ [a \to b, b \to a]}\ \text{Ix=LR}$$

$$\frac{\Xi \vdash u_1 = t_1 : \mathbb{B}\ [\xi_1] \qquad \Xi \vdash u_2 = t_2 : \mathbb{B}\ [\xi_2]}{\Xi \vdash (u_1, u_2) = (t_1, t_2) : \mathbb{B}\ [\xi_1 \cup \xi_2]}\ \text{Ix}=\times$$

$$\frac{\text{no other rule applies} \qquad \Xi \vdash t_1 : \kappa\ [\xi_1] \qquad \Xi \vdash t_2 : \kappa\ [\xi_2]}{\Xi \vdash t_1 = t_2 : \mathbb{B}\ [\cdot]}\ \text{Ix}=$$

$$\frac{\Xi \vdash \varphi_1 : \mathbb{B}\ [\xi_1] \qquad \Xi \vdash \varphi_2 : \mathbb{B}\ [\xi_2]}{\Xi \vdash \varphi_1 \wedge \varphi_2 : \mathbb{B}\ [\xi_1 \cup \xi_2]}\ \text{Ix}\wedge \qquad \frac{\Xi \vdash \varphi_1 : \mathbb{B}\ [\xi_1] \qquad \Xi \vdash \varphi_2 : \mathbb{B}\ [\xi_2]}{\Xi \vdash \varphi_1 \vee \varphi_2 : \mathbb{B}\ [\cdot]}\ \text{Ix}\vee$$

$$\frac{\Xi \vdash \varphi : \mathbb{B}\ [\xi_\varphi]}{\Xi \vdash \neg\varphi : \mathbb{B}\ [\cdot]}\ \text{Ix}\neg \qquad \frac{\kappa \in \{\mathbb{N}, \mathbb{Z}\} \qquad \Xi \vdash t_1 : \kappa\ [\xi_1] \qquad \Xi \vdash t_2 : \kappa\ [\xi_2]}{\Xi \vdash t_1 \leq t_2 : \mathbb{B}\ [\cdot]}\ \text{Ix}\leq$$

$\boxed{\Xi; [\tau] \vdash \boldsymbol{t} : \kappa}$ Under input Ξ, fully applying an index of sort τ to $\boldsymbol{t}$ (inputs) yields an index of sort κ (output)

$$\frac{}{\Xi; [\kappa] \vdash \cdot : \kappa}\ \text{IxSpineNil} \qquad \frac{\Xi \vdash t_0 : \kappa_0 \qquad \Xi; [\tau] \vdash \boldsymbol{t} : \kappa}{\Xi; [\kappa_0 \Rightarrow \tau] \vdash t_0, \boldsymbol{t} : \kappa}\ \text{IxSpineEntry}$$

$$\frac{k \in \{1, 2\} \qquad \Xi; [\tau_k] \vdash \boldsymbol{t} : \kappa}{\Xi; [\tau_1 \times \tau_2] \vdash .k, \boldsymbol{t} : \kappa}\ \text{IxSpineProj}_k$$

Figure 6.8: Declarative index spine sorting

Figure 6.8 defines a largely standard index sorting judgment $\Xi \vdash t : \tau\ [\xi]$ presupposing Ξ ctx and saying that, under sortings Ξ, index term t has sort τ, outputting value-determined dependencies ξ. For example, $a : \mathbb{N} \vdash \neg(a \leq a+1) : \mathbb{B}\ [\cdot]$ and $a : \mathbb{N} \vdash a = 3 : \mathbb{N}\ [\emptyset{\rightarrow}a]$. We treat the sort of index sorting (except the spine judgment output) as an input, but we could do sort inference. In general we leave enhancing inference to future work.

What is novel is the ξ output, which soundly tracks value-determined dependencies. For example, if we have a value of type $1 \wedge ((a,5) = (3,b))$ then it must be the case that a and b are uniquely determined, semantically speaking: in particular, a must be equal to 3 and b must be equal to 5. However, if we have a value of type $1 \wedge ((a = 3) \vee (a = 4))$ then a is not uniquely determined, which is why rule Ix$\vee$ outputs an empty set of dependencies. If we have a value of type $1 \wedge ((a = 3) \vee (a = 3))$ it is true that a is uniquely determined, but to track this would require bringing the SMT logic into our index sorting judgment, which would greatly complicate things (if it would work at all). In this thesis, we take a *loosely coupled* approach, where we underapproximate the set of uniquely determined indices by way of a simple inspection of syntax: an equality $a = t$ or $t = a$ uniquely determines a if the free variables of t are uniquely determined, and we can collect all these dependencies that should hold conjunctively (via rules Ix$\wedge$ and Ix$=\times$). For example, a value of type $1 \wedge ((a = 1) \wedge ((2 + a = c) \wedge (b = a + c)))$ determines a to be 1 and c to be 3 and b to be 4 semantically speaking: the ξ of this type (its set of value-determined dependencies) is $\emptyset{\rightarrow}a, \{a\}{\rightarrow}c, \{a,c\}{\rightarrow}b$ which means that a is value-determined and that c is value-determined if a is value-determined and that b is value-determined if a and c are both value-determined. As such, in our system, a set of value-determined dependencies ξ is a set of Horn clauses [Horn, 1951] $\mathfrak{D}{\rightarrow}a$ where we write index variable sets as $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{C}$, or $\mathfrak{D}$. We want to track these dependencies if we can, so rule Ix$=$ is the last possible rule

you want to use for an equality index: we express this with the side condition "no other rule applies", in this case no rule with an equality index of the form of the other rules.

We can take unions and intersections $\xi \cup \xi'$ and $\xi \cap \xi'$ and the notation $\xi, \mathfrak{D}{\to}a$ means $\xi \cup \mathfrak{D}{\to}a$ together with $\mathfrak{D}{\to}a \notin \xi$.

To simplify metatheory, we only add *value-determined* dependencies: that is why, for example, rules Ix=L and Ix=R have premise $a \overset{\mathsf{d}\cdot}{\div} \kappa \in \Xi$ and not $a : \kappa \in \Xi$, and premise $\overset{\mathsf{d}\cdot}{\div}\Xi \vdash t : \kappa\ [\xi_t]$ and not $\Xi \vdash t : \kappa\ [\xi_t]$, where the $\overset{\mathsf{d}\cdot}{\div}$ operation gets the value-determined sublist of a logical context *and also* removes propositions (explained in Sec. 6.2).

Definition 6.1 (Get Value-Determined Indices ("Get Det."))**.**
For any Θ ctx, define $\overset{\mathsf{d}\cdot}{\div}(\Theta)$ by:

$$\overset{\mathsf{d}\cdot}{\div}(\cdot) = \cdot$$

$$\overset{\mathsf{d}\cdot}{\div}(\Theta, a \div \tau) = \overset{\mathsf{d}\cdot}{\div}\Theta$$

$$\overset{\mathsf{d}\cdot}{\div}(\Theta, a \overset{\mathsf{d}\cdot}{\div} \tau) = \overset{\mathsf{d}\cdot}{\div}\Theta, a \overset{\mathsf{d}\cdot}{\div} \tau$$

$$\overset{\mathsf{d}\cdot}{\div}(\Theta, a \overset{\mathsf{d}\cdot}{\div} \tau, a\,\mathrm{Id}) = \overset{\mathsf{d}\cdot}{\div}\Theta, a \overset{\mathsf{d}\cdot}{\div} \tau, a\,\mathrm{Id}$$

$$\overset{\mathsf{d}\cdot}{\div}(\Theta, \varphi) = \overset{\mathsf{d}\cdot}{\div}\Theta$$

Rule Ix$\wedge$ takes the union of the output dependencies of the conjuncts as all the dependencies still hold in the conjunction. Rule Ix$=\times$ is similar to Ix$\wedge$. The rule IxSub is only used in the metatheory for syntactic substitution, discussed later. Every other rule outputs the empty ξ: this is sound but not complete. The most important thing that we should check first when determining whether we have set up the output ξ correctly in our rules, is whether we can prove a lemma called the soundness of value-determined dependencies.

(See Lemma 7.1 in Chapter 7.) The rule

$$\frac{(a \stackrel{\mathsf{d}}{\div} \kappa) \in \Xi \qquad (t \stackrel{\mathsf{d}}{\div} \kappa) \notin \Xi \qquad \stackrel{\mathsf{d}}{\div}\Xi \vdash t : \kappa\ [\xi_t]}{\Xi \vdash a = t : \mathbb{B}\ [\xi_t \cup FV(t) \rightarrow a]}$$

for example, is unsound, so we exclude ξ_t from the output of Ix=L. If we had the unsound rule instead, the ξ of $a = (b = 0)$ would be $\emptyset \rightarrow a, \emptyset \rightarrow b$ even though neither a nor b require unique solutions to make the equality true. Both $\mathsf{tt} = (0 = 0)$ and $\mathsf{ff} = (1 = 0)$ are true but $\mathsf{tt} \neq \mathsf{ff}$ and $0 \neq 1$, violating the requirement that ξ tracks only value-determined indices.

How does an index variable b get marked in a context as determined in the first place? Ultimately, if b is determined according to some set ξ of Horn clauses: $\xi \vdash b\ \mathsf{det}$. We define the judgment as follows.

$$\frac{\emptyset \rightarrow a \in \xi}{\xi \vdash a\ \mathsf{det}}\ \text{DetUnit} \qquad\qquad \frac{\xi \vdash c\ \mathsf{det} \qquad \xi \cup \mathfrak{C} \rightarrow b \vdash a\ \mathsf{det}}{\xi, (\mathfrak{C}, c) \rightarrow b \vdash a\ \mathsf{det}}\ \text{DetCut}$$

Rule DetUnit says the variable a is determined under ξ if according to ξ it depends on nothing. In DetCut, we write $(\mathfrak{C}, c)$: this notation means $\mathfrak{C} \cup \{c\}$ together with $c \notin \mathfrak{C}$. The rule DetCut says a is determined under $\xi, (\mathfrak{C}, c) \rightarrow b$ if it is determined under $\xi \cup \mathfrak{C} \rightarrow b$ and we know that c is determined under ξ. By $\xi \vdash \mathfrak{D}\ \mathsf{det}$ we mean $\xi \vdash a\ \mathsf{det}$ for all $a \in \mathfrak{D}$. For example, $\emptyset \rightarrow a, \{a\} \rightarrow c, \{a, c\} \rightarrow b \vdash \{a, b, c\}\ \mathsf{det}$.

More abstractly, we can define a closure operator cl which takes a ξ and an initial set $\mathfrak{C}$ of index variables as inputs and computes the set of all value-determined variables:

Definition 6.2 (ξ Closure Operator)**.** Define

$$cl^0(\xi)(\mathfrak{D}) = \mathfrak{D}$$
$$cl^{n+1}(\xi)(\mathfrak{D}) = cl^n(\xi)(\mathfrak{D}) \cup \{b \mid \mathfrak{A} \subseteq cl^n(\xi)(\mathfrak{D}) \text{ for some } \mathfrak{A}{\to}b \in \xi\}$$

Define

$$cl(\xi)(\mathfrak{D}) = \bigcup_{k \in \mathbb{N}} cl^k(\xi)(\mathfrak{D})$$

Definition 6.2 (= B.1) is equivalent to the operator c of Wild [2017] who surveys pure Horn formulas. It is straightforward to prove that $cl(\xi)(-)$ is monotone: if $\mathfrak{A} \subseteq \mathfrak{B}$ then $cl(\xi)(\mathfrak{A}) \subseteq cl(\xi)(\mathfrak{B})$ for any ξ. The operator $cl(-)(\mathfrak{C})$ is also monotone (for any $\mathfrak{C}$).

We prove *cl* and det are equivalent[5]. We use these notions interchangeably and may refer to both as *dependency closure*.

Lemma 6.1 (Equivalence of *cl* and det)**.** *(Lemma B.4)*

$\xi \vdash \mathfrak{D}\ \mathsf{det}$ *if and only if* $\mathfrak{D} \subseteq cl(\xi)(\emptyset)$

Figure 6.9 defines type well-formedness $\Xi \vdash A\ \mathsf{type}[\xi]$, read "under Ξ, type A is well-formed and has value-determined dependencies ξ". The judgment $\Xi \vdash A\ \mathsf{type}[\xi]$ presupposes $\Xi\ \mathsf{ctx}$. The metavariable ξ is an output in $\Xi \vdash A\ \mathsf{type}[\xi]$, and like the output of index sorting, tracks value-determined dependencies, in this case, of comprehension types in A: the right-hand sides t of measurements $(\mathsf{fold}_F\ \alpha)\, v\, \boldsymbol{t} =_\tau t$ in $\mathscr{M}(F)$ depend on the result of the application of their measures $(\mathsf{fold}_F\ \alpha)$ (which we require to be closed) to a value v and a value-determined index spine $\boldsymbol{t}$; as well as by index equalities which must hold (as described by index sorting).

[5]It may be simpler to use only *cl*, but I just happened to use det first, and then only later, namely when I needed to prove algorithmic completeness, found it wise to use *cl*.

$\boxed{\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi_{\mathscr{M}(F)}]}$ Under input Ξ, input measurements $\mathscr{M}(F)$ are well-formed, with output value-determined dependencies $\xi_{\mathscr{M}(F)}$

$$\frac{\cdot \vdash F\ \mathsf{functor}[_]}{\Xi \vdash \cdot_F\ \mathsf{msmts}[\cdot]}$$

$$\frac{\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi] \qquad \cdot \vdash \alpha : F(\tau) \Rightarrow \tau \qquad {}^{\mathsf{d}\div}\Xi;[\tau] \vdash \boldsymbol{t} : \kappa \qquad (t\ {}^{\mathsf{d}\div}\ \kappa) \in \Xi}{\Xi \vdash \mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\ \nu\, \boldsymbol{t} =_\tau t\ \mathsf{msmts}[\xi \cup FV(\boldsymbol{t}) \to t]}$$

$$\frac{\begin{array}{c}\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi] \\ \cdot \vdash \alpha : F(\tau) \Rightarrow \tau \qquad {}^{\mathsf{d}\div}\Xi;[\tau] \vdash \boldsymbol{t} : \kappa \qquad (t\ {}^{\mathsf{d}\div}\ \kappa) \notin \Xi \qquad {}^{\mathsf{d}\div}\Xi \vdash t : \kappa\end{array}}{\Xi \vdash \mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\ \nu\, \boldsymbol{t} =_\tau t\ \mathsf{msmts}[\xi]}$$

$\boxed{\Xi \vdash A\ \mathsf{type}[\xi_A]}$ Under Ξ (input), type A (input) is well-formed, with (output) value-determined dependencies ξ_A

$$\frac{\Xi, {}^{\mathsf{d}}\Xi \vdash Q\ \mathsf{type}[\xi_Q] \qquad \xi_Q - {}^{\mathsf{d}\div}\Xi \vdash {}^{\mathsf{d}}\Xi\ \mathsf{det}}{\Xi \vdash \exists {}^{\mathsf{d}}\Xi.\, Q\ \mathsf{type}[\xi_Q - {}^{\mathsf{d}}\Xi]}\ \mathrm{DeclTp}\exists$$

$$\frac{\Xi \vdash R\ \mathsf{type}[\xi_R] \qquad \Xi \vdash \overrightarrow{\varphi} : \mathbb{B}\ [\xi_{\overrightarrow{\varphi}}]}{\Xi \vdash R \wedge \overrightarrow{\varphi}\ \mathsf{type}[\xi_R \cup \xi_{\overrightarrow{\varphi}}]}\ \mathrm{DeclTp}\wedge$$

$$\frac{\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi]}{\Xi \vdash \{\nu : \mu F \mid \mathscr{M}(F)\}\ \mathsf{type}[\xi]}\ \mathrm{DeclTp}\mu$$

$$\frac{\Xi \vdash P_1\ \mathsf{type}[\xi_1] \qquad \Xi \vdash P_2\ \mathsf{type}[\xi_2]}{\Xi \vdash P_1 + P_2\ \mathsf{type}[\cdot]}\ \mathrm{DeclTp}+$$

$$\frac{\Xi \vdash R_1\ \mathsf{type}[\xi_1] \qquad \Xi \vdash R_2\ \mathsf{type}[\xi_2]}{\Xi \vdash R_1 \times R_2\ \mathsf{type}[\xi_1 \cup \xi_2]}\ \mathrm{DeclTp}\times \qquad \frac{}{\Xi \vdash 0\ \mathsf{type}[\cdot]}\ \mathrm{DeclTp0}$$

$$\frac{}{\Xi \vdash 1\ \mathsf{type}[\cdot]}\ \mathrm{DeclTp1} \qquad \frac{\Xi \vdash N\ \mathsf{type}[\xi_N]}{\Xi \vdash \downarrow N\ \mathsf{type}[\cdot]}\ \mathrm{DeclTp}\downarrow \qquad \frac{\Xi \vdash P\ \mathsf{type}[\xi_P]}{\Xi \vdash \uparrow P\ \mathsf{type}[\cdot]}\ \mathrm{DeclTp}\uparrow$$

$$\frac{\Xi \vdash R\ \mathsf{type}[\xi_R] \qquad \Xi \vdash L\ \mathsf{type}[\xi_L]}{\Xi \vdash R \to L\ \mathsf{type}[\xi_R \cup \xi_L]}\ \mathrm{DeclTp}\to$$

$$\frac{\Xi \vdash L\ \mathsf{type}[\xi_L] \qquad \Xi \vdash \overrightarrow{\varphi} : \mathbb{B}\ [\xi_{\overrightarrow{\varphi}}]}{\Xi \vdash \overrightarrow{\varphi} \supset L\ \mathsf{type}[\xi_L \cup \xi_{\overrightarrow{\varphi}}]}\ \mathrm{DeclTp}\supset$$

$$\frac{\Xi, {}^{\mathsf{d}}\Xi \vdash M\ \mathsf{type}[\xi_M] \qquad \xi_M - {}^{\mathsf{d}\div}\Xi \vdash {}^{\mathsf{d}}\Xi\ \mathsf{det}}{\Xi \vdash \forall {}^{\mathsf{d}}\Xi.\, M\ \mathsf{type}[\xi_M - {}^{\mathsf{d}}\Xi]}\ \mathrm{DeclTp}\forall$$

Figure 6.9: Declarative well-formedness of types (and measurements)

That mostly explains rule DeclTpμ and the auxiliary rules used in its premise. The algebra/functor well-formedness premises of these auxiliary rules require the closedness of measures: $\cdot \vdash \alpha : F(\tau) \Rightarrow \tau$ has an empty context. This ensures that existential variables never appear in algebras, which will be helpful later because folds with algebras solve existential variables when typechecking a value (see Chapter 8). Previously we allowed free variables in F but this would greatly complicate the more general setting, if it would work at all. It's unclear at the moment what (if anything) free index variables in F would provide.

Rules DeclTp$\wedge$ and DeclTp$\supset$ include the dependencies of the index equalities which must hold (as described by index sorting). The notation $\Xi \vdash \overrightarrow{\varphi} : \mathbb{B}\ [\xi]$ means that for all $\varphi_k \in \overrightarrow{\varphi}$ there exists ξ_k such that $\Xi \vdash \varphi_k : \mathbb{B}\ [\xi_k]$, and also $\xi = \cup_{\varphi_k \in \overrightarrow{\varphi}} \xi_{\varphi_k}$. A value of product type is a pair of values, so we take the union of what each component value determines. We also take the union for function types $R \to L$, because to use a function, due to focusing, we must provide values for all its arguments. The ξ of $\mathsf{Nat}(a) \to \uparrow \mathsf{Nat}(a)$ is $\emptyset \to a$, so $\forall a : \mathbb{N}.\, \mathsf{Nat}(a) \to \uparrow \mathsf{Nat}(a)$ is well-formed. In applying a head of this type to a value, we must instantiate a to an index semantically equal to what that value determines; for example, if the value is one, then a gets instantiated to an index semantically equal to $1 \in \mathbb{N}$.

However, a value of sum type is either a left- or right-injected value, but we don't know which, so we output $\cdot$ in rule DeclTp$+$[6]. The unit type 1 and void (empty) type 0 both have empty ξ. We also empty out value-determined dependencies at shifts, preventing certain quantifications over shifts. For example, $\forall a : \mathbb{N}. \uparrow \mathsf{Nat}(a)$ (which is void anyway) is not

[6]Previously, we took the intersection (in that case of only unit dependencies $\emptyset \to a$), but we changed that in the upgrade to multiple higher-order measurements. We did this to keep the metatheory simple. In future work it would be interesting to add some kind of intersection (and more generally to make the syntactic tracking of value-determined indices more complete).

well-formed. Crucially, we will see that this restriction, together with focusing, guarantees indexes will be algorithmically solved by the end of certain stages. Again this restriction is sound but not complete: in future work it would be interesting to improve completeness.

In order to allow output existentials to depend on input universals, we make use of the distinction between $\overset{\mathsf{d}}{\div}$ and $\div$ in logical contexts: we require bound indices to be value-determined under the ξ of the bodies of the binders *assuming* the variables marked $\overset{\mathsf{d}}{\div}$ are determined. This makes types like

$$\forall a : \mathbb{N}.\, \mathsf{Nat}(a) \to \uparrow \exists b : \mathbb{B}.\, \{v : \mathsf{NatList} \mid \mathsf{isincr}\ v\ a = b\}$$

well-formed (this type is inhabited by any function taking a natural number and returning a list of natural numbers in increasing order starting from that input natural number, such as for example a singleton list containing that number plus 100). In this example, b is determined by the input natural number which determines a, together with how the function of this type is defined: for example, applying $\lambda x.\mathtt{return}\,[x+100]$ to 3 determines b to be true, but applying $\lambda x.\mathtt{return}\,[0]$ to 2 determines b to be false.

The way we happen to assume the variables marked $\overset{\mathsf{d}}{\div}$ are determined is by subtracting them from ξ. We define the subtraction operation on ξ as follows.

$$\cdot - a \triangleq \cdot$$

$$(\xi, \mathfrak{D}{\to}c) - a \triangleq \begin{cases} \xi - a & \text{if } c = a \\ (\xi - a) \cup ((\mathfrak{D} - a){\to}c) & \text{else} \end{cases}$$

We define the notation $\xi - \mathfrak{C}$ by $\xi - \emptyset = \xi$ and $\xi - (\mathfrak{C}, c) = (\xi - c) - \mathfrak{C}$. We define the notation $\xi - {}^{\mathsf{d}}\Xi$ by $\xi - \mathit{dom}({}^{\mathsf{d}}\Xi)$. At this point the premises $\xi_Q - \overset{\mathsf{d}}{\div}\Xi \vdash {}^{\mathsf{d}}\Xi$ det of rules

DeclTp$\exists$ and DeclTp$\forall$ should be understandable (recall the operator $\overset{\mathsf{d}}{\div}-$ gets the value-determined sublist of a logical context). These rules output the ξ of the body of the binders with the bound variables subtracted: ξ should only refer to free value-determined index variables.

We define functor and algebra well-formedness in Fig. 6.10.

Functor well-formedness $\Xi \vdash \mathscr{F}\ \mathsf{functor}[\xi]$ (presupposes $\Xi\ \mathsf{ctx}$ and is read "under Ξ functor $\mathscr{F}$ is well-formed and outputs value-determined dependencies ξ") is similar to type well-formedness: constant functors output the ξ of the underlying positive type, the sum, identity, and unit functors Id and I have empty ξ, and the product functor $\hat{B} \otimes \hat{P}$ takes the union of the component ξs. The outputs of DeclFunc$\oplus$ and DeclFunc$\otimes$ reflect the unrolling of algebraic datatypes (Sec. 6.4), which generates $+$ types from $\oplus$ functors and $\times$ types from $\otimes$ functors. That I has empty ξ reflects that 1 (unrolled from I) does too. We need to consider the output ξ of functors in order to prove that we can always form valid types with unrolling outputs.

Figure 6.10 defines algebra well-formedness $\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$ which presupposes $\Xi\ \mathsf{ctx}$ and $\Xi \vdash F\ \mathsf{functor}[\xi]$, and is read "under Ξ, algebra α is well-formed and has the 'type' $F(\tau) \Rightarrow \tau$". Rule DeclAlg$\oplus$ uses the judgment $\alpha \circ \mathsf{inj}_k \stackrel{\circ}{=} \alpha_k$ that outputs the kth clauses α_k of input algebra α. It is defined in Fig. 6.11. Comprehension type well-formedness depends on closed measures, but index variables can be bound in the body of an algebra: the rule DeclAlg$\exists$ simultaneously binds ${}^{\mathsf{d}}\Xi'$ in both t and Q, and the rule DeclAlgId only binds a in t. For metatheoretic convenience, we set up algebra well-formedness so that the pack pattern must be used even if nothing is bound in the algebra body. We mark the variables a of Id functor patterns by putting $a\mathrm{Id}$ immediately after them in well-formed logical contexts: we do this to restrict the power of submeasuring and

$\boxed{\Xi \vdash \mathscr{F}\ \mathsf{functor}[\xi_{\mathscr{F}}]}$ Under Ξ (input), functor $\mathscr{F}$ (input) is well-formed, with (output) value-determined dependencies $\xi_{\mathscr{F}}$

$$\frac{\Xi \vdash P\ \mathsf{type}[\xi]}{\Xi \vdash \underline{P}\ \mathsf{functor}[\xi]}\ \text{DeclFuncConst} \qquad \frac{}{\Xi \vdash \mathrm{Id}\ \mathsf{functor}[\cdot]}\ \text{DeclFuncId}$$

$$\frac{}{\Xi \vdash I\ \mathsf{functor}[\cdot]}\ \text{DeclFunc}I \qquad \frac{\Xi \vdash \hat{B}\ \mathsf{functor}[\xi_1] \qquad \Xi \vdash \hat{P}\ \mathsf{functor}[\xi_2]}{\Xi \vdash \hat{B} \otimes \hat{P}\ \mathsf{functor}[\xi_1 \cup \xi_2]}\ \text{DeclFunc}\otimes$$

$$\frac{\Xi \vdash F_1\ \mathsf{functor}[\xi_1] \qquad \Xi \vdash F_2\ \mathsf{functor}[\xi_2]}{\Xi \vdash F_1 \oplus F_2\ \mathsf{functor}[\cdot]}\ \text{DeclFunc}\oplus$$

$\boxed{\Xi \vdash \alpha : F(\tau) \Rightarrow \tau}$ Under Ξ (input), α (input) is a well-formed algebra of kind $F(\tau) \Rightarrow \tau$ (inputs: F and τ)

$$\frac{\begin{array}{ll}\alpha \circ \mathsf{inj}_1 \mathrel{\hat{=}} \alpha_1 & \Xi \vdash \alpha_1 : F_1(\tau) \Rightarrow \tau \\ \alpha \circ \mathsf{inj}_2 \mathrel{\hat{=}} \alpha_2 & \Xi \vdash \alpha_2 : F_2(\tau) \Rightarrow \tau\end{array}}{\Xi \vdash \alpha : (F_1 \oplus F_2)(\tau) \Rightarrow \tau}\ \text{DeclAlg}\oplus$$

$$\frac{\Xi \vdash Q\ \mathsf{type}[\xi_Q] \qquad \Xi \vdash q \Rightarrow t : \hat{P}(\tau) \Rightarrow \tau}{\Xi \vdash (\top, q) \Rightarrow t : (\underline{Q} \otimes \hat{P})(\tau) \Rightarrow \tau}\ \text{DeclAlgConst}$$

$$\frac{\Xi, {}^{\mathsf{d}}\Xi' \vdash (\top, q) \Rightarrow t : (\underline{Q} \otimes \hat{P})(\tau) \Rightarrow \tau}{\Xi \vdash (\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q) \Rightarrow t : (\underline{\exists^{\mathsf{d}}\Xi'.\, Q} \otimes \hat{P})(\tau) \Rightarrow \tau}\ \text{DeclAlg}\exists$$

$$\frac{\Xi, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id} \vdash q \Rightarrow t : \hat{I}(\tau) \Rightarrow \tau}{\Xi \vdash (a, q) \Rightarrow t : (\mathrm{Id} \otimes \hat{I})(\tau) \Rightarrow \tau}\ \text{DeclAlgId}$$

$$\frac{\overset{\mathsf{d}}{\div}\Xi \vdash t : \tau}{\Xi \vdash () \Rightarrow t : I(\tau) \Rightarrow \tau}\ \text{DeclAlg}I$$

$\boxed{\Xi \vdash \alpha : (F)\mathbb{B} \Rightarrow \mathbb{B} = _ \mapsto \mathsf{tt}}$ $\alpha : F(\mathbb{B}) \Rightarrow \mathbb{B}$ is constantly true, often writing α as $\underline{\mathsf{tt}}^{(F)}$ (inputs: α and F)

$$\frac{\Xi \vdash \alpha_1 : (F_1)\mathbb{B} \Rightarrow \mathbb{B} = _ \mapsto \mathsf{tt} \qquad \Xi \vdash \alpha_2 : (F_2)\mathbb{B} \Rightarrow \mathbb{B} = _ \mapsto \mathsf{tt}}{\Xi \vdash \mathsf{inj}_1\,\alpha_1 \mid \mathsf{inj}_2\,\alpha_2 : (F_1 \oplus F_2)\mathbb{B} \Rightarrow \mathbb{B} = _ \mapsto \mathsf{tt}}$$

$$\frac{\Xi \vdash q \Rightarrow t : (\hat{P})\mathbb{B} \Rightarrow \mathbb{B} = _ \mapsto \mathsf{tt}}{\Xi \vdash (\mathtt{pk}({}^{\mathsf{d}}\Xi, \top), q) \Rightarrow t : (\underline{\exists^{\mathsf{d}}\Xi.\, Q} \otimes \hat{P})\mathbb{B} \Rightarrow \mathbb{B} = _ \mapsto \mathsf{tt}}$$

$$\frac{\Xi \vdash q \Rightarrow t : (\hat{I})\mathbb{B} \Rightarrow \mathbb{B} = _ \mapsto \mathsf{tt}}{\Xi \vdash (a, q) \Rightarrow t : (\mathrm{Id} \otimes \hat{I})\mathbb{B} \Rightarrow \mathbb{B} = _ \mapsto \mathsf{tt}} \qquad \frac{}{\Xi \vdash () \Rightarrow \mathsf{tt} : (I)\mathbb{B} \Rightarrow \mathbb{B} = _ \mapsto \mathsf{tt}}$$

Figure 6.10: Declarative well-formedness of functors and algebras

simplify metatheory, in particular the interaction of subtyping/submeasuring and unrolling. We sort algebra bodies only when a product ends at a unit (possible by design of the functor grammar), and merely under the value-determined sublist of Ξ, which is always Ξ itself because (in algebra WF) we always begin with empty Ξ and only add value-determined variables. For simplicity, rule DeclAlgConst requires the type to be Q (so it can't be an $\exists$), forcing the use of the pack pattern (an implementation may want to relax this). The figure also defines constantly true algebras, used later to handle empty lists of measurements in subtyping.

$\boxed{\begin{array}{l}\alpha \circ \mathsf{inj}_1 \mathrel{\mathring{=}} \alpha_1 \\ \alpha \circ \mathsf{inj}_2 \mathrel{\mathring{=}} \alpha_2\end{array}}$ The left pattern of algebra α (input) is α_1 (output)
The right pattern of algebra α (input) is α_2 (output)

$$\frac{}{\cdot \circ \mathsf{inj}_1 \mathrel{\mathring{=}} \cdot} \qquad \frac{\alpha \circ \mathsf{inj}_1 \mathrel{\mathring{=}} \beta}{(\mathrm{inj}_2\, p \Rightarrow t \mid \alpha) \circ \mathsf{inj}_1 \mathrel{\mathring{=}} \beta} \qquad \frac{\alpha \circ \mathsf{inj}_1 \mathrel{\mathring{=}} \beta}{(\mathrm{inj}_1\, p \Rightarrow t \mid \alpha) \circ \mathsf{inj}_1 \mathrel{\mathring{=}} (p \Rightarrow t \mid \beta)}$$

$$\frac{}{\cdot \circ \mathsf{inj}_2 \mathrel{\mathring{=}} \cdot} \qquad \frac{\alpha \circ \mathsf{inj}_2 \mathrel{\mathring{=}} \beta}{(\mathrm{inj}_1\, p \Rightarrow t \mid \alpha) \circ \mathsf{inj}_2 \mathrel{\mathring{=}} \beta} \qquad \frac{\alpha \circ \mathsf{inj}_2 \mathrel{\mathring{=}} \beta}{(\mathrm{inj}_2\, p \Rightarrow t \mid \alpha) \circ \mathsf{inj}_2 \mathrel{\mathring{=}} (p \Rightarrow t \mid \beta)}$$

Figure 6.11: Algebra pattern selection

By restricting the bodies of algebras to (value-determined) *index terms* t and the carriers of our F-algebras to *index sorts* τ, we uphold the phase distinction: we can therefore safely refine inductive types by folding them with algebras, and also manage decidable typing.

6.2 Properties of the Index Domain

The denotations of indices are defined in Fig. A.47. We prove (Lemma C.21) well-sorted index terms and spines $\Xi \vdash t : \tau$ and $\Xi; [\omega] \vdash u : \kappa$ denote functions $[\![t]\!] : [\![\Xi]\!] \to [\![\tau]\!]$ and $[\![u]\!] : [\![\Xi]\!] \to [\![\omega]\!] \to [\![\kappa]\!]$. For each Θ ctx, we define $[\![\Theta]\!]$ to be the set of index-level

semantic substitutions (defined in this paragraph) $\{\delta \mid \vdash \delta : \Theta\}$. For example, $[\![4+a]\!]_{3/a} = 7$ and $[\![b = 1+0]\!]_{1/b} = \{\bullet\}$ (that is, true) and $[\![a = 1]\!]_{2/a} = \emptyset$ (that is, false). An *index-level semantic substitution* $\vdash \delta : \Theta$ (presupposes Θ ctx) assigns exactly one semantic index value d to each index variable in $dom(\Theta)$ such that every proposition φ in Θ is true (written $\{\bullet\}$; false is $\emptyset$):

$$\frac{}{\vdash \cdot : \cdot} \qquad \frac{\vdash \delta : \Theta \qquad d \in [\![\tau]\!] \qquad a \notin dom(\Theta)}{\vdash (\delta, d/a) : (\Theta, a : \tau[, a\,\mathrm{Id}])} \qquad \frac{\vdash \delta : \Theta \qquad [\![\varphi]\!]_\delta = \{\bullet\}}{\vdash \delta : (\Theta, \varphi)}$$

A *propositional validity* or *truth* judgment $\Theta \vdash \varphi$ true, which is a *semantic* entailment relation, holds if φ is valid under Θ, that is, if φ is true under every interpretation of variables in Θ such that all propositions in Θ are true. We say t and t' are *logically equal* or *SMT-equal* under Θ if $\Theta \vdash t = t'$ true (or equivalently $\Theta \vdash t' = t$ true).

An *index-level syntactic substitution* σ is a list of index terms to be substituted for index variables: $\sigma ::= \cdot \mid \sigma, t/a$. The metaoperation $[\sigma]\mathscr{O}$, where $\mathscr{O}$ is some syntax with free index variables, like an index term (or index spine), program term, or type, is parallel

hereditary substitution[7]:

$$\langle t \mid \cdot \rangle = t$$

$$\langle b \mid \boldsymbol{u} \rangle = b(\boldsymbol{u}) \quad \text{if } \boldsymbol{u} \neq \cdot$$

$$\langle \lambda b.t \mid u_0, \boldsymbol{u} \rangle = \langle [u_0/b]t \mid \boldsymbol{u} \rangle$$

$$\langle (t_1, t_2) \mid .k, \boldsymbol{u} \rangle = \langle t_k \mid \boldsymbol{u} \rangle \quad \text{if } k \in \{1,2\}$$

$$\langle a(\boldsymbol{t}) \mid \boldsymbol{u} \rangle = \langle a \mid \boldsymbol{t}, \boldsymbol{u} \rangle$$

$$\langle t \mid \boldsymbol{u} \rangle \text{ is undefined} \quad \text{for inputs } t \text{ and } \boldsymbol{u} \text{ not matching the above patterns}$$

$$(\langle t \mid \boldsymbol{u} \rangle \text{ is defined if } \Xi \vdash t : \tau \text{ and } \Xi; [\tau] \vdash \boldsymbol{u} : \kappa \text{ by Lemma 6.2})$$

$$[\sigma](a(\boldsymbol{u})) = \begin{cases} u & \text{if } \langle \sigma(a) \mid [\sigma]\boldsymbol{u} \rangle = u \\ a([\sigma]\boldsymbol{u}) & \text{if } a \notin dom(\sigma) \end{cases}$$

$$[\sigma](a(\boldsymbol{u})) \text{ is undefined} \quad \text{if } a \in dom(\sigma) \text{ and } \langle \sigma(a) \mid [\sigma]\boldsymbol{u} \rangle \text{ is undefined}$$

$$([\sigma](a(\boldsymbol{u})) \text{ is defined if } a \in dom(\sigma),\ \Xi_0 \vdash \sigma : \Xi \text{ and } \Xi; [\tau] \vdash \boldsymbol{u} : \kappa \text{ by Lemma 6.2})$$

$$[\sigma]a = \begin{cases} \sigma(a) & \text{if } a \in dom(\sigma) \\ a & \text{else} \end{cases}$$

$$[\sigma](t_1 + t_2) = [\sigma]t_1 + [\sigma]t_2$$

$$\vdots$$

$$[\sigma]\mathsf{tt} = \mathsf{tt}$$

$$[\sigma](\neg\varphi) = \neg([\sigma]\varphi)$$

$$\vdots$$

[7] Previously[Economou et al., 2023], it was sequential substitution, but we found parallel substitution simpler when we extended the system with multi-argument measures, which also necessitated making the substitution hereditary.

Index substitution is hereditary, that is, performs reduction as it substitutes, in order to maintain the form of uninterpreted functions where a variable is always the head of an application, and never a lambda term.

Syntactic substitutions (index-level) are typed ("sorted") in a largely standard way. Because syntactic substitutions substitute terms that may have free variables, their judgment form includes a context to the left of the turnstile, in contrast to semantic substitution:

$$\frac{}{\Theta_0 \vdash \cdot : \cdot} \qquad \frac{\Theta_0 \vdash \sigma : \Theta \qquad \overline{\Theta_0} \vdash t : \tau \qquad a \notin dom(\Theta)}{\Theta_0 \vdash (\sigma, t/a) : (\Theta, a \div \tau)}$$

$$\frac{\Theta_0 \vdash \sigma : \Theta \qquad \overset{\mathsf{d}\cdot}{\div}\Theta_0 \vdash t : \tau \qquad a \notin dom(\Theta)}{\Theta_0 \vdash (\sigma, t/a) : (\Theta, a \overset{\mathsf{d}\cdot}{\div} \tau[, a\,\mathrm{Id}])} \qquad \frac{\Theta_0 \vdash \sigma : \Theta \qquad \Theta_0 \vdash [\sigma]\varphi\ \mathsf{true}}{\Theta_0 \vdash \sigma : (\Theta, \varphi)}$$

What's nonstandard (or new) is the third rule, which should be read as two rules:

$$\frac{\Theta_0 \vdash \sigma : \Theta \qquad \overset{\mathsf{d}\cdot}{\div}\Theta_0 \vdash t : \tau \qquad a \notin dom(\Theta)}{\Theta_0 \vdash (\sigma, t/a) : (\Theta, a \overset{\mathsf{d}\cdot}{\div} \tau)}$$

$$\frac{\Theta_0 \vdash \sigma : \Theta \qquad \overset{\mathsf{d}\cdot}{\div}\Theta_0 \vdash t : \tau \qquad a \notin dom(\Theta)}{\Theta_0 \vdash (\sigma, t/a) : (\Theta, a \overset{\mathsf{d}\cdot}{\div} \tau, a\,\mathrm{Id})}$$

We require well-typed substitutions to substitute *value-determined* index terms, terms whose free variables are value-determined, for value-determined index variables. As a consequence, substitution preserves well-formedness.

Whenever we want to apply a substitution to a value-determined index, it should result in a value-determined index, so that all its free variables are determined and hence still get

subtracted in the key side premise of DeclTp$\exists$ and DeclTp$\forall$. Whenever a substitution is applied to a value-determined index, it is first restricted to its value-determined part. Now we can explain why $\overset{\mathsf{d}}{\div}-$ also removes propositions: otherwise, $\Theta_0 \vdash \sigma : \Theta$ would not imply $\overset{\mathsf{d}}{\div}\Theta_0 \vdash \sigma\restriction_{\overset{\mathsf{d}}{\div}\Theta} : \overset{\mathsf{d}}{\div}\Theta$. Consider

$$\Theta = a \overset{\mathsf{d}}{\div} \mathbb{N}, b \div \mathbb{N}, c \overset{\mathsf{d}}{\div} \mathbb{N}, b > a-1, b < a+1, b > c-1, b < c+1$$

We have $\Theta \vdash \Theta/\Theta : \Theta, a = c$ but $a \overset{\mathsf{d}}{\div} \mathbb{N}, c \overset{\mathsf{d}}{\div} \mathbb{N}, b > a-1, b < a+1, b > c-1, b < c+1$ is not even a well-formed context because b is no longer in scope for any of the propositions, but which are needed to verify $a = c$. To get this to work, we'd have to strengthen the coupling of our system to the SMT logic, but in this thesis we attempt to maximize the looseness of SMT coupling.

Derivations with logical contexts often need to be weakened in the metatheory, and for this it's convenient to define the following judgment. Figure 6.12 defines the judgment $\Theta \subseteq \Theta'$ which presupposes Θ ctx and Θ' ctx and says Θ is a subcontext of Θ'.

$\boxed{\Theta \subseteq \Theta'}$ Input logical context Θ is a subcontext of input Θ'

$$\frac{}{\cdot \subseteq \cdot} \qquad \frac{\Theta \subseteq \Theta'}{\Theta \subseteq \Theta', a : \tau} \qquad \frac{\Theta \subseteq \Theta'}{\Theta \subseteq \Theta', \varphi} \qquad \frac{\Theta \subseteq \Theta'}{\begin{array}{c}\Theta, a \div \tau \subseteq \Theta', a \div \tau \\ \Theta, a \overset{\mathsf{d}}{\div} \tau \subseteq \Theta', a \overset{\mathsf{d}}{\div} \tau \\ \Theta, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id} \subseteq \Theta', a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id}\end{array}}$$

$$\frac{\Theta \subseteq \Theta'}{\Theta, \varphi \subseteq \Theta', \varphi}$$

Figure 6.12: Logical subcontext

The decidability of our system depends on the decidability of propositional validity. The first-order fragment of our index domain is decidable [Barrett et al., 2009]. We use

hereditary substitution to SMT-verify indices only in the first-order fragment. Our system is parametric in the index domain, provided the latter has certain properties. In particular, propositional validity must satisfy the following basic properties required of a logical theory (Θ ctx is logical context well-formedness).

- *Reflexivity*[8]: If $\Theta_1, \varphi, \Theta_2$ ctx then $\Theta_1, \varphi, \Theta_2 \vdash \varphi$ true.
- *Transitivity*[9]: If $\Theta_1 \vdash \psi$ true and $\Theta_1, \psi, \Theta_2 \vdash \varphi$ true then $\Theta_1, \Theta_2 \vdash \varphi$ true.
- *Weakening*: If $\Theta \subseteq \Theta'$ and $\Theta \vdash \varphi$ true then $\Theta' \vdash \varphi$ true.
- *Substitution*: If $\Theta \vdash \varphi$ true and $\Theta_0 \vdash \sigma : \Theta$ then $\Theta_0 \vdash [\sigma]\varphi$ true.
- *Equivalence*: The relation $\Theta \vdash t_1 = t_2$ true is an equivalence relation.
- *Consistency*: It is not the case that $\cdot \vdash \mathsf{ff}$ true.

We also assume that $\Xi \vdash t : \tau\ [\xi]$ (and the mutually recursive $\Xi; [\tau] \vdash \boldsymbol{t} : \kappa$) is decidable and satisfies weakening and substitution. Further, the ξ output should be *sound*: if $\Xi \vdash \varphi : \mathbb{B}\ [\xi]$ and $\delta_1, \delta_2 \in [\![\Xi]\!]$ and $[\![\varphi]\!]_{\delta_1} = \{\bullet\} = [\![\varphi]\!]_{\delta_2}$ then for all $\mathfrak{D} \to a \in \xi$ if $\delta_1 \restriction_{\mathfrak{D}} = \delta_2 \restriction_{\mathfrak{D}}$ then $\delta_1(a) = \delta_2(a)$.[10] (For convenience, we define the judgment $\delta_1 \restriction_\xi = \delta_2 \restriction_\xi$ by "for all $\mathfrak{D} \to a \in \xi$ if $\delta_1 \restriction_{\mathfrak{D}} = \delta_2 \restriction_{\mathfrak{D}}$ then $\delta_1(a) = \delta_2(a)$".)

Our example index domain satisfies all these properties.

[8] Also called "Assumption". It also looks like identity arrows in a category or the distinguished initial rule in a sequent calculus. It's not exactly what we usually call reflexivity (xRx) but this slight abuse (perhaps) of language seems similar to that of "Transitivity", discussed next footnote.

[9] Also called "Consequence". It also looks like arrow composition in a (multi [Došen, 1997]) category or a cut in a sequent calculus. Ripley [2017] rightly points out that "transitivity" is a bit of a misnomer here (it's not exactly $xRy \wedge yRz \supset xRz$), but there seems to be a tradition for describing such a property of consequence relations in this way, which I don't mind following.

[10] We define $\delta \restriction_{\mathfrak{D}}$ by $\cdot \restriction_{\mathfrak{D}} = \cdot$ and $(\delta, d/a) \restriction_{\mathfrak{D}} = \begin{cases} \delta \restriction_{\mathfrak{D}}, d/a & \text{if } a \in \mathfrak{D} \\ \delta \restriction_{\mathfrak{D}} & \text{else} \end{cases}$

Proving the substitution lemma at the index level is tricky due to the use of hereditary substitution. It is mutually recursive with sorting the active part of hereditary substitution: part (3) below.

Lemma 6.2 (Ix. Syntactic Substitution)**.** *(Lemma C.17)*

(1) If $\Xi \vdash t : \tau\ [\xi_t]$ *and* $\Xi_0 \vdash \sigma : \Xi$ *then* $\Xi_0 \vdash [\sigma]t : \tau\ [{}^{\mathsf{d}}[\sigma]\xi_t]$

(2) If $\Xi; [\tau] \vdash \boldsymbol{t} : \kappa$ *and* $\Xi_0 \vdash \sigma : \Xi$ *then* $\Xi_0; [\tau] \vdash [\sigma]\boldsymbol{t} : \kappa$.

(3) If $\Xi_0 \vdash u : \omega$ *and* $\Xi_0; [\omega] \vdash \boldsymbol{t} : \kappa$ *then* $\Xi_0 \vdash \langle u \mid \boldsymbol{t} \rangle : \kappa$.

Follow the link to the proof in the appendix to see the complicated induction metric. It uses some properties of substitution on ξ like the fact that it distributes over set union. Hereditary substitution also complexifies a needed distribution property and its induction metric.

Lemma 6.3 (Ix. Barendregt)**.** *(Lemma C.20)*

Assume $\Xi_0 \vdash \sigma : \Xi_1$ *and* $\Xi_1, \Xi_2 \vdash \sigma' : \Xi'$

and $dom(\Xi') \cap dom(\Xi_0) = \emptyset$ *and* $dom(\Xi') \neq \emptyset$.

(1) If $\Xi_1, \Xi', \Xi_2 \vdash t : \tau$ *then* $[\sigma][\sigma']t = [[\sigma]\sigma'][\sigma]t$.

(2) If $\Xi_1, \Xi', \Xi_2; [\tau] \vdash \boldsymbol{t} : \kappa$ *then* $[\sigma][\sigma']\boldsymbol{t} = [[\sigma]\sigma'][\sigma]\boldsymbol{t}$.

(3) If $\Xi_1, \Xi_2 \vdash u : \omega$ *and* $\Xi_1, \Xi_2; [\omega] \vdash \boldsymbol{t} : \kappa$ *then* $[\sigma]\langle u \mid \boldsymbol{t} \rangle = \langle [\sigma]u \mid [\sigma]\boldsymbol{t} \rangle$.

We often implicitly use this lemma and similar lemmas using it, like Lemma C.38 (Type/Functor Barendregt).

We also prove index level syntactic and semantic substitution commute.

Lemma 6.4 (Index Substitution Soundness)**.** *(Lemma C.24)*

Assume $\Xi_0;\cdot \vdash \sigma : \Xi;\cdot$ *and* $\vdash \delta : \Xi_0;\cdot$.

(1) If $\Xi \vdash t : \tau\ [\xi_t]$ *then* $[\![[\sigma]t]\!]_\delta = [\![t]\!]_{[\![\sigma]\!]_\delta}$ *(for any derivation* $\Xi_0 \vdash [\sigma]t : \tau\ [{}^{\mathsf{d}}[\sigma]\xi_t]$*).*

(2) If $\Xi;[\tau] \vdash \boldsymbol{t} : \kappa$ *then* $[\![[\sigma]\boldsymbol{t}]\!]_\delta = [\![\boldsymbol{t}]\!]_{[\![\sigma]\!]_\delta}$ *(for any derivation* $\Xi_0;[\tau] \vdash [\sigma]\boldsymbol{t} : \kappa$*).*

(3) If $\Xi_0 \vdash u : \omega$ *and* $\Xi_0;[\omega] \vdash \boldsymbol{t} : \kappa$ *then* $[\![\langle u \mid \boldsymbol{t}\rangle]\!]_\delta = [\![\boldsymbol{t}]\!]_\delta\,[\![u]\!]_\delta$.

This is necessary to prove the semantic soundness of syntactic substitution typing. That is, a syntactic substitution denotes a semantic substitution: for example (at the index level), Lemma C.25.

We prove type well-formedness is stable under syntactic substitution: Lemma C.51 (WF Syn. Substitution). This uses weakening and some technical lemmas pertaining to ξ.

6.3 Subtyping and Submeasuring

Declarative subtyping $\Theta \vdash A \leq^{\pm} B$ is defined in Fig. 6.13. Like in the PhD thesis of Dunfield [2007b], our subtyping relation mixes syntax and semantics: it is based on syntax at the level of types but semantics at the level of indices. An SMT solver should soundly and completely implement semantic entailment.

Subtyping is polarized into mutually recursive positive $\Theta \vdash P' \leq^{+} P$ and negative $\Theta \vdash N \leq^{-} N'$ relations. The design of inference rules for subtyping is guided by sequent calculi, most clearly seen in the left and right rules pertaining to quantifiers ($\exists$, $\forall$), asserting types ($\wedge$), and guarding types ($\supset$). This is helpful to establish key properties such as *reflexivity* and *transitivity* (viewing subtyping as a sequent system, we might instead say that the structural *identity* and *cut* rules, respectively, are admissible). We interpret types as sets with some additional structure (Chapter 7), but considering only the sets, we prove that a

$\boxed{\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)}$ Under Θ, measurement list $\mathscr{M}'(F')$ covers $\mathscr{M}(F)$ (all inputs)

$$\frac{\cdot \vdash \underline{\mathsf{tt}}^{(F')}; F' \leq_{\mathbb{B}} \underline{\mathsf{tt}}^{(F)}; F}{\Theta \vdash \mathscr{M}'(F') \geq \cdot_F}$$

$$\frac{\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F) \qquad (\mathsf{fold}_{F'}\ \alpha')\ \nu\, t' =_\tau t' \in \mathscr{M}'(F') \\ \cdot \vdash \alpha'; F' \leq_\tau \alpha; F \qquad \overset{\mathsf{d}}{\div}\Theta; [\tau] \vdash t' \equiv t : \kappa \qquad \overset{\mathsf{d}}{\div}\Theta \vdash t' = t\ \mathsf{true}}{\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\ \nu\, t =_\tau t}$$

$\boxed{\Theta \vdash A \leq^{\pm} B}$ Under input Θ, input type A is a subtype of input B

$$\frac{}{\Theta \vdash 1 \leq^+ 1}\ \leq^+\!1 \qquad \frac{}{\Theta \vdash 0 \leq^+ 0}\ \leq^+\!0$$

$$\frac{\Theta \vdash R_1 \leq^+ R_1' \qquad \Theta \vdash R_2 \leq^+ R_2'}{\Theta \vdash R_1 \times R_2 \leq^+ R_1' \times R_2'}\ \leq^+\!\times \qquad \frac{\Theta \vdash P_1 \leq^+ P_1' \qquad \Theta \vdash P_2 \leq^+ P_2'}{\Theta \vdash P_1 + P_2 \leq^+ P_1' + P_2'}\ \leq^+\!+$$

$$\frac{\Theta, \overrightarrow{\varphi} \vdash R \leq^+ P}{\Theta \vdash R \wedge \overrightarrow{\varphi} \leq^+ P}\ \leq^+\!\wedge\mathrm{L} \qquad \frac{\Theta, {}^{\mathsf{d}}\Xi \vdash Q \leq^+ P}{\Theta \vdash \exists^{\mathsf{d}}\Xi.\, Q \leq^+ P}\ \leq^+\!\exists\mathrm{L}$$

$$\frac{\Theta \vdash R \leq^+ R' \qquad \Theta \vdash \overrightarrow{\varphi}\ \mathsf{true}}{\Theta \vdash R \leq^+ R' \wedge \overrightarrow{\varphi}}\ \leq^+\!\wedge\mathrm{R} \qquad \frac{\overset{\mathsf{d}}{\div}\Theta \vdash \sigma : {}^{\mathsf{d}}\Xi \qquad \Theta \vdash R \leq^+ [\sigma]Q}{\Theta \vdash R \leq^+ \exists^{\mathsf{d}}\Xi.\, Q}\ \leq^+\!\exists\mathrm{R}$$

$$\frac{\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)}{\Theta \vdash \left\{\nu : \mu F' \;\middle|\; \mathscr{M}'(F')\right\} \leq^+ \{\nu : \mu F \mid \mathscr{M}(F)\}}\ \leq^+\!\mu$$

$$\frac{\Theta \vdash N \leq^- N'}{\Theta \vdash \downarrow N \leq^+ \downarrow N'}\ \leq^+\!\downarrow \qquad \frac{\Theta \vdash P \leq^+ P'}{\Theta \vdash \uparrow P \leq^- \uparrow P'}\ \leq^-\!\uparrow$$

$$\frac{\Theta \vdash L' \leq^- L \qquad \Theta \vdash \overrightarrow{\varphi}\ \mathsf{true}}{\Theta \vdash \overrightarrow{\varphi} \supset L' \leq^- L}\ \leq^-\!\supset\mathrm{L} \qquad \frac{\overset{\mathsf{d}}{\div}\Theta \vdash \sigma : {}^{\mathsf{d}}\Xi \qquad \Theta \vdash [\sigma]M \leq^- L}{\Theta \vdash \forall^{\mathsf{d}}\Xi.\, M \leq^- L}\ \leq^-\!\forall\mathrm{L}$$

$$\frac{\Theta, \overrightarrow{\varphi} \vdash N \leq^- L}{\Theta \vdash N \leq^- \overrightarrow{\varphi} \supset L}\ \leq^-\!\supset\mathrm{R} \qquad \frac{\Theta, {}^{\mathsf{d}}\Xi \vdash N \leq^- M}{\Theta \vdash N \leq^- \forall^{\mathsf{d}}\Xi.\, M}\ \leq^-\!\forall\mathrm{R}$$

$$\frac{\Theta \vdash R' \leq^+ R \qquad \Theta \vdash L \leq^- L'}{\Theta \vdash R \to L \leq^- R' \to L'}\ \leq^-\!\to$$

Figure 6.13: Declarative subtyping

subtype denotes a subset of the set denoted by any of its supertypes. That is, membership of a (semantic) value in the subtype *implies* its membership in any supertype of the subtype. We may also view subtyping as implication.

Consider the perfectly reasonable (but ungrammatical because the product types do not have simple components) question of judgment $a : \mathbb{N} \vdash 1 \times (1 \wedge a = 3) \leq^{+} (1 \wedge a \geq 3) \times 1$: (ignoring the violation of grammar) subtyping for the first component requires verifying $a \geq 3$, which is impossible under no further logical assumptions. But from a logical perspective, $1 \times (1 \wedge a = 3)$ implies $a \geq 3$. The idea is that, for a type in an assumptive position (that is, a positive subtype or negative supertype), it does not matter which product component (products are viewed conjunctively) or function argument (in our system, functions must be fully applied to values) to which index data is attached. Economou et al. [2023] made room for this kind of judgment via the obviated extraction judgment, but now we require types to be simple wherever they need to be, via the type grammar and subtyping rules. In this example, the product types are ungrammatical because $1 \wedge a = 3$ and $1 \wedge a \geq 3$ are not simple (that is, they are not of form R; rather, they have form $Q \wedge \varphi$) and grammatically the factors of a product type must be simple.

According to our delay principle (mentioned in Ch. 3), in order to prevent backtracking, we should delay index verification until as much information as possible has been collected soundly. This is why we require the subtype of $\leq^{+}\exists$R and $\leq^{+}\wedge$R (and, dually, the supertype of $\leq^{-}\forall$L and $\leq^{-}\supset$L) to be simple: that is, rules $\leq^{+}\exists$L and $\leq^{+}\wedge$L (and, dually, $\leq^{-}\forall$R and $\leq^{-}\supset$R) must have already been applied (if possible).

Rule $\leq^{+}\wedge$R and its dual rule $\leq^{-}\supset$L verify the validity of the attached propositions. The notation $\Theta \vdash \overrightarrow{\varphi}\ \mathsf{true}$ means $\Theta \vdash \varphi_k\ \mathsf{true}$ for all $\varphi_k \in \overrightarrow{\varphi}$. In rule $\leq^{+}\exists$R and its dual rule $\leq^{-}\forall$L, we assume that we can conjure suitable index terms, the codomain of substitution σ

(the terms being substituted); in practice (that is, algorithmically), we will have to introduce existential variables, and then solve them.

For the unit type and the void type, rules $\leq^{+}1$ and $\leq^{+}0$ are simply reflexivity. Product subtyping $\leq^{+}\times$ is covariant subtyping of component types: a product type is a subtype of another if each component of the former is a subtype of the respective component of the latter. We have covariant shift rules $\leq^{+}\downarrow$ and $\leq^{-}\uparrow$. Function subtyping $\leq^{-}\rightarrow$ is standard: contravariant (in moving from conclusion to premise, the subtyping direction flips) in the function type's domain and covariant (in moving from conclusion to premise, the subtyping direction does not change) in the function type's codomain.

Economou et al. [2023] used judgmental equivalence rather than subtyping premises in rule $\leq^{+}+$, but that is unnecessary. Rule $\leq^{+}+$ is simply covariant subtyping of the respective summands, which may or may not be simple. While this is an improvement, the rule is still somewhat restrictive as written: for example, $(1 \wedge \mathsf{ff}) + (1 \wedge \mathsf{ff})$ logically implies $(1+1) \wedge \mathsf{ff}$ but the former is not a subtype of the latter. (It may be possible to increase the power of $\leq^{+}+$ by extracting the strongest consequence of the subtype, but this is potential future work.) Regardless, we don't expect this to be very restrictive in practice because programmers tend not to work with sum types themselves, but rather algebraic inductive types (like μF), and don't need to directly compare, via subtyping, (the unrolling of) different such types (such as the type of lists and the type of natural numbers). Further, unrolling never outputs a sum type with (a positive, nonzero number of) outer assertions like in $(1+1) \wedge \mathsf{ff}$.

Rule $\leq^{+}\mu$ says $\{v : \mu F' \mid \mathscr{M}'(F')\}$ is a subtype of $\{v : \mu F \mid \mathscr{M}(F)\}$ if $\mathscr{M}'(F')$ covers $\mathscr{M}(F)$: for each measurement $(\mathsf{fold}_F\ \alpha)\, v\, \boldsymbol{t} =_\tau t$ in $\mathscr{M}(F)$, there exists a measurement $(\mathsf{fold}_{F'}\ \alpha')\, v\, \boldsymbol{t}' =_\tau t'$ in $\mathscr{M}'(F')$ such that α' is a *submeasure* of α, that is, $\cdot \vdash \alpha'; F' \leq_\tau \alpha; F$, and the spines the measures are being applied to are SMT-equivalent, that is, $\stackrel{\mathsf{d}}{\div}\Theta; [\tau] \vdash \boldsymbol{t}' \equiv$

$t : \kappa$, and the results are SMT-equivalent, that is, $\overset{\mathsf{d}}{\div}\Theta \vdash t' = t$ true.

$\boxed{\Xi \vdash \alpha; F \leq_\tau \beta; G}$ Under Ξ, algebra $\alpha : F(\tau) \Rightarrow \tau$ is a submeasure of $\beta : G(\tau) \Rightarrow \tau$ (inputs: $\Xi, \alpha, F, \tau, \beta, G$)

$$\frac{\begin{array}{lll}\alpha \circ \mathsf{inj}_1 \doteq \alpha_1 & \beta \circ \mathsf{inj}_1 \doteq \beta_1 & \Xi \vdash \alpha_1; F_1 \leq_\tau \beta_1; G_1 \\ \alpha \circ \mathsf{inj}_2 \doteq \alpha_2 & \beta \circ \mathsf{inj}_2 \doteq \beta_2 & \Xi \vdash \alpha_2; F_2 \leq_\tau \beta_2; G_2\end{array}}{\Xi \vdash \alpha; F_1 \oplus F_2 \leq_\tau \beta; G_1 \oplus G_2}\ \text{Meas}{\leq}\oplus$$

$$\frac{\Xi, {}^{\mathsf{d}}\Xi' \vdash (\top, q) \Rightarrow t; \underline{Q} \otimes \hat{P} \leq_\tau (o', q') \Rightarrow t'; \underline{P} \otimes \hat{P}'}{\Xi \vdash (\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q) \Rightarrow t; \underline{\exists^{\mathsf{d}}\Xi'.\, Q} \otimes \hat{P} \leq_\tau (o', q') \Rightarrow t'; \underline{P} \otimes \hat{P}'}\ \text{Meas}{\leq}\exists\text{L}$$

$$\frac{\overset{\mathsf{d}}{\div}\Xi \vdash \sigma : {}^{\mathsf{d}}\Xi' \qquad \Xi \vdash (\top, q) \Rightarrow t; \underline{Q} \otimes \hat{P} \leq_\tau (\top, q') \Rightarrow [\sigma]t'; \underline{[\sigma]Q'} \otimes \hat{P}'}{\Xi \vdash (\top, q) \Rightarrow t; \underline{Q} \otimes \hat{P} \leq_\tau (\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q') \Rightarrow t'; \underline{\exists^{\mathsf{d}}\Xi'.\, Q'} \otimes \hat{P}'}\ \text{Meas}{\leq}\exists\text{R}$$

$$\frac{\Xi \vdash Q \leq^+ Q' \qquad \Xi \vdash q \Rightarrow t; \hat{P} \leq_\tau q' \Rightarrow t'; \hat{P}'}{\Xi \vdash (\top, q) \Rightarrow t; \underline{Q} \otimes \hat{P} \leq_\tau (\top, q') \Rightarrow t'; \underline{Q'} \otimes \hat{P}'}\ \text{Meas}{\leq}\text{Const}$$

$$\frac{\Xi, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id} \vdash q \Rightarrow t; \hat{I} \leq_\tau q' \Rightarrow t'; \hat{I}}{\Xi \vdash (a, q) \Rightarrow t; \mathrm{Id} \otimes \hat{I} \leq_\tau (a, q') \Rightarrow t'; \mathrm{Id} \otimes \hat{I}}\ \text{Meas}{\leq}\text{Id}$$

$$\frac{\overset{\mathsf{d}}{\div}\Xi \vdash u \equiv t : \tau}{\Xi \vdash () \Rightarrow u; I \leq_\tau () \Rightarrow t; I}\ \text{Meas}{\leq}I$$

Figure 6.14: Declarative submeasuring

Figure 6.14 defines the declarative submeasuring used by the $\leq^+\mu$ rule in subtyping. In this definition of submeasuring, the functors must be structurally equivalent, but we allow subtyping at constant functors; and parts of algebra bodies with free Id variables must be structurally equivalent, but we allow SMT equivalence for the parts of the bodies without free Id variables (see Fig. 6.15). Because we can pack indices in the bodies of algebras and because we allow subtyping at constant functors, in rule Meas$\leq\exists$R we need to apply the substitution σ to the (super)body t' as well as the (super)type Q'. That's why we cannot

$\boxed{\Theta \vdash u \equiv t : \tau}$ Under Θ, index terms u and t are equivalent and have sort τ (inputs: Θ, u, t, τ)

$$\frac{\Theta - \mathrm{Id} \vdash u = t \text{ true}}{\Theta \vdash u \equiv t : \kappa} \qquad \frac{(a : \tau) \in \Theta}{\Theta \vdash a \equiv a : \tau}$$

$$\frac{\kappa \in \{\mathbb{N}, \mathbb{Z}\} \qquad FV(u_1,u_2,t_1,t_2) \nsubseteq dom(\Theta - \mathrm{Id}) \qquad \Theta \vdash u_1 \equiv t_1 : \kappa \qquad \Theta \vdash u_2 \equiv t_2 : \kappa}{\begin{array}{c}\Theta \vdash u_1 + u_2 \equiv t_1 + t_2 : \kappa \\ \Theta \vdash u_1 - u_2 \equiv t_1 - t_2 : \kappa\end{array}}$$

$$\frac{FV(u_1,u_2,t_1,t_2) \nsubseteq dom(\Theta - \mathrm{Id}) \text{ or } \tau_1 \times \tau_2 \neq \kappa \qquad \Theta \vdash u_1 \equiv t_1 : \tau_1 \qquad \Theta \vdash u_2 \equiv t_2 : \tau_2}{\Theta \vdash (u_1,u_2) \equiv (t_1,t_2) : \tau_1 \times \tau_2}$$

$$\frac{FV(u,t) \nsubseteq dom((\Theta, a \div \kappa) - \mathrm{Id}) \qquad \Theta, a \div \kappa \vdash u \equiv t : \tau}{\Theta \vdash \lambda a.u \equiv \lambda a.t : \kappa \Rightarrow \tau}$$

$$\frac{FV(a,\boldsymbol{t},\boldsymbol{t}') \nsubseteq dom(\Theta - \mathrm{Id}) \qquad (a \stackrel{\mathrm{d}}{\div} \tau) \in \Theta \qquad \Theta;[\tau] \vdash \boldsymbol{t} \equiv \boldsymbol{t}' : \kappa}{\Theta \vdash a(\boldsymbol{t}) \equiv a(\boldsymbol{t}') : \kappa}$$

$$\frac{\begin{array}{c}FV(t_1,t_2,t_1',t_2') \nsubseteq dom(\Theta - \mathrm{Id}) \\ \overline{\Theta} \vdash t_1 : \kappa \qquad \overline{\Theta} \vdash t_1' : \kappa \qquad \Theta \vdash t_1 \equiv t_1' : \kappa \qquad \Theta \vdash t_2 \equiv t_2' : \kappa\end{array}}{\Theta \vdash t_1 = t_2 \equiv t_1' = t_2' : \mathbb{B}}$$

$$\frac{FV(\varphi_1,\varphi_2,\psi_1,\psi_2) \nsubseteq dom(\Theta - \mathrm{Id}) \qquad \Theta \vdash \varphi_1 \equiv \psi_1 : \mathbb{B} \qquad \Theta \vdash \varphi_2 \equiv \psi_2 : \mathbb{B}}{\begin{array}{c}\Theta \vdash \varphi_1 \wedge \varphi_2 \equiv \psi_1 \wedge \psi_2 : \mathbb{B} \\ \Theta \vdash \varphi_1 \vee \varphi_2 \equiv \psi_1 \vee \psi_2 : \mathbb{B}\end{array}}$$

$$\frac{FV(\varphi,\psi) \nsubseteq dom(\Theta - \mathrm{Id}) \qquad \Theta \vdash \varphi \equiv \psi : \mathbb{B}}{\Theta \vdash \neg\varphi \equiv \neg\psi : \mathbb{B}}$$

$$\frac{FV(t_1,t_2,t_1',t_2') \nsubseteq dom(\Theta - \mathrm{Id}) \qquad \kappa \in \{\mathbb{N}, \mathbb{Z}\} \qquad \Theta \vdash t_1 \equiv t_1' : \kappa \qquad \Theta \vdash t_2 \equiv t_2' : \kappa}{\Theta \vdash t_1 \leq t_2 \equiv t_1' \leq t_2' : \mathbb{B}}$$

$\boxed{\Theta;[\tau] \vdash \boldsymbol{t} \equiv \boldsymbol{t}' : \kappa}$ Under Θ, index spines $\boldsymbol{t}$ and $\boldsymbol{t}'$ are equivalent and have sort τ returning κ (Inputs: $\Theta, \tau, \boldsymbol{t}, \boldsymbol{t}'$; outputs: κ)

$$\frac{}{\Theta;[\kappa] \vdash \cdot \equiv \cdot : \kappa} \qquad \frac{\Theta \vdash t_0 \equiv t_0' : \kappa_0 \qquad \Theta;[\tau] \vdash \boldsymbol{t} \equiv \boldsymbol{t}' : \kappa}{\Theta;[\kappa_0 \Rightarrow \tau] \vdash t_0, \boldsymbol{t} \equiv t_0', \boldsymbol{t}' : \kappa}$$

$$\frac{k \in \{1,2\} \qquad \Theta;[\tau_k] \vdash \boldsymbol{t} \equiv \boldsymbol{t}' : \kappa}{\Theta;[\tau_1 \times \tau_2] \vdash .k, \boldsymbol{t} \equiv .k, \boldsymbol{t}' : \kappa}$$

Figure 6.15: Declarative index equivalence

simply have the proposed (incorrect) rule

$$\frac{\Xi \vdash P \leq^{+} P' \qquad \Xi \vdash q \Rightarrow t; \hat{P} \leq_{\tau} q' \Rightarrow t'; \hat{P}'}{\Xi \vdash (\top, q) \Rightarrow t; \underline{P} \otimes \hat{P} \leq_{\tau} (\top, q') \Rightarrow t'; \underline{P'} \otimes \hat{P}'}$$

replacing the rules Meas$\leq\exists$L, Meas$\leq\exists$R, and Meas$\leq$Const (as we wouldn't have access to the substitution we need to apply to t'). We slightly disobey the delay principle (see Ch. 3) in rule Meas$\leq\exists$R in that Q might not be simple, but these indices in σ are conjured from nowhere anyway in the declarative system such that the subtyping in Meas$\leq$Const holds. We completely uphold the delay principle in the algorithmic system. Rule Meas$\leq\mathrm{Id}$ is paired with an implicit rule

$$\frac{\Xi \vdash (a, q) \Rightarrow t; \mathrm{Id} \otimes \hat{I} \leq_{\tau} (a, q') \Rightarrow [a/a']t'; \mathrm{Id} \otimes \hat{I}}{\Xi \vdash (a, q) \Rightarrow t; \mathrm{Id} \otimes \hat{I} \leq_{\tau} (a', q') \Rightarrow t'; \mathrm{Id} \otimes \hat{I}}$$

Rule Meas$\leq\mathrm{Id}$ puts an Id variable hypothesis in the context, which is used by index equivalence in rule Meas$\leq I$ in order to simplify the interaction between submeasuring and unrolling when it comes to proving subsumption admissibility, discussed in Sec. 6.4. Index

equivalence (Fig. 6.15) uses the operation $- - \mathrm{Id}$ that subtracts Id variables from well-formed logical contexts, defined as follows.

$$\cdot - \mathrm{Id} = \cdot$$

$$(\Theta, a \div \tau) - \mathrm{Id} = (\Theta - \mathrm{Id}), a \div \tau$$

$$(\Theta, a \overset{\mathsf{d}}{\div} \kappa) - \mathrm{Id} = (\Theta - \mathrm{Id}), a \overset{\mathsf{d}}{\div} \kappa$$

$$(\Theta, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id}) - \mathrm{Id} = \Theta - \mathrm{Id}$$

$$(\Theta, \varphi) - \mathrm{Id} = (\Theta - \mathrm{Id}), \varphi$$

Anyway, this kind of submeasuring allows us to express that, for example, the type of lists of nonzero natural numbers is a subtype of the type of lists of natural numbers (and more). I anticipate this should also help with polymorphism and abstract refinements, potential extensions of this thesis in future.

In the appendix, we prove that subtyping is reflexive (Lemma C.62) and transitive (Lemma C.63). This relies on the reflexivity and transitivity of index equivalence among other basic properties.

6.4 Unrolling

Given $a : \mathbb{N}$, in our system, the type $\mathsf{List}\ P\ a$ of a-length lists of elements of type P is defined as $\{v : \mu \mathsf{ListF}_P \mid (\mathsf{fold}_{\mathsf{ListF}_P}\ \mathsf{lenalg})\, v = a\}$ where $\mathsf{ListF}_P = I \oplus (\underline{P} \otimes \mathrm{Id} \otimes I)$ and $\mathsf{lenalg} = \mathrm{inj}_1\ () \Rightarrow 0 \mid \mathrm{inj}_2\,(\top, (b, ())) \Rightarrow 1 + b$. Assuming we have $\mathsf{suc} : \forall a : \mathbb{N}.\, \mathsf{Nat}(a) \to \uparrow \mathsf{Nat}(1 + a)$ for incrementing a (program-level) natural number by one, we define length

in our system as follows:

$$
\begin{array}{l}
\mathtt{rec}\ \mathsf{length} : (\forall a : \mathbb{N}.\, \mathsf{List}(P)(a) \to \uparrow \mathsf{Nat}(a)).\ \lambda x.\ \mathtt{match}\ x.\ \{ \\
\quad \mathtt{into}(x') \Rightarrow \mathtt{match}\ x'\ \{ \\
\qquad \mathsf{inj}_1\, \langle\rangle \Rightarrow \qquad\qquad\qquad\qquad \text{-- } a = 0 \\
\qquad\quad \mathtt{return}\, \mathtt{into}(\mathsf{inj}_1\, \langle\rangle) \\
\quad \mid\ \mathsf{inj}_2\, \langle _, \langle y, \langle\rangle\rangle\rangle \Rightarrow \qquad\quad \text{-- } a = 1 + a' \text{ such that } a' \text{ is the length of } y \\
\qquad\quad \mathsf{let}\ z' = \mathsf{length}(y); \\
\qquad\quad \mathsf{let}\ z = \mathsf{suc}(z'); \\
\qquad\quad \mathtt{return}\, z \\
\quad \} \\
\}
\end{array}
$$

Checking length against its type annotation, the lambda rule assumes $x : \mathsf{List}(P)(a)$ for an arbitrary $a : \mathbb{N}$. Upon matching x against the pattern $\mathtt{into}(x')$, we know x' should have the *unrolled* type of $\mathsf{List}(P)(a)$. Ignoring refinements, we know that the erasure of this unrolling should be a sum type where the left component represents the empty list and the right component represents a head element together with a tail list. However, in order to verify the refinement that length does what we intend, we need to know more about the length index associated with x—that is, a—in the case where x is nil and in the case where x is a cons cell. Namely, the unrolling of $\mathsf{List}(P)(a)$ should know that $a = 0$ when x is the empty list, and that $a = 1 + a'$ where a' is the length of the tail of x when x is a nonempty list. This is the role of the unrolling judgment, to output just what we need here (left of $\stackrel{\circ}{=}$

are inputs; right of $\mathrel{\dot{=}}$ are outputs):

$$\cdot \vdash \{v : \mathsf{ListF}_P[\mu\mathsf{ListF}_P] \mid \mathsf{lenalg}\,(\mathsf{ListF}_P\ (\mathsf{fold}_{\mathsf{ListF}_P}\ \mathsf{lenalg})\ v) =_{\mathbb{N}} a\}$$
$$\mathrel{\dot{=}} \underbrace{(1 \wedge a = 0) + \left(P \times (\exists a' : \mathbb{N}.\, \left\{v : \mu\mathsf{ListF}_P \;\middle|\; (\mathsf{fold}_{\mathsf{ListF}_P}\ \mathsf{lenalg})\ v =_{\mathbb{N}} a'\right\} \times (1 \wedge a = 1 + a'))\right)}_{R}$$

That is, the type of P-lists of length a unrolls to either the unit type 1 (representing the empty list) together with the fact that a is 0, or the product of P (the type of the head element) and P-lists (representing the tail) of length a' such that a' is a minus one. The above is actually syntactic sugar for the unrolling judgment we will present:

$$\cdot \vdash \wr \mathsf{lenalg}; \mathsf{ListF}_P; (\mathsf{fold}_{\mathsf{ListF}_P}\ \mathsf{lenalg})\ v\ \cdot =_{\mathbb{N}} a \wr \mathrel{\dot{=}} \cdot; R$$

Refined inductive type unrolling $\Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \wr \mathrel{\dot{=}} {}^{\mathsf{d}}\Theta; R$, inspired by work in fibrational dependent type theory [Atkey et al., 2012], is defined in Fig. 6.16. The functor G is called the *principal* functor and $\overrightarrow{\beta}$ the *principal* measures of the unrolling judgment; we speak of "G-unrolling" (…outputs ${}^{\mathsf{d}}\Theta$ and R, say). The judgment $\Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \wr \mathrel{\dot{=}} {}^{\mathsf{d}}\Theta; R$ presupposes that $\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi]$ and that

$$\Xi \vdash \beta_k : G(\tau_k) \Rightarrow \tau_k$$

for all $(\beta_k, (\mathsf{fold}_F\ _)\ v\ _ =_{\tau_k} _) \in zip(\overrightarrow{\beta})(\mathscr{M}(F))$. As in the list example above, unrolling is always initiated with $\Xi = \cdot$ and $G = F$ and $\mathscr{M}(F) \rightsquigarrow \overrightarrow{\beta}$ where $\rightsquigarrow$ on measurements simply extracts the algebras in the same order: here, $\rightsquigarrow$ is defined by $\cdot_F \rightsquigarrow \cdot$
and $\mathscr{M}(F), (\mathsf{fold}_F\ \alpha_k)\ v\ t_k =_{\tau_k} t_k \rightsquigarrow \overrightarrow{\alpha}, \alpha_k$ if $\mathscr{M}(F) \rightsquigarrow \overrightarrow{\alpha}$.

$\boxed{\Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Theta; R}$ The "$\overrightarrow{\beta}; G$" part of unrolling $\{\nu : \mu F \mid \mathscr{M}(F)\}$ (ins.: $\Xi, \overrightarrow{\beta}, G, \mathscr{M}(F)$) corresponds to the (from outputs ${}^{\mathsf{d}}\Theta$ and R) type $\exists^{\mathsf{d}}\Theta.\,(R \wedge {}^{\mathsf{d}}\Theta)$

$$\frac{\begin{array}{cc}\overrightarrow{\beta} \circ \mathsf{inj}_1 \doteq \overrightarrow{\beta_1} & \Xi \vdash \wr \overrightarrow{\beta_1}; G_1; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Theta_1; R_1 \\ \overrightarrow{\beta} \circ \mathsf{inj}_2 \doteq \overrightarrow{\beta_2} & \Xi \vdash \wr \overrightarrow{\beta_2}; G_2; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Theta_2; R_2\end{array}}{\Xi \vdash \wr \overrightarrow{\beta}; G_1 \oplus G_2; \mathscr{M}(F) \wr \doteq \cdot; (\exists^{\mathsf{d}}\Theta_1.\,(R_1 \wedge {}^{\mathsf{d}}\Theta_1)) + (\exists^{\mathsf{d}}\Theta_2.\,(R_2 \wedge {}^{\mathsf{d}}\Theta_2))}\ \wr\oplus\wr$$

($^{\mathsf{d}}\Xi'$ may be $\cdot$ and $\overrightarrow{\varphi}$ may be $\cdot$)

$$\frac{\overrightarrow{\beta} \rightsquigarrow \overrightarrow{\beta'} \qquad \Xi, {}^{\mathsf{d}}\Xi' \vdash \wr \overrightarrow{\beta'}; \hat{P}; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Theta_0; R_0}{\Xi \vdash \wr \overrightarrow{\beta}; \underline{\exists^{\mathsf{d}}\Xi'.\,R' \wedge \overrightarrow{\varphi}} \otimes \hat{P}; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Xi', {}^{\mathsf{d}}\Theta_0, \overrightarrow{\varphi}; R' \times R_0}\ \wr\mathrm{Const}\wr$$

$$\frac{\begin{array}{c}\overrightarrow{a \overset{\mathsf{d}\cdot}{\div} \tau} = \overrightarrow{a} \overset{\mathsf{d}\cdot}{\div} \mathscr{M}(F) \\ \Xi, \overrightarrow{a \overset{\mathsf{d}\cdot}{\div} \tau}, a\,\mathrm{Id} \vdash \wr \overrightarrow{q \Rightarrow t'}; \hat{I}; \mathscr{M}(F) \wr \doteq \Xi'', \overrightarrow{\psi''}; R'' \\ \Xi; \Xi''; zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash \overrightarrow{\psi''} \rightsquigarrow \breve{\Xi}_1; \mathscr{M}_1(F); \overrightarrow{\psi'} \\ \Xi; \Xi''; zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash R'' \rightsquigarrow \breve{\Xi}_2; \mathscr{M}_2(F); R' \\ \breve{\Xi} = \breve{\Xi}_1 \cup \breve{\Xi}_2 \qquad \mathscr{M}'(F) = \mathscr{M}_1(F) \cup \mathscr{M}_2(F) \\ dom(\Xi') \cap dom(\Xi, \overrightarrow{a \overset{\mathsf{d}\cdot}{\div} \tau}, \Xi'', \breve{\Xi}) = \emptyset \qquad \rho = \Xi'/\breve{\Xi} \text{ is a variable renaming}\end{array}}{\Xi \vdash \wr \overrightarrow{(a,q) \Rightarrow t'}; \mathrm{Id} \otimes \hat{I}; \mathscr{M}(F) \wr \doteq \Xi', \Xi'', [\rho]\overrightarrow{\psi'}; \{\nu : \mu F \mid [\rho]\mathscr{M}'(F)\} \times [\rho]R'}\ \wr\mathrm{Id}\wr$$

$$\frac{\overrightarrow{t'} @ \mathscr{M}(F) \doteq \overrightarrow{\varphi}}{\Xi \vdash \wr \overrightarrow{() \Rightarrow t'}; I; \mathscr{M}(F) \wr \doteq \overrightarrow{\varphi}; 1}\ \wr I\wr$$

where

$$\frac{}{\cdot \circ \mathsf{inj}_k \doteq \cdot} \qquad \frac{\overrightarrow{\beta} \circ \mathsf{inj}_k \doteq \overrightarrow{\beta'} \qquad \beta_n \circ \mathsf{inj}_k \doteq \beta_{nk}}{\overrightarrow{\beta}, \beta_n \circ \mathsf{inj}_k \doteq \overrightarrow{\beta'}, \beta_{nk}}$$

$$\frac{}{\cdot \rightsquigarrow \cdot} \qquad \frac{\overrightarrow{\beta} \rightsquigarrow \overrightarrow{\beta'}}{\overrightarrow{\beta}, (\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q) \Rightarrow t' \rightsquigarrow \overrightarrow{\beta'}, q \Rightarrow t' \qquad \overrightarrow{\beta}, (\top, q) \Rightarrow t' \rightsquigarrow \overrightarrow{\beta'}, q \Rightarrow t'}$$

$$\frac{}{\cdot @ \cdot \doteq \cdot} \qquad \frac{\overrightarrow{u} @ \mathscr{M}(F) \doteq \overrightarrow{\varphi}}{(\overrightarrow{u}, t') @ (\mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\, \nu\, \boldsymbol{t} =_\tau t) \doteq \overrightarrow{\varphi}, (t = \langle t' \mid \boldsymbol{t} \rangle)}$$

$$\cdot \overset{\mathsf{d}\cdot}{\div} \cdot_F = \cdot$$

$$(\overrightarrow{a}, a_k) \overset{\mathsf{d}\cdot}{\div} (\mathscr{M}(F), (\mathsf{fold}_F\ \alpha_k)\, \nu\, \boldsymbol{t}_k =_{\tau_k} t_k) = (\overrightarrow{a} \overset{\mathsf{d}\cdot}{\div} \mathscr{M}(F)), a_k \overset{\mathsf{d}\cdot}{\div} \tau_k$$

Figure 6.16: Unrolling

We define the following metaoperations to form types with logical contexts.

$$\exists \cdot . P = P$$
$$\exists (\Theta, a \div \kappa). P = \exists \Theta . P$$
$$\exists (\Theta, a \overset{\mathsf{d}}{\div} \kappa). P = \exists \Theta . (\exists a \overset{\mathsf{d}}{\div} \kappa . P)$$
$$\exists (\Theta, \varphi). P = \exists \Theta . P$$

$$Q \wedge \cdot = Q$$
$$Q \wedge (\Theta, a : \kappa) = Q \wedge \Theta$$
$$Q \wedge (\Theta, \varphi) = (Q \wedge \varphi) \wedge \Theta$$

$$\forall \cdot . N = N$$
$$\forall (\Theta, a \div \kappa). N = \forall \Theta . N$$
$$\forall (\Theta, a \overset{\mathsf{d}}{\div} \kappa). N = \forall \Theta . (\forall a \overset{\mathsf{d}}{\div} \kappa . N)$$
$$\forall (\Theta, \varphi). N = \forall \Theta . N$$

$$\cdot \supset M = M$$
$$(\Theta, a : \kappa) \supset M = \Theta \supset M$$
$$(\Theta, \varphi) \supset M = \Theta \supset (\varphi \supset M)$$

$\wr \oplus \int$ unrolls each branch and then sums the types formed (using the above metaoperations) from the outputs. $\wr \mathrm{Const} \int$ collects (in the logical context output together with the rest of the unrolling context output) the indices of the constant functor, and outputs the product of the underlying simple type R' and the rest of the simple unrolling output type R_0. $\wr \mathrm{Id} \int$ outputs the product of the original inductive type but with measurements given by the recursive results Ξ' of the measures (over which we will existentially quantify), together with the rest of the unrolling. The recursive results are calculated by a kind of hereditary substitution on $\hat{I}$-unrolling outputs defined in Fig. 6.17, which we often call the liftapps judgment (as in lifting the applications of Id variables). In particular, the judgment

$$\Xi; \Xi'; \overrightarrow{(a, (\!|\alpha|\!)_F \, \nu _ =_\tau _)} \vdash \mathscr{O} \rightsquigarrow \check{\Xi}; \mathscr{M}'(F); \mathscr{O}'$$

replaces in $\mathscr{O}$ (an output of $\hat{I}$ unrolling) each application of an Id variable $a_k \in \overrightarrow{a}$ to a spine $\boldsymbol{u}$ with a fresh variable which is basically the result of the recursive fold applied to the spine

$\boldsymbol{u}$. We say "basically" because the same Id variable a_k can be used in $\boldsymbol{u}$[11], and these also get replaced in the same way, so the fresh variable is the result of the recursive fold applied to the output $\boldsymbol{u}'$ of the same judgment with input $\boldsymbol{u}$. We keep track of which applications were already lifted by a superscript on special variables $\breve{a}^{a(\boldsymbol{u})}$ and then take the union of the output contexts and measurements with these special variables to avoid redundancy. In $\wr\mathrm{Id}\wr$, we then discharge these special variables by renaming them (ρ) to fresh index variables (normal ones). $\wr I\wr$ outputs the unit type together with the index term equalities given by the (unrolled) measurements. We have set up these rules so that the output logical context ${}^{\mathsf{d}}\Theta$ of $\hat{P}$-unrolling can always be partitioned like so:

$$\underbrace{{}^{\mathsf{d}}\Xi}_{\text{result of Id var. apps.}}\ ,\ \underbrace{{}^{\mathsf{d}}\Xi'}_{\text{existentials of constant functors}}\ ,\ \underbrace{\overrightarrow{\psi}}_{\wr I\wr\text{ ctx. output}}\ ,\ \underbrace{\overrightarrow{\varphi}}_{\text{assertions of constant functors}}$$

If our functor and algebra grammars were instead more direct, like those implicitly used in the introduction (Chapter 1) and background (Chapter 3), and explicitly discussed in Chapter 5, then we would have to modify the unrolling judgment, and it would need more rules. We expect everything would still work, but we prefer having to consider fewer rules when proving metatheory.

We prove that we can form (well-formed) types from refined unrolling outputs. The proof is tedious, largely uninteresting, and requires proving auxilliary judgments transform ξ nicely: Lemma C.54 (liftapps WF). In future work it would be a good idea to simplify the liftapps judgment and related metatheory if possible. This is also intrinsically challenging to simplify because there is much freedom in how one can define an algebra.

Lemma 6.5 (Unrolling Output WF)**.** *(Lemma C.55)*

[11] Due to modularity, Id variables other than a_k (from independent measures) cannot be used in $\boldsymbol{u}$, hence why the main premise of the third rule singles out the a_k entry.

$\boxed{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \mathscr{O} \rightsquigarrow \check{\Xi};\mathscr{M}'(F);\mathscr{O}'}$

Presupposes $F_k = F$ and $FV(F_k,\alpha_k)=\emptyset$ for all $(a_k,(\!|\alpha_k|\!)_{F_k}\,\nu_=_{\tau_k}\,_) \in \overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)}$; and $\mathscr{O}$ is well-formed under $\Xi,\Xi',a\overset{\mathrm{d}\cdot}{\div}\tau,a\,\mathrm{Id}$
(Inputs: left of $\rightsquigarrow$; outputs: right of $\rightsquigarrow$)

$$\frac{a_k \notin FV(t) \text{ for all } (a_k,_) \in \overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)}}{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash t \rightsquigarrow \cdot;\cdot_F;t}$$

$$\frac{(a_k,(\!|\alpha_k|\!)_F\,\nu_=_{\tau_k}\,_) \in \overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)}}{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash a_k \rightsquigarrow \check{a}_k^{a_k(\cdot)} \overset{\mathrm{d}\cdot}{\div} \tau_k;(\!|\alpha_k|\!)_F\,\nu\cdot=_{\tau_k}\check{a}_k^{a_k(\cdot)};\check{a}_k^{a_k(\cdot)}}$$

$$\frac{\begin{array}{c}(a_k,(\!|\alpha_k|\!)_F\,\nu_=_{\tau_k}\,_) \in \overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \\ \Xi;\Xi';(a_k,(\!|\alpha_k|\!)_F\,\nu_=_{\tau_k}\,_) \vdash \boldsymbol{u} \rightsquigarrow \check{\Xi};\mathscr{M}'(F);\boldsymbol{u}' \qquad \overset{\mathrm{d}\cdot}{\div}(\Xi,\Xi',\check{\Xi});[\tau_k] \vdash \boldsymbol{u}' : \kappa\end{array}}{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash a_k(\boldsymbol{u}) \rightsquigarrow \check{\Xi},\check{a}_k^{a_k(\boldsymbol{u}')} \overset{\mathrm{d}\cdot}{\div} \kappa;\mathscr{M}',(\!|\alpha_k|\!)_F\,\nu\,\boldsymbol{u}'=_{\tau_k}\check{a}_k^{a_k(\boldsymbol{u}')};\check{a}_k^{a_k(\boldsymbol{u}')}}$$

$$\frac{\begin{array}{c}a_k \neq b \text{ for all } (a_k,_) \in \overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \\ \Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \boldsymbol{u} \rightsquigarrow \check{\Xi};\mathscr{M}';\boldsymbol{u}'\end{array}}{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash b(\boldsymbol{u}) \rightsquigarrow \check{\Xi};\mathscr{M}';b(\boldsymbol{u}')}$$

$$\frac{op \in \{\neg_\} \qquad \Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash t \rightsquigarrow \check{\Xi};\mathscr{M}';t'}{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash op\,t \rightsquigarrow \check{\Xi};\mathscr{M}';op\,t'}$$

$$\frac{\begin{array}{c}op \in \{-+-,---,-=-,-\leq-,-\wedge-,-\vee-,(-,-)\} \\ \Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash t_1 \rightsquigarrow \check{\Xi}_1;\mathscr{M}_1;t_1' \\ \Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash t_2 \rightsquigarrow \check{\Xi}_2;\mathscr{M}_2;t_2'\end{array}}{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash t_1\,op\,t_2 \rightsquigarrow \check{\Xi}_1\cup\check{\Xi}_2;\mathscr{M}_1\cup\mathscr{M}_2;t_1'\,op\,t_2'}$$

$$\overline{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \cdot \rightsquigarrow \cdot;\cdot_F;\cdot}$$

$$\frac{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash t \rightsquigarrow \check{\Xi};\mathscr{M};t' \qquad \Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \boldsymbol{t} \rightsquigarrow \check{\Xi}';\mathscr{M}';\boldsymbol{t}'}{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash t,\boldsymbol{t} \rightsquigarrow \check{\Xi}\cup\check{\Xi}';\mathscr{M}\cup\mathscr{M}';t',\boldsymbol{t}'}$$

$$\frac{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \boldsymbol{t} \rightsquigarrow \check{\Xi};\mathscr{M}';\boldsymbol{t}'}{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash .k,\boldsymbol{t} \rightsquigarrow \check{\Xi};\mathscr{M}';.k,\boldsymbol{t}'}$$

$$\frac{\begin{array}{c}\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \mathscr{M}_0 \rightsquigarrow \check{\Xi};\mathscr{M};\mathscr{M}_0' \\ \Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash R \rightsquigarrow \check{\Xi}';\mathscr{M}';R'\end{array}}{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \{\nu:\mu F \mid \mathscr{M}_0\}\times R \rightsquigarrow \check{\Xi}\cup\check{\Xi}';\mathscr{M}\cup\mathscr{M}';\left\{\nu:\mu F \;\middle|\; \mathscr{M}_0'\right\}\times R'}$$

$$\overline{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash 1 \rightsquigarrow \cdot;\cdot_F;1} \qquad \overline{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \cdot_F \rightsquigarrow \cdot;\cdot_F;\cdot_F}$$

$$\frac{\begin{array}{c}\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \mathscr{M}_0 \rightsquigarrow \check{\Xi};\mathscr{M};\mathscr{M}_0' \\ \Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \boldsymbol{t} \rightsquigarrow \check{\Xi}';\mathscr{M}';\boldsymbol{t}'\end{array}}{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \mathscr{M}_0,(\mathsf{fold}_F\;\alpha)\,\nu\,\boldsymbol{t}=_\tau t \rightsquigarrow \check{\Xi}\cup\check{\Xi}';\mathscr{M}\cup\mathscr{M}';\mathscr{M}_0',(\mathsf{fold}_F\;\alpha)\,\nu\,\boldsymbol{t}'=_\tau t}$$

Figure 6.17: A judgment (also called "liftapps") used for $\wr\mathrm{Id}\int$

If $\Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta; R$ *and* $\Xi \vdash G\ \mathsf{functor}[\xi_G]$
then there exists ξ *such that* $\Xi \vdash \exists^{\mathsf{d}}\Theta.R \wedge {}^{\mathsf{d}}\Theta\ \mathsf{type}[\xi]$ *and* $\xi_G \subseteq \xi$.

Unrolling and subtyping There are four key lemmas for proving subsumption admissibility in the refined ADT cases (see Sec. 6.6).

First, unrolling a sublist of measurements outputs a supertype:

Lemma 6.6 (Unroll Sublist)**.** *(Lemma C.78)*
If $\Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta; R$ *and* $zip(\overrightarrow{\beta'})(\mathscr{M}'(F)) \subseteq zip(\overrightarrow{\beta})(\mathscr{M}(F))$
then $\Xi \vdash \wr \overrightarrow{\beta'}; G; \mathscr{M}'(F) \int \doteq {}^{\mathsf{d}}\Theta'; R'.$ *and* $\Xi \vdash \exists^{\mathsf{d}}\Theta.R \wedge {}^{\mathsf{d}}\Theta \leq^{+} \exists^{\mathsf{d}}\Theta'.R' \wedge {}^{\mathsf{d}}\Theta'$.

Second, unrolling a superlist of measurements outputs a subtype:

Lemma 6.7 (Unroll Superlist)**.** *(Lemma C.79)*
If $\Xi \vdash \wr \overrightarrow{\beta'}; G; \mathscr{M}'(F) \int \doteq {}^{\mathsf{d}}\Theta'; R'$ *and* $zip(\overrightarrow{\beta'})(\mathscr{M}'(F)) \subseteq zip(\overrightarrow{\beta})(\mathscr{M}(F))$
then $\Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta; R$ *and* $\Xi \vdash \exists^{\mathsf{d}}\Theta.R \wedge {}^{\mathsf{d}}\Theta \leq^{+} \exists^{\mathsf{d}}\Theta'.R' \wedge {}^{\mathsf{d}}\Theta'$.

These first two lemmas allow us to set up a situation involving the measurement covering judgment ${}^{\mathsf{d}}\Xi \vdash \mathscr{M}'(F)' \geq_{\equiv} \mathscr{M}(F)$ which simply means $\Theta \vdash \mathscr{M}'(F) \geq \mathscr{M}(F)$ and $\#\mathscr{M}' = \#\mathscr{M}$ and $\mathscr{M}'$ and $\mathscr{M}$ are in the same order:

$$\frac{\cdot \vdash \underline{\mathsf{tt}}^{(F')}; F' \equiv_{\mathbb{B}} \underline{\mathsf{tt}}^{(F)}; F}{\Theta \vdash \cdot_{F'} \geq_{\equiv} \cdot_{F}}$$

$$\frac{\begin{array}{c}\Theta \vdash \mathscr{M}'(F') \geq_{\equiv} \mathscr{M}(F) \\ \cdot \vdash \alpha'; F' \leq_{\tau} \alpha; F \qquad {}^{\mathsf{d}}\dot{\div}\Theta; [\tau] \vdash t' \equiv t : \kappa \qquad {}^{\mathsf{d}}\dot{\div}\Theta \vdash t' = t\ \mathsf{true}\end{array}}{\Theta \vdash \mathscr{M}'(F'), (\mathsf{fold}_{F'}\ \alpha')\ v\ t' =_{\tau} t' \geq_{\equiv} \mathscr{M}(F), (\mathsf{fold}_{F}\ \alpha)\ v\ t =_{\tau} t}$$

Third, unrolling supermeasures outputs a supertype:

Lemma 6.8 (Unroll to Supertype)**.** *(Lemma C.75)*

If $^{\mathsf{d}}\Xi, \Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta; R$

and $\Xi \vdash \overrightarrow{\beta}; G \leq_{\overrightarrow{\tau}} \overrightarrow{\beta'}; G'$ *and* $^{\mathsf{d}}\Xi \vdash \mathscr{M}(F) \geq_{\equiv} \mathscr{M}'(F')$

then there exist $^{\mathsf{d}}\Theta'$ *and* R' *such that* $^{\mathsf{d}}\Xi, \Xi \vdash \wr \overrightarrow{\beta'}; G'; \mathscr{M}'(F') \int \doteq {}^{\mathsf{d}}\Theta'; R'$

and $^{\mathsf{d}}\Xi, \Xi \vdash \exists^{\mathsf{d}}\Theta. R \wedge {}^{\mathsf{d}}\Theta \leq^{+} \exists^{\mathsf{d}}\Theta'. R' \wedge {}^{\mathsf{d}}\Theta'$.

Fourth and finally, unrolling submeasures outputs a subtype:

Lemma 6.9 (Unroll to Subtype)**.** *(Lemma C.77)*

If $^{\mathsf{d}}\Xi, \Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta; R$

and $\Xi \vdash \overrightarrow{\beta'}; G' \leq_{\overrightarrow{\tau}} \overrightarrow{\beta}; G$ *and* $^{\mathsf{d}}\Xi \vdash \mathscr{M}'(F') \geq_{\equiv} \mathscr{M}(F)$

then there exist $^{\mathsf{d}}\Theta'$ *and* R' *such that* $^{\mathsf{d}}\Xi, \Xi \vdash \wr \overrightarrow{\beta'}; G'; \mathscr{M}'(F') \int \doteq {}^{\mathsf{d}}\Theta'; R'$

and $^{\mathsf{d}}\Xi, \Xi \vdash \exists^{\mathsf{d}}\Theta'. R' \wedge {}^{\mathsf{d}}\Theta' \leq^{+} \exists^{\mathsf{d}}\Theta. R \wedge {}^{\mathsf{d}}\Theta$.

We state these last two lemmas in the appendix with more technical information to facilitate the proof. In particular, for Lemma 6.8 (Unroll to Supertype),

naming $^{\mathsf{d}}\Xi, \Xi \vdash \exists^{\mathsf{d}}\Theta. R \wedge {}^{\mathsf{d}}\Theta \leq^{+} \exists^{\mathsf{d}}\Theta'. R' \wedge {}^{\mathsf{d}}\Theta'$ by $\mathscr{D}$,

if $G = \hat{I}$

then $^{\mathsf{d}}\Xi, \frac{\mathsf{d}}{\div}\Xi, \overline{^{\mathsf{d}}\Theta} \vdash {}^{\mathsf{d}}\Theta - \overline{^{\mathsf{d}}\Theta} \equiv {}^{\mathsf{d}}\Theta' - \overline{^{\mathsf{d}}\Theta'} : \mathbb{B}$

and every subderivation $^{\mathsf{d}}\Xi, \frac{\mathsf{d}}{\div}\Xi, \overline{^{\mathsf{d}}\Theta} \vdash \mathscr{M}_1 \geq \mathscr{M}_1'$ that is a premise of $\leq^{+}\mu$ in $\mathscr{D}$

satisfies $^{\mathsf{d}}\Xi, \frac{\mathsf{d}}{\div}\Xi, \overline{^{\mathsf{d}}\Theta} \vdash \mathscr{M}_1 \geq_{\equiv} \mathscr{M}_1'$,

and the $\leq^{+}\exists$R-witness $^{\mathsf{d}}\Xi, \frac{\mathsf{d}}{\div}\Xi, \overline{^{\mathsf{d}}\Theta} \vdash \sigma : \overline{^{\mathsf{d}}\Theta'}$ of $\mathscr{D}$

is the identity substitution on $\overline{^{\mathsf{d}}\Theta'}$ and $dom({}^{\mathsf{d}}\Theta') = dom({}^{\mathsf{d}}\Theta)$. (Similarly for Lemma 6.9 (Unroll to Subtype).) These facts greatly simplify the proof, and are the entire point of Id variables and how they are used in submeasuring (in particular, submeasuring's use of index equivalence at leaves).

Here the logical context equivalence judgment (Fig. 6.18) is helpful as we can always swap out judgmentally equivalent logical contexts without changing the structure or height of a derivation. (See, for example, appendix Lemma C.71 (Ctx. Equiv. Compat).) We also use this judgment elsewhere in the metatheory (algorithmic completeness). Two logical contexts are judgmentally equivalent under Θ if they have exactly the same variable sortings (in the same list positions) and logically equivalent (under Θ) propositions, in the same order. The most interesting rule is the one for propositions, where, in the second premise, we filter out propositions from Θ_1 because we want each respective proposition to be logically equivalent under the propositions (and indexes) of Θ, but variables in Θ_1 (or Θ_2) may appear in φ_1 (or φ_2). (Note that it is equivalent to use $\overline{\Theta_2}$ rather than $\overline{\Theta_1}$ in the second premise of the last rule.)

$\boxed{\Theta \vdash \Theta_1 \equiv \Theta_2 \ \mathsf{ctx}}$ Under input Θ, input logical contexts Θ_1 and Θ_2 are equivalent

$$\frac{}{\Theta \vdash \cdot \equiv \cdot \ \mathsf{ctx}}\ \text{Ctx}{\equiv}\text{Empty} \qquad \frac{\Theta \vdash \Theta_1 \equiv \Theta_2 \ \mathsf{ctx}}{\begin{array}{l}\Theta \vdash \Theta_1, a \div \tau \equiv \Theta_2, a \div \tau \ \mathsf{ctx} \\ \Theta \vdash \Theta_1, a \overset{\mathsf{d}}{\div} \tau \equiv \Theta_2, a \overset{\mathsf{d}}{\div} \tau \ \mathsf{ctx}\end{array}}\ \text{Ctx}{\equiv}\text{Var}$$

$$\frac{\Theta \vdash \Theta_1 \equiv \Theta_2 \ \mathsf{ctx} \qquad \Theta, \overline{\Theta_1} \vdash \varphi_1 \equiv \varphi_2 : \mathbb{B}}{\Theta \vdash \Theta_1, \varphi_1 \equiv \Theta_2, \varphi_2 \ \mathsf{ctx}}\ \text{Ctx}{\equiv}\text{Prop}$$

Figure 6.18: Declarative logical context equivalence

6.5 Typing

Declarative bidirectional typing rules are given in Figs. 6.19, 6.20, and 6.21. By careful design, guided by logical principles, all typing rules are syntax-directed. That is, when deriving a conclusion, at most one rule is compatible with the syntax of the input program term and the principal input type. All declarative typing judgments with input contexts Θ

$\boxed{\Theta;\Gamma \vdash h \Rightarrow P}$ Under inputs Θ and Γ, input head h synthesizes (output) type P

$$\frac{(x : R) \in \Gamma}{\Theta;\Gamma \vdash x \Rightarrow R}\ \text{Decl}{\Rightarrow}\text{Var} \qquad \frac{\overline{\Theta} \vdash P\ \mathsf{type}[\xi_P] \qquad \Theta;\Gamma \vdash v \Leftarrow P}{\Theta;\Gamma \vdash (v : P) \Rightarrow P}\ \text{Decl}{\Rightarrow}\text{ValAnnot}$$

$\boxed{\Theta;\Gamma \vdash g \Rightarrow \uparrow P}$ Under inputs Θ and Γ, input bound expression g synthesizes (output) type $\uparrow P$

$$\frac{\Theta;\Gamma \vdash h \Rightarrow \downarrow N \qquad \Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P}{\Theta;\Gamma \vdash h(s) \Rightarrow \uparrow P}\ \text{Decl}{\Rightarrow}\text{App}$$

$$\frac{\overline{\Theta} \vdash P\ \mathsf{type}[\xi_P] \qquad \Theta;\Gamma \vdash e \Leftarrow \uparrow P}{\Theta;\Gamma \vdash (e : \uparrow P) \Rightarrow \uparrow P}\ \text{Decl}{\Rightarrow}\text{ExpAnnot}$$

Figure 6.19: Declarative head and bound expression type synthesis

and Γ presuppose Θ ctx and $\overline{\Theta} \vdash \Gamma$ ctx.

To manage the interaction between subtyping and program typing, types in a well-formed (under $\overline{\Theta}$) program context Γ must be simple. We maintain this invariant in program typing by extracting indices before adding any variable typings to the context. Extracting indices is made easy by the type grammar: any positive type can be written uniquely (up to alpha-equivalence) as $\exists^{\mathsf{d}}\Xi.R \wedge \overrightarrow{\varphi}$ (type R is simple); any negative type can be written uniquely (up to alpha-equivalence) as $\forall^{\mathsf{d}}\Xi.\,\overrightarrow{\varphi} \supset L$ (type L is simple).

The judgment $\Theta;\Gamma \vdash h \Rightarrow P$ (Fig. 6.19) synthesizes the type P from the head h. This judgment is synthesizing, because it is used in what are, from a Curry–Howard perspective, kinds of cut rules: Decl$\Rightarrow$App and Decl$\Leftarrow$`match`, discussed later. The synthesized type is the cut type, which does not appear in the conclusion of Decl$\Rightarrow$App or Decl$\Leftarrow$`match`. For head variables, we look up the variable's type in the context Γ (Decl$\Rightarrow$Var). For annotated values, we synthesize the annotation (Decl$\Rightarrow$ValAnnot).

$\boxed{\Theta;\Gamma \vdash v \Leftarrow P}$ Under inputs Θ and Γ, input value v checks against input type P

$$\frac{(x : R') \in \Gamma \qquad \Theta \vdash R' \leq^{+} R}{\Theta;\Gamma \vdash x \Leftarrow R}\ \text{Decl}\Leftarrow\text{Var} \qquad \frac{}{\Theta;\Gamma \vdash \langle\rangle \Leftarrow 1}\ \text{Decl}\Leftarrow 1$$

$$\frac{\Theta;\Gamma \vdash v_1 \Leftarrow R_1 \qquad \Theta;\Gamma \vdash v_2 \Leftarrow R_2}{\Theta;\Gamma \vdash \langle v_1, v_2\rangle \Leftarrow R_1 \times R_2}\ \text{Decl}\Leftarrow\times \qquad \frac{\Theta;\Gamma \vdash v \Leftarrow P_k}{\Theta;\Gamma \vdash \text{inj}_k\, v \Leftarrow P_1 + P_2}\ \text{Decl}\Leftarrow +_k$$

$$\frac{\overset{\mathsf{d}}{\div}\Theta \vdash \sigma : {}^{\mathsf{d}}\Xi \qquad \Theta;\Gamma \vdash v \Leftarrow [\sigma]Q}{\Theta;\Gamma \vdash v \Leftarrow (\exists^{\mathsf{d}}\Xi.\, Q)}\ \text{Decl}\Leftarrow\exists \qquad \frac{\Theta \vdash \overrightarrow{\varphi}\ \mathsf{true} \qquad \Theta;\Gamma \vdash v \Leftarrow R}{\Theta;\Gamma \vdash v \Leftarrow R \wedge \overrightarrow{\varphi}}\ \text{Decl}\Leftarrow\wedge$$

$$\frac{\begin{array}{c}\nexists x.\, v = \overrightarrow{\text{inj}_{k_i}}^{i}\left(\overrightarrow{\langle __j, - \rangle}^{j} x\right) \\ \overset{\mathsf{d}}{\div}\Theta \vdash \wr\, \overrightarrow{\alpha_i}^{i}; F; \mathscr{M}(F)\, \wr \doteq {}^{\mathsf{d}}\Theta; R \qquad \Theta;\Gamma \vdash v \Leftarrow \exists^{\mathsf{d}}\Theta.\, (R \wedge {}^{\mathsf{d}}\Theta)\end{array}}{\Theta;\Gamma \vdash \text{into}(v) \Leftarrow \left\{ v : \mu F \;\middle|\; \underbrace{\overrightarrow{(\text{fold}_F\ \alpha_i)\, v\, \boldsymbol{t}_i =_{\tau_i} u_i}^{i}}_{\mathscr{M}(F)} \right\}}\ \text{Decl}\Leftarrow\mu$$

$$\frac{\Theta;\Gamma \vdash e \Leftarrow N}{\Theta;\Gamma \vdash \{e\} \Leftarrow \downarrow N}\ \text{Decl}\Leftarrow\downarrow$$

$\boxed{\Theta;\Gamma \vdash e \Leftarrow N}$ Under inputs Θ and Γ, input expression e checks against input type N

$$\frac{\Theta;\Gamma \vdash v \Leftarrow P}{\Theta;\Gamma \vdash \text{return}\, v \Leftarrow \uparrow P}\ \text{Decl}\Leftarrow\uparrow$$

$$\frac{\Theta;\Gamma \vdash g \Rightarrow \uparrow(\exists^{\mathsf{d}}\Xi.\, R \wedge \overrightarrow{\psi}) \qquad \Theta, {}^{\mathsf{d}}\Xi, \overrightarrow{\psi};\Gamma, x : R \vdash e \Leftarrow L}{\Theta;\Gamma \vdash \text{let}\ x = g;\ e \Leftarrow L}\ \text{Decl}\Leftarrow\text{let}$$

$$\frac{\Theta;\Gamma \vdash h \Rightarrow P \qquad \Theta;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow L}{\Theta;\Gamma \vdash \text{match}\ h\ \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow L}\ \text{Decl}\Leftarrow\text{match}$$

$$\frac{\Theta;\Gamma, x : R \vdash e \Leftarrow L}{\Theta;\Gamma \vdash \lambda x.e \Leftarrow R \to L}\ \text{Decl}\Leftarrow\lambda \qquad \frac{\Theta \vdash \mathsf{ff}\ \mathsf{true}}{\Theta;\Gamma \vdash \text{unreachable} \Leftarrow L}\ \text{Decl}\Leftarrow\text{Unreachable}$$

$$\frac{\begin{array}{c}\Theta \vdash \forall a \overset{\mathsf{d}}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, M \leq^{-} L \\ \Theta, a \div \mathbb{N};\Gamma, x : \downarrow\forall a' \overset{\mathsf{d}}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, a' < a \supset [a'/a]M \vdash e \Leftarrow \forall^{\mathsf{d}}\Xi.\, M\end{array}}{\Theta;\Gamma \vdash \text{rec}\, x : (\forall a \overset{\mathsf{d}}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, M).\, e \Leftarrow L}\ \text{Decl}\Leftarrow\text{rec}$$

$$\frac{\Theta, {}^{\mathsf{d}}\Xi;\Gamma \vdash e \Leftarrow M}{\Theta;\Gamma \vdash e \Leftarrow \forall^{\mathsf{d}}\Xi.\, M}\ \text{Decl}\Leftarrow\forall \qquad \frac{\Theta, \overrightarrow{\varphi};\Gamma \vdash e \Leftarrow L}{\Theta;\Gamma \vdash e \Leftarrow \overrightarrow{\varphi} \supset L}\ \text{Decl}\Leftarrow\supset$$

Figure 6.20: Declarative value and expression type checking

$\boxed{\Theta;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}$ Under Θ and Γ, patterns r_i match against type P and branch expressions e_i check against type N (all inputs)

$$\frac{\Theta, {}^{\mathsf{d}}\Xi;\Gamma;[Q] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}{\Theta;\Gamma;[\exists^{\mathsf{d}}\Xi.\,Q] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}\ \text{DeclMatch}\exists$$

$$\frac{\Theta, \overrightarrow{\varphi};\Gamma;[R] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}{\Theta;\Gamma;[R \wedge \overrightarrow{\varphi}] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}\ \text{DeclMatch}\wedge$$

$$\frac{\Theta;\Gamma, x_1 : R_1, x_2 : R_2 \vdash e \Leftarrow N}{\Theta;\Gamma;[R_1 \times R_2] \vdash \{\langle x_1, x_2\rangle \Rightarrow e\} \Leftarrow N}\ \text{DeclMatch}\times$$

$$\frac{\Theta, {}^{\mathsf{d}}\Xi_1, \overrightarrow{\psi}_1;\Gamma, x_1 : R_1 \vdash e_1 \Leftarrow N \qquad \Theta, {}^{\mathsf{d}}\Xi_2, \overrightarrow{\psi}_2;\Gamma, x_2 : R_2 \vdash e_2 \Leftarrow N}{\Theta;\Gamma;[(\exists^{\mathsf{d}}\Xi_1.R_1 \wedge \overrightarrow{\psi}_1) + (\exists^{\mathsf{d}}\Xi_2.R_2 \wedge \overrightarrow{\psi}_2)] \vdash \{\mathtt{inj}_1\, x_1 \Rightarrow e_1 \mid \mathtt{inj}_2\, x_2 \Rightarrow e_2\} \Leftarrow N}\ \text{DeclMatch}+$$

$$\frac{}{\Theta;\Gamma;[0] \vdash \{\} \Leftarrow N}\ \text{DeclMatch0}$$

$$\frac{\mathscr{M}(F) \rightsquigarrow \overrightarrow{\alpha} \qquad {}^{\mathsf{d}\div}\Theta \vdash \wr \overrightarrow{\alpha}; F; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Theta; R \qquad \Theta, {}^{\mathsf{d}}\Theta;\Gamma, x : R \vdash e \Leftarrow N}{\Theta;\Gamma;[\{\nu : \mu F \mid \mathscr{M}(F)\}] \vdash \{\mathtt{into}(x) \Rightarrow e\} \Leftarrow N}\ \text{DeclMatch}\mu$$

$\boxed{\Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P}$ Under inputs Θ and Γ, if a head of type $\downarrow N$ (input: N) is applied to the spine s (input), then it will return a result of type $\uparrow P$ (output)

$$\frac{{}^{\mathsf{d}\div}\Theta \vdash \sigma : {}^{\mathsf{d}}\Xi \qquad \Theta;\Gamma;[[\sigma]M] \vdash s \Rightarrow \uparrow P}{\Theta;\Gamma;[\forall^{\mathsf{d}}\Xi.M] \vdash s \Rightarrow \uparrow P}\ \text{DeclSpine}\forall$$

$$\frac{\Theta \vdash \overrightarrow{\varphi}\,\mathsf{true} \qquad \Theta;\Gamma;[L] \vdash s \Rightarrow \uparrow P}{\Theta;\Gamma;[\overrightarrow{\varphi} \supset L] \vdash s \Rightarrow \uparrow P}\ \text{DeclSpine}\supset$$

$$\frac{\Theta;\Gamma \vdash v \Leftarrow R \qquad \Theta;\Gamma;[L] \vdash s \Rightarrow \uparrow P}{\Theta;\Gamma;[R \to L] \vdash v, s \Rightarrow \uparrow P}\ \text{DeclSpineApp}$$

$$\frac{}{\Theta;\Gamma;[\uparrow P] \vdash \cdot \Rightarrow \uparrow P}\ \text{DeclSpineNil}$$

Figure 6.21: Declarative pattern matching and spine typing

The judgment $\Theta;\Gamma \vdash g \Rightarrow \uparrow P$ (Fig. 6.19) synthesizes the type $\uparrow P$ from the bound expression g. Similarly to the synthesizing judgment for heads, this judgment is synthesizing because it is used in a cut rule Decl$\Leftarrow$let (the synthesized type is again the cut type). Bound expressions only synthesize an upshift because of their (lone) role in rule Decl$\Leftarrow$let, discussed later. For an application of a head to a spine (Decl$\Rightarrow$App), we first synthesize the head's type (which must be a downshift), and then check the spine against the thunked computation type, synthesizing the latter's return type. (Function applications must always be fully applied, but we can simulate partial application via η-expansion. For example, given $x : P_1$ and $h \Rightarrow \downarrow(P_1 \to P_2 \to \uparrow Q)$, to partially apply h to x we can write $\lambda y.\texttt{let}\ z = h(x,y); \cdots.$) For annotated expressions, we synthesize the annotation (Decl$\Rightarrow$ExpAnnot), which must be an upshift. If an e of type N is a function to be applied (as a head to a spine; Decl$\Rightarrow$App) only if the guards of N can be verified and the universally quantified indexes of N can be instantiated, then the programmer must annotate it like so: $(\texttt{return}\,\{e\} : \uparrow\downarrow N)$. The two annotation rules have explicit type well-formedness premises to emphasize that type annotations are provided by the programmer.

The judgment $\Theta;\Gamma \vdash v \Leftarrow P$ (Fig. 6.20) checks the value v against the type P. From a Curry–Howard perspective, this judgment corresponds to a *right-focusing* stage. According to rule Decl$\Leftarrow\exists$, a value checks against an existential type if there is an index instantiation it checks against (declaratively, indices are conjured, but algorithmically we will have to solve for them). For example, as discussed in Sec. 6.1, checking the program value one representing 1 against type $\exists a : \mathbb{N}.\,\mathsf{Nat}(a)$ solves a to an index semantically equal to 1. According to rule Decl$\Leftarrow\wedge$, a value checks against an asserting type if it the asserted proposition φ holds (and the value checks against the type to which φ is connected). Instead of

a general value type subsumption rule like

$$\frac{\Theta;\Gamma \vdash v \Leftarrow P' \qquad \Theta \vdash P' \leq^{+} P}{\Theta;\Gamma \vdash v \Leftarrow P}$$

we restrict the subsumption rule to simple (value) variables and simple supertypes, and prove that subsumption is admissible (see Section 6.6). This is easier to implement efficiently because the type checker would otherwise have to guess P' (and possibly need to backtrack), whereas Decl$\Leftarrow$Var need only look up the variable. Further, requiring the input type of Decl$\Leftarrow$Var to be simple means that any top-level $\exists$ or $\wedge$ constraints must be verified before subtyping, eliminating nondeterminism of verifying these in subtyping or typing.

Rule Decl$\Leftarrow\mu$ checks the unrolled value against the unrolled inductive type. Its first premise requires the unrolled value to not be in a certain form, namely it cannot be zero or more injections applied to zero or more right-associative pairs where the rightmost factor is a variable; this form should be excluded by elaboration from a surface language using named constructors. For example, a unary constructor cons (first among two constructors of an ADT) applied to a variable x would be elaborated to $\mathtt{into}(\mathtt{inj}_1\,\langle x, \langle\rangle\rangle)$, which is permissible under this restriction (the rightmost factor is $\langle\rangle$). This restriction on unrolled values greatly simplifies the typing algorithm and its metatheory.

Rule Decl$\Leftarrow$1 says $\langle\rangle$ checks against 1. Rule Decl$\Leftarrow\times$ says a pair checks against a product if each pair component checks against its corresponding factor. Rule Decl$\Leftarrow+_k$ says a value injected into the kth position checks against a sum if it can be checked against the kth summand. Rule Decl$\Leftarrow\downarrow$ checks the thunked expression against the computation type N under the given thunk type $\downarrow N$.

The judgment $\Theta;\Gamma \vdash e \Leftarrow N$ (Fig. 6.20) checks the expression e against the type N.

From a Curry–Howard perspective, this judgment is a *right-inversion* stage with *stable* moments (Decl⇐let and Decl⇐match, which enter left- or right-focusing stages, respectively). To eliminate Decl⇐∀ and Decl⇐⊃ nondeterminism the other (expression) rules must check against a *simple* type L. This means Decl⇐∀ and Decl⇐⊃ must always be applied first if possible.

All applications $h(s)$ must be named and sequenced via Decl⇐let, which we may think of as monadic binding, and is a key cut rule. Other computations—annotated returner expressions $(e : \uparrow P)$—must also be named and sequenced via Decl⇐let. It would not make sense to allow arbitrary negative annotations because that would require verifying constraints and instantiating indexes that should only be done when the annotated expression is applied, which does not occur in Decl⇐let itself.

Heads, that is, head variables and annotated values, can be pattern matched via rule Decl⇐match. From a Curry–Howard perspective, Decl⇐match is a cut rule dual to the cut rule Decl⇐let: the latter binds the result of a computation to a (sequenced) computation, whereas the former binds the deconstruction of a value to, and directs control flow of, a computation. Rule Decl⇐λ is standard (beside the fact that the arrow type must be simple). Rule Decl⇐rec requires an annotation that universally quantifies over the argument a that must be smaller at each recursive call, as dictated by its annotation in the last premise: $x : \downarrow\left(\forall a' : \mathbb{N}.\,(a' < a) \supset [a'/a]M\right)$ only allows x to be used for $a' < a$, ensuring that *refined* recursive functions are well-founded (according to $<$ on naturals). Rule Decl⇐↑ checks that the value being returned has the positive type under the given returner type (↑); this may be thought of as a monadic return operation. Rule Decl⇐Unreachable says that unreachable checks against any type, provided the logical context is inconsistent; for example, an impossible pattern in pattern matching extracts to an inconsistent context.

Rule Decl$\Leftarrow$rec appears to handle only one termination metric, namely $<$ on natural numbers. But it actually handles more. For example, if either n_1 or n_2 get smaller in applying a function of type $\forall n_1, n_2. M$ then $\forall n, n_1, n_2. n = n_1 + n_2 \supset M$ suffices for Decl$\Leftarrow$rec. However, we probably cannot simulate lexicographic induction. Adding further termination metrics should be straightforward but it would also take a lot of work and not be terribly interesting. Regardless, it would be sensible to make the metric parametric in the typing rule for (total) recursive expressions. It may be possible to use program-level ghost parameters (like the one we used in the old mergesort example of Economou et al. [2023] to express the sum of integers getting smaller) to simulate lexicographic induction but that takes us away from the spirit of refinement types (programmers shouldn't have to refactor their code so much).

The judgment $\Theta; \Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$ (Fig. 6.21) decomposes P, according to patterns r_i (if $P \neq \wedge$ or $\exists$, which have no computational content; if $P = \wedge$ or $\exists$, the index is put in logical context for use), and checks that each branch e_i has type N. The rules are straightforward. Indexes from matching on existential and asserting types are used, not verified (as in value typechecking); we deconstruct heads, and to synthesize a type for a head, its indexes must hold, so within the pattern matching stage itself we may assume and use them. From a Curry–Howard perspective, this judgment corresponds to a *left-inversion* stage. However, it is not *strongly* focused, that is, it does not decompose P eagerly and as far as possible; therefore, "stage" might be slightly misleading. If our system were more strongly focused, we would have nested patterns, at least for all positive types except inductive types; it's unclear how strong focusing on inductive types would work.

The judgment $\Theta; \Gamma; [N] \vdash s \Rightarrow \uparrow P$ (Fig. 6.21) checks the spine s against N, synthesizing the return type $\uparrow P$. From a Curry–Howard perspective, this judgment corresponds to a

left-focusing stage. The rules are straightforward: decompose the given N, checking index constraints (DeclSpine$\forall$ and DeclSpine$\supset$) and values (DeclSpineApp) until an upshift, the return type, is synthesized (DeclSpineNil). Similarly to dual rule Decl$\Leftarrow\exists$, the declarative rule DeclSpine$\forall$ conjures indices measuring a value, but in this case argument values of a spine. For example, in applying a head of type $\forall a : \mathbb{N}.\, \mathsf{Nat}(a) \to \uparrow \mathsf{Nat}(a)$ to the spine with program value one representing 1, we must instantiate a to an index semantically equal to 1; we show how this works algorithmically in Sec. 8.5. All universal quantifiers (in the input type of a spine judgment) are solvable algorithmically because they are value-determined.

6.6 Program Substitution

A key correctness result that we prove is a substitution lemma: substitution (of index terms for index variables and program values for program variables) preserves typing. We now extend the index-level syntactic substitutions (and the parallel substitution operation) introduced in Chapter 6.2. A *syntactic substitution* $\sigma ::= \cdot \mid \sigma, t/a \mid \sigma, v : P/x$ is essentially a list of terms to be substituted for variables. Substitution application $[\sigma]-$ is a parallel substitution metaoperation on types and terms. On program terms, it avoids full *hereditary substitution*[12] [Watkins et al., 2004, Pfenning, 2008] at the program level, in the sense that, at head variables (note the h superscript in the Fig. 6.22 definition; we may elide h if clear from context), an annotation is produced if the value and the head variable being replaced by it are not equal—thereby modifying the syntax tree of the substitutee *but not reducing it*. Otherwise, substitution is standard (homomorphic application) and does not use the value's associated type given in σ: see Fig. 6.22.

In the definition given in Fig. 6.22, an annotation is not produced if $v = x$ so that $x : P/x$

[12] Typically, hereditary substitution reduces terms after substitution, modifying the syntax tree.

$$
\begin{aligned}
[\sigma]^{\mathsf{h}} x &= \begin{cases} x & \text{if } x \notin dom(\sigma) \text{ or } \sigma(x) = (x : P) \\ \sigma(x) & \text{else} \end{cases} \\
[\sigma]^{\mathsf{h}}(v : P) &= ([\sigma]v : [\lfloor\sigma\rfloor]P) \\
\\
[\sigma](h(s)) &= ([\sigma]^{\mathsf{h}} h)([\sigma]s) \\
[\sigma](e : \uparrow P) &= ([\sigma]e : [\lfloor\sigma\rfloor](\uparrow P)) \\
\\
[\sigma]x &= \begin{cases} x & \text{if } x \notin dom(\sigma) \\ v & \text{if } \sigma(x) = (v : P) \end{cases} \\
[\sigma]\langle v_1, v_2\rangle &= \langle [\sigma]v_1, [\sigma]v_2\rangle \\
&\vdots \\
[\sigma](\mathtt{match}\ h\ \{r_i \Rightarrow e_i\}_{i\in I}) &= \mathtt{match}\ \left([\sigma]^{\mathsf{h}} h\right)\ ([\sigma]\{r_i \Rightarrow e_i\}_{i\in I}) \\
&\vdots \\
[\sigma](\lambda x.e) &= \lambda x.[\sigma]e \\
[\sigma](\mathtt{rec}\, x : (\forall a : \mathbb{N}.N).\ e) &= \mathtt{rec}\, x : [\lfloor\sigma\rfloor](\forall a : \mathbb{N}.N).\ [\sigma]e \\
&\vdots
\end{aligned}
$$

Figure 6.22: Definition of syntactic substitution on program terms

is always an *identity* substitution: that is, $[_, x : P/x, _]^{\mathsf{h}} x = x$ (the program terms x and $(x : P)$ are syntactically distinct). As usual, we assume variables are alpha-renamed to avoid capture by substitution.

The judgment $\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma$ (appendix Fig. A.19) presupposes the well-formedness of all the contexts, and means that, under Θ_0 and Γ_0, we know σ is a substitution of index terms and program values for variables in Θ and Γ, respectively. The key rule of this judgment is for program value entries (the three elided rules are similar to the three rules for syntactic substitution typing at index level, found in Sec. 6.2, but adds program contexts

Γ where appropriate):

$$\frac{\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma \qquad \Theta_0;\Gamma_0 \vdash v \Leftarrow \lfloor[\sigma]\rfloor P \qquad x \notin dom(\Gamma)}{\Theta_0;\Gamma_0 \vdash (\sigma, v : P/x) : \Theta;\Gamma, x : P}$$

Economou et al. [2023] applied σ to v in the premise because their substitution was applied sequentially, but now substitution is parallel so this is unnecessary. The value v may mention variables in $dom(\Gamma_0)$ and $dom(\Theta_0)$, and P may mention variables in $dom(\Theta)$. The metaoperation $\lfloor - \rfloor$ filters out program variable entries (program variables cannot appear in types, functors, algebras or indexes).

That substitution respects typing is an important correctness property of the type system. All six parts are mutually recursive.

Lemma 6.10 (Syntactic Substitution)**.** *(Lemma C.82 in appendix)*

Assume $\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma$.

(1) If $\Theta;\Gamma \vdash h \Rightarrow P$

then there exists P' *such that* $\Theta_0 \vdash P' \leq^+ \lfloor[\sigma]\rfloor P$ *and* $\Theta_0;\Gamma_0 \vdash [\sigma]^{\mathsf{h}} h \Rightarrow P'$.

Moreover, either (a) $P' = \lfloor[\sigma]\rfloor P$ *or (b)* $P' = R$ *for some* R.

(2) If $\Theta;\Gamma \vdash g \Rightarrow \uparrow P$

then there exists P' *such that* $\Theta_0 \vdash \uparrow P' \leq^- \lfloor[\sigma]\rfloor \uparrow P$ *and* $\Theta_0;\Gamma_0 \vdash [\sigma]g \Rightarrow \uparrow P'$.

(3) If $\Theta;\Gamma \vdash v \Leftarrow P$ *then* $\Theta_0;\Gamma_0 \vdash [\sigma]v \Leftarrow \lfloor[\sigma]\rfloor P$.

(4) If $\Theta;\Gamma \vdash e \Leftarrow N$ *then* $\Theta_0;\Gamma_0 \vdash [\sigma]e \Leftarrow \lfloor[\sigma]\rfloor N$.

(5) If $\Theta;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$

then $\Theta_0;\Gamma_0;[\lfloor[\sigma]\rfloor P] \vdash [\sigma]\{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow \lfloor[\sigma]\rfloor N$.

(6) If $\Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P$ *then* $\Theta_0;\Gamma_0;[[\lfloor\sigma\rfloor]N] \vdash [\sigma]s \Rightarrow [\lfloor\sigma\rfloor]\uparrow P$.

In part (1), if substitution creates a head variable with stronger type, then the stronger type P' is synthesized. The proof relies on other structural properties such as weakening. It also relies on subsumption admissibility, which captures what we mean by "stronger type". All parts are mutually recursive. The proof of subsumption admissibility uses previously mentioned lemmas 6.9, 6.7, 6.6, 6.8, and on the interaction of subtyping and unrolling. The former two for checking types against inductive values and, dually, the latter two for pattern-matching on inductive values.

Lemma 6.11 (Subsumption Admissibility)**.** *(Lemma C.80 in appendix)*

Assume $\Theta \vdash \Gamma' \leq^+ \Gamma$. *Then:*

(1) If $\Theta;\Gamma \vdash h \Rightarrow P$

then there exists P' *such that* $\Theta \vdash P' \leq^+ P$ *and* $\Theta;\Gamma' \vdash h \Rightarrow P'$.

Moreover, either (a) $P' = P$ *or (b)* $P' = R$ *for some* R.

(2) If $\Theta;\Gamma \vdash g \Rightarrow \uparrow P$

then there exists P' *such that* $\Theta \vdash \uparrow P' \leq^- \uparrow P$ *and* $\Theta;\Gamma' \vdash g \Rightarrow \uparrow P'$.

(3) If $\Theta;\Gamma \vdash v \Leftarrow P$ *and* $\Theta \vdash P \leq^+ P'$, *then* $\Theta;\Gamma' \vdash v \Leftarrow P'$.

(4) If $\Theta;\Gamma \vdash e \Leftarrow N$ *and* $\Theta \vdash N \leq^- N'$, *then* $\Theta;\Gamma' \vdash e \Leftarrow N'$.

(5) If $\Theta;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$ *and* $\Theta \vdash N \leq^- N'$ *and* $\Theta \vdash P' \leq^+ P$

then $\Theta;\Gamma';[P'] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N'$.

(6) If $\Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P$ *and* $\Theta \vdash N' \leq^- N$

then there exists P' *such that* $\Theta \vdash \uparrow P' \leq^- \uparrow P$ *and* $\Theta;\Gamma';[N'] \vdash s \Rightarrow \uparrow P'$.

Subtypes are stronger than supertypes. That is, if we can check a value against a type, then we know that it also checks against any of the type's supertypes; similarly for expressions. Pattern matching is similar, but it also says we can match on a stronger type. A head or bound expression can synthesize a stronger type under a stronger context. Similarly, with a stronger input type, a spine can synthesize a stronger return type.

Chapter 7

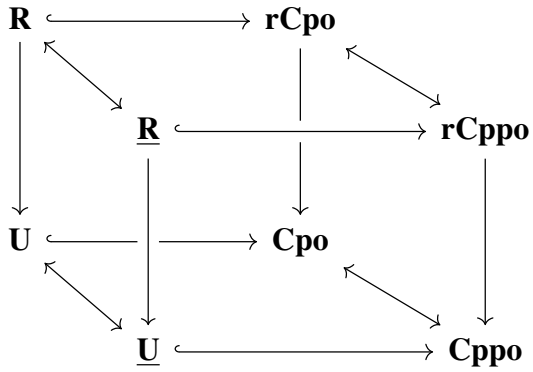
Semantics and Typing Soundness

We prove type (and substitution) soundness of the declarative (refined) system with respect to a domain-theoretic denotational semantics. Refined typing soundness implies the refined system's totality and logical consistency.

Refinement type systems refine already-given type systems, and the soundness of the former depends on that of the latter [Melliès and Zeilberger, 2015]. Thus, the semantics of our refined system is defined in terms of that of its underlying, unrefined system, which we discussed in Chapter 5.

The category **rCpo** of refined cpos has (D,R) as objects (corresponding to value types) where D is a cpo and $R \subseteq D$ (this is a slight abuse of notation: we mean R is a subset of the underlying set of D which is a set equipped with a complete partial order), and a morphism (corresponding to a refined value typing derivation) $f : (D_1,R_1) \to (D_2,R_2)$ in **rCpo** is a morphism $f : D_1 \to D_2$ in **Cpo** such that $f(R_1) \subseteq R_2$. The category **rCppo** is defined similarly to **rCpo**. A refined functor applies its erasure to D and the refined functor itself to R; and its application to morphisms is just the application of its erasure. Our refinement system semantically corresponds to an adjunction of lift and thunk functors between

subcategories **R** and $\underline{\mathbf{R}}$ of **rCpo** and **rCppo** respectively; the unrefined system also semantically corresponds to an adjunction of lift and thunk functors but between subcategories **U** and $\underline{\mathbf{U}}$ of **Cpo** and **Cppo** respectively (**R** and **U** stand for "refined" and "unrefined" at value level; when underlined, at computation level). In terms of Melliès and Zeilberger [2015] our refined system can be described by the commutative diagram



where the functors going straight down forget the refinement subset R and the diagonal arrows are pairs of adjoint lift and thunk functors where, as usual, lifting is left adjoint.

Indexes A sort τ denotes $(\llbracket\tau\rrbracket, \sqsubseteq_{\llbracket\tau\rrbracket})$ where $\sqsubseteq_{\llbracket\tau\rrbracket} = \{(d,d) \mid d \in \llbracket\tau\rrbracket\}$ is the discrete order, which is a cpo.

Semantic Substitution We introduced semantic substitutions δ (at the index level) when discussing propositional validity (Sec. 6.2). Here, they are extended in the obvious way to semantic program values by adding the rule

$$\frac{\vdash \delta : \Theta;\Gamma \qquad V \in \llbracket P\rrbracket_{\lfloor\delta\rfloor} \qquad x \notin dom(\Gamma)}{\vdash (\delta, V/x) : \Theta;\Gamma, x : P}$$

(where $\lfloor - \rfloor$ filters out program entries, which is not strictly necessary here, but it emphasizes that no program variables occur in types) and modifying the other rules:

$$\frac{}{\vdash \cdot : \cdot ; \cdot} \qquad \frac{\vdash \delta : \Theta ; \Gamma \qquad d \in [\![\tau]\!] \qquad a \notin dom(\Theta)}{\vdash \delta, d/a : \Theta, a : \tau[, a\,\mathrm{Id}] ; \Gamma} \qquad \frac{\vdash \delta : \Theta ; \Gamma \qquad [\![\varphi]\!]_\delta = \{\bullet\}}{\vdash \delta : \Theta, \varphi ; \Gamma}$$

Notation: we define $[\![\Theta;\Gamma]\!] = \{\delta \mid \vdash \delta : \Theta;\Gamma\}$ and $[\![\Theta]\!] = [\![\Theta;\cdot]\!]$ and $[\![\Gamma]\!] = [\![\cdot;\Gamma]\!]$.

Index erasure The *index erasure* metaoperation $|-|$ (Figs. 7.1, 7.2, and 7.3) erases all indexes from well-formed (refined) types, well-typed program terms (which can have type annotations, but those do not affect program denotation), and (well-typed) syntactic and semantic substitutions.

$$\begin{aligned}
|1| &= 1 \\
|R \times R'| &= |R| \times |R'| \\
|0| &= 0 \\
|P + P'| &= |P| + |P'| \\
|{\downarrow}N| &= {\downarrow}|N| \\
|\{\nu : \mu F \mid \mathscr{M}(F)\}| &= \mu |F| \\
|\exists a : \kappa . P| &= |P| \\
|Q \wedge \varphi| &= |Q| \\
|R \to L| &= |R| \to |L| \\
|{\uparrow}P| &= {\uparrow}|P| \\
|\forall a : \kappa . N| &= |N| \\
|\varphi \supset M| &= |M|
\end{aligned}
\qquad
\begin{aligned}
|F \oplus G| &= |F| \oplus |G| \\
|I| &= I \\
|\hat{B} \otimes \hat{P}| &= |\hat{B}| \otimes |\hat{P}| \\
|\underline{P}| &= \underline{|P|} \\
|\mathrm{Id}| &= \mathrm{Id}
\end{aligned}$$

Figure 7.1: Index erasure of refined types and functors to unrefined types and functors

We use many facts about erasure to prove refined type/substitution soundness (appendix lemmas):

$$
\begin{aligned}
|x| &= x \\
|(v : P)| &= (|v| : |P|) \\
|h(s)| &= |h|(|s|) \\
|(e : N)| &= (|e| : |N|) \\
|\cdot| &= \cdot \\
|v, s| &= |v|, |s| \\
|\langle\rangle| &= \langle\rangle \\
|\langle v_1, v_2\rangle| &= \langle |v_1|, |v_2| \rangle \\
|\mathsf{inj}_1\, v| &= \mathsf{inj}_1\, |v| \\
|\mathsf{inj}_2\, v| &= \mathsf{inj}_2\, |v| \\
|\mathsf{into}(v)| &= \mathsf{into}(|v|) \\
|\{e\}| &= \{|e|\} \\
|\mathsf{return}\, v| &= \mathsf{return}\, |v| \\
|\mathsf{let}\, x = g;\, e| &= \mathsf{let}\, x = |g|;\, |e| \\
|\mathsf{match}\, h\, \{r_i \Rightarrow e_i\}_{i \in I}| &= \mathsf{match}\, |h|\, |\{r_i \Rightarrow e_i\}_{i \in I}| \\
|\lambda x.e| &= \lambda x.|e| \\
|\mathsf{rec}\, x : (\forall a : \mathbb{N}.N).\, e| &= \mathsf{rec}\, x.\, |e| \\
|\Theta; \Gamma \vdash \mathsf{unreachable} \Leftarrow L| &= \mathsf{diverge}_{|L|} \\
|\{r_i \Rightarrow e_i\}_{i \in I}| &= \{r_i \Rightarrow |e_i|\}_{i \in I}
\end{aligned}
$$

Figure 7.2: Index erasure of refined program terms

$$
\begin{aligned}
|\cdot| &= \cdot & \qquad |\cdot| &= \cdot \\
|\sigma, t/a| &= |\sigma| & \qquad |\delta, d/a| &= |\delta| \\
|\sigma, v : R/x| &= |\sigma|, |v| : |R|/x & \qquad |\delta, V/x| &= |\delta|, V/x
\end{aligned}
$$

Figure 7.3: Index erasure of substitution

- Refined types and functors denote subsets of what their erasures denote: Lemma D.26 (Refinement Subset of Erasure).
- Refined and unrefined functor applications agree on refined domain: Lemma D.27 (Refined and Unrefined fmap Agree).
- Index substitution does not affect result of erasure: Lemma D.28 (Erasure Subst. Invariant).
- The erasure of both types appearing in subtyping judgments results in equal (unrefined) types: Lemma D.29 (Subtyping Erases to Equality).
- Refined unrolling and typing are sound with respect to their erasure: Lemma D.30 (Unrolling Erasure), Lemma D.31 (Erasure of Typing), and Lemma D.33 (Erasure of Substitution Typing).
- Erasure commutes with syntactic and semantic substitution: Lemma D.32 (Erasure and Substitution Commute) and Lemma D.34 ($|-|$ and $[\![-]\!]$ Commute).

Types, functors, algebras, and folds The denotations of refined types and functors are defined as logical subsets of the denotations of their erasures (together with their erasure denotations themselves). They are defined mutually with the denotations of well-formed algebras.

In appendix Fig. A.49, we recursively define the denotations of well-formed types $\Xi \vdash A\ \mathsf{type}[_]$. This judgment is mutually recursive with functor and algebra well-formedness, discussed soon. We prove (refined) types denote refined cp(p)os, (refined) functors denote refined cpos, and (refined) algebras denote algebras over (refined) functors. By a refined cp(p)o we mean a cp(p)o together with a subset of it.

We briefly discuss a few of the cases of the denotation of a (refined) type, well-formed under Ξ, at $\delta \in [\![\Xi]\!]$. The denotation of an asserting type is the set of refined values such that the asserted index proposition holds (read $\{\bullet\}$ as true and $\emptyset$ as false):

$$[\![Q \wedge \varphi]\!]_\delta = \{V \in [\![|Q|]\!] \mid V \in [\![Q]\!]_\delta \text{ and } [\![\varphi]\!]_\delta = \{\bullet\}\}$$

Existential and universal types denote elements of their erasure such that the relevant index quantification holds:

$$[\![\exists a : \tau. P]\!]_\delta = \left\{V \in [\![|P|]\!] \;\middle|\; \exists d \in [\![\tau]\!]. V \in [\![P]\!]_{\delta,d/a}\right\}$$
$$[\![\forall a : \tau. N]\!]_\delta = \left\{f \in [\![|N|]\!] \;\middle|\; \forall d \in [\![\tau]\!]. f \in [\![N]\!]_{\delta,d/a}\right\}$$

Guarding types denote elements of their erasure such that they are also in the refined type being guarded if the guard holds ($\{\bullet\}$ means true):

$$[\![\varphi \supset M]\!]_\delta = \{f \in [\![|M|]\!] \mid \text{if } [\![\varphi]\!]_\delta = \{\bullet\} \text{ then } f \in [\![M]\!]_\delta\}$$

The denotation of refined function types $[\![R \to L]\!]_\delta$ is *not* the set $[\![R]\!]_\delta \Rightarrow [\![L]\!]_\delta$ of (continuous) functions from refined R-values to refined L-values; if it were, then typing soundness would break:

$$\begin{aligned}[\![\cdot;\cdot \vdash \lambda x.\mathtt{return}\, x \Leftarrow ((1 \wedge \mathsf{ff}) + (1 \wedge \mathsf{ff})) \to \uparrow 1]\!]. &= [\![|\lambda x.\mathtt{return}\, x|]\!]. \\ &= [\![\lambda x.\mathtt{return}\, x]\!]. \\ &= (y \mapsto inj_1\, y)\end{aligned}$$

which is not in the empty set ($\emptyset \Rightarrow \{\bullet\} \uplus \{\bot_\uparrow\}$). Instead, the denotation of a refined function type is a set

$$\{f \in [\![|R \to L|]\!] \mid \forall V \in [\![R]\!]_\delta . f(V) \in [\![L]\!]_\delta\}$$

of *unrefined* (continuous) functions that take refined values to refined values. The denotation of *refined* upshifts enforces termination (if refined typing soundness holds, and we will see it does):

$$[\![\uparrow P]\!]_\delta = \{inj_1 V \mid V \in [\![P]\!]_\delta\}$$

Note that divergence $inj_2 \bot_\uparrow$ is *not* in the set $[\![\uparrow P]\!]_\delta$.

In Fig. A.50, we recursively define the denotations of well-formed refined functors F and algebras α (mutually with well-formed refined types, discussed previously). The main difference between refined and unrefined functors is that in refined functors, constant functors produce subsets of their erasure. Is is straightforward to prove all functors, refined or otherwise, (forgetting the partial order structure) denote endofunctors on the category of sets and functions.

As with our unrefined functors, we prove (Lemma E.10) our refined functors denote functors with a fixed point: $[\![F]\!]_\delta (\mu [\![F]\!]_\delta) = \mu [\![F]\!]_\delta$. Moreover, $\mu [\![F]\!]_\delta$ satisfies a recursion principle such that we can recursively define measures on $\mu [\![F]\!]_\delta$ via $[\![F]\!]_\delta$-algebras (discussed next).

Categorically, given an endofunctor F, an *F-algebra* is an evaluator map $\alpha : F(\tau) \to \tau$ for some carrier set τ. We may think of this in terms of elementary algebra: we form algebraic expressions with F and evaluate them with α. A morphism f from algebra $\alpha : F(\tau) \to \tau$ to algebra $\beta : F(\tau') \to \tau'$ is a morphism $f : \tau \to \tau'$ such that $f \circ \alpha = \beta \circ (F(f))$.

If an endofunctor F has an *initial*[1] algebra $\mathit{into} : F(\mu F) \to \mu F$, then it has a recursion principle. By the recursion principle for μF, we can define a recursive function from μF to τ by *folding* μF with an F-algebra $\alpha : F(\tau) \to \tau$ like so:

$$
\begin{aligned}
&(\mathit{fold}_F\ \alpha) : \mu F \to \tau \\
&(\mathit{fold}_F\ \alpha)\ v = \alpha\ \Big(\big(\mathit{fmap}\ F\ (\mathit{fold}_F\ \alpha)\big)\ \big(\mathit{out}(v)\big)\Big)
\end{aligned}
$$

where $\mathit{out} : \mu F \to F(\mu F)$, which by Lambek's lemma exists and is inverse to *into*, embeds (semantic) inductive values into the unrolling of the (semantic) inductive type (we usually elide *fmap*). Conveniently, in our system's semantics, *out* is always $d \mapsto d$, and we almost never explicitly mention it. Syntactic values v in our system must be rolled into inductive types—$\texttt{into}(v)$—and this is also how (syntactic) inductive values are pattern-matched ("applying *out*" to $\texttt{into}(v)$), but $\texttt{into}(-)$ conveniently also denotes $d \mapsto d$ (our *into*) and we also almost never explicitly mention *into*.

We specify inductive types abstractly as sums of products so that they denote polynomial endofunctors more directly. Polynomial endofunctors always have a "least" (initial) fixed point[2], and hence specify inductive types, which have a recursion principle. For example, we specify (modulo the unrolling simplification) $\mathsf{len} : \mathsf{List}\mathsf{F}_A(\mathbb{N}) \Rightarrow \mathbb{N}$ (Chapter 1) by the (syntactic) algebra

$$\alpha\ =\ \mathrm{inj}_1\ () \Rightarrow 0 \mid \mathrm{inj}_2\ (\top, a) \Rightarrow 1 + a$$

[1] An object X in a category $\mathbf{C}$ is *initial* if for every object Y in $\mathbf{C}$, there exists a unique morphism $X \to Y$ in $\mathbf{C}$. Algebras and their homomorphisms form a category.

[2] This is not necessarily the case for all endofunctors. Therefore, not all endofunctors can be said to specify an inductive type. For example, consider the powerset functor (assuming a conventional logic—in an unconventional logic it is possible to prove the universal set equals its powerset [Petersen, 2023]).

which denotes the (semantic) algebra

$$[\![\alpha]\!] : \underbrace{[\![\mathsf{ListF}_A]\!]\,(\mathbb{N})}_{1 \uplus ([\![A]\!] \times \mathbb{N})} \to \mathbb{N}$$

defined by $[\![\alpha]\!] = [\bullet \mapsto 0, (a,n) \mapsto 1+n]$. By initiality (the recursion principle), there is a unique function

$$\mathit{fold}_{[\![\mathsf{ListF}_A]\!]}\ [\![\alpha]\!] : \mu\,[\![\mathsf{ListF}_A]\!] \to \mathbb{N}$$

such that $\mathit{fold}_{[\![\mathsf{ListF}_A]\!]}\ [\![\alpha]\!] = [\![\alpha]\!] \circ ([\![F]\!]\,(\mathit{fold}_{[\![\mathsf{ListF}_A]\!]}\ [\![\alpha]\!]))$, which semantically captures len (Chapter 1). Lemma E.9 (Semantic Fold) proves this equality for any F and α.

In our system, a well-formed refined ADT $\{v : \mu F \mid \mathscr{M}(F)\}$ denotes

$$\{V \in \mu\,[\![|F|]\!] \mid V \in \mu\,[\![F]\!]_\cdot \text{ and } [\![\mathscr{M}(F)]\!]_\delta\, V = \{\bullet\}\}$$

where $[\![\mathscr{M}(F)]\!]_\delta\, V$ is true (that is, $\{\bullet\}$) if and only if

$$[\![t]\!]_{\delta'}\,((\mathit{fold}_{[\![F]\!]_\cdot}\ [\![\alpha]\!]_\cdot)\ V) = [\![t]\!]_{\delta'} \text{ for all } (\mathsf{fold}_F\ \alpha)\,v\,\boldsymbol{t} =_\tau t \in \mathscr{M}(F)$$

where $\delta' = \delta\restriction_{\mathsf{d}_\div\,\Xi}$.

A well-formed algebra $\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$ denotes a dependent function

$$\prod_{\delta \in [\![\Xi]\!]} [\![F]\!]_\delta\,[\![\tau]\!] \to [\![\tau]\!]$$

The definition (appendix Fig. A.50) is mostly standard, but the unit and pack cases could use some explanation. To emphasize that the bodies of algebras must be value-determined,

we restrict δ to $\overset{\mathsf{d}}{\div}\Xi$ at algebra bodies:

$$[\![\Xi \vdash () \Rightarrow t : I(\tau) \Rightarrow \tau]\!]_\delta \bullet \;=\; [\![\overset{\mathsf{d}}{\div}\Xi \vdash t : \tau]\!]_{\delta\restriction_{\overset{\mathsf{d}}{\div}\Xi}}$$

This is not strictly necessary, though, because weakening a derivation does not change its denotation (see, for example, Lemma E.5 ($[\![-]\!]$ Weakening Invariant); we often use such weakening denotation invariance lemmas implicitly). The most interesting part of the definition concerns index packing in measures:

$$\begin{aligned}\left[\!\!\left[\Xi \vdash (\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q) \Rightarrow t : (\underline{\exists^{\mathsf{d}}\Xi'.\, Q} \otimes \hat{P})(\tau) \Rightarrow \tau\right]\!\!\right]_\delta (V_1, V_2) = \\ \left[\!\!\left[\Xi, {}^{\mathsf{d}}\Xi' \vdash (\top, q) \Rightarrow t : (\underline{Q} \otimes \hat{P})(\tau) \Rightarrow \tau\right]\!\!\right]_{(\delta,\delta')} (V_1, V_2) \\ \text{where } \delta' \in \left[\!\!\left[{}^{\mathsf{d}}\Xi'\right]\!\!\right] \text{ satisfies } V_1 \in [\![Q]\!]_{\delta,\delta'}\end{aligned}$$

The pack clause lets us bind the witness δ' of ${}^{\mathsf{d}}\Xi'$ in the existential type $\exists^{\mathsf{d}}\Xi'.\, Q$ to $dom({}^{\mathsf{d}}\Xi')$ in the body t of the algebra. We know δ' exists since $V_1 \in \left[\!\!\left[\exists^{\mathsf{d}}\Xi'.\, Q\right]\!\!\right]_\delta$, but it is not immediate that it is unique. However, we prove δ' is uniquely determined by V_1. We call this property the *soundness of value-determined indexes*: syntactic value-determined indices are determined uniquely by semantic values. It is decomposed into two lemmas, the *soundness of value-determined dependencies* (the ξ outputs of well-formedness judgments) and the *dependency agreement closure*.

We define dependency agreement $\delta_1\restriction_\xi = \delta_2\restriction_\xi$ (that is δ_1 and δ_2 agree at dependencies ξ) by for all $\mathfrak{B}{\to}a \in \xi$ if $\delta_1\restriction_{\mathfrak{B}} = \delta_2\restriction_{\mathfrak{B}}$ then $\delta_1(a) = \delta_2(a)$.

Lemma 7.1 (Soundness of Value-Determined Dependencies)**.** *(Lemma E.17)*

Assume $\vdash \delta_1 : \Xi$ *and* $\delta_2 : \Xi$.

(1) If $\Xi \vdash P\ \mathsf{type}[\xi]$ *and* $V \in [\![P]\!]_{\delta_1}$ *and* $V \in [\![P]\!]_{\delta_2}$ *then* $\delta_1\restriction_\xi = \delta_2\restriction_\xi$.

(2) If $\Xi \vdash \mathscr{F}\ \mathsf{functor}[\xi]$ *and* X_1 *and* X_2 *are sets and* $V \in [\![\mathscr{F}]\!]_{\delta_1} X_1$ *and* $V \in [\![\mathscr{F}]\!]_{\delta_2} X_2$ *then* $\delta_1 \restriction_\xi = \delta_2 \restriction_\xi$.

(3) If $\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi]$ *and* $V \in \mu\,[\![F]\!].$ *and* $[\![\mathscr{M}(F)]\!]_{\delta_1} V = \{\bullet\} = [\![\mathscr{M}(F)]\!]_{\delta_2} V$ *then* $\delta_1 \restriction_\xi = \delta_2 \restriction_\xi$.

If according to ξ the index a depends on $\mathfrak{C}$ the first part says a is uniquely determined by the semantic value V if the indices in $\mathfrak{C}$ are uniquely determined. The next lemma shows dependency agreement interacts with dependency closure as expected. If δ_1 and δ_2 agree at ξ and δ_1 and δ_2 agree at ${}^{\mathsf{d}}\Xi'$ and ${}^{\mathsf{d}}\Xi_0$ is value-determined under ξ assuming ${}^{\mathsf{d}}\Xi'$ is value-determined then δ_1 and δ_2 agree at ${}^{\mathsf{d}}\Xi_0$.

Lemma 7.2 (Dependency Agreement Closure)**.** *(Lemma E.15)*

If $\delta_1 \restriction_\xi = \delta_2 \restriction_\xi$ *and* $\delta_1 \restriction_{{}^{\mathsf{d}}\Xi'} = \delta_2 \restriction_{{}^{\mathsf{d}}\Xi'}$ *and* $\xi - {}^{\mathsf{d}}\Xi' \vdash {}^{\mathsf{d}}\Xi_0\ \mathsf{det}$

then $\delta_1 \restriction_{{}^{\mathsf{d}}\Xi_0} = \delta_2 \restriction_{{}^{\mathsf{d}}\Xi_0}$.

We only need these two lemmas in situations involving *measures*, so that the Ξ of the first part (the type part) of the former is ${}^{\mathsf{d}}\Xi$. Suppose ${}^{\mathsf{d}}\Xi \vdash \exists {}^{\mathsf{d}}\Xi'.\, Q\ \mathsf{type}[\xi]$ and $\delta \in [\![{}^{\mathsf{d}}\Xi]\!]$. By inversion on type well-formedness, $\xi_Q - {}^{\mathsf{d}}\dot{-}{}^{\mathsf{d}}\Xi \vdash {}^{\mathsf{d}}\Xi'\ \mathsf{det}$. Since ${}^{\mathsf{d}}\Xi$ is all value-determined, its value-determined sublist ${}^{\mathsf{d}}\dot{-}{}^{\mathsf{d}}\Xi$ is itself: $\xi_Q - {}^{\mathsf{d}}\Xi \vdash {}^{\mathsf{d}}\Xi'\ \mathsf{det}$. Due to the way we interpret pack algebras, in the metatheory we sometimes get into a situation where $\delta_1 \in \left[\!\!\left[{}^{\mathsf{d}}\Xi'\right]\!\!\right]$ and $\delta_2 \in \left[\!\!\left[{}^{\mathsf{d}}\Xi'\right]\!\!\right]$ and $V \in [\![Q]\!]_{\delta,\delta_1}$ and $V \in [\![Q]\!]_{\delta,\delta_2}$ and we need to prove $[\![\alpha]\!]_{\delta,\delta_1} V = [\![\alpha]\!]_{\delta,\delta_2} V$. By Lemma 7.1 (Soundness of Value-Determined Dependencies) we have $(\delta,\delta_1)\restriction_{\xi_Q} = (\delta,\delta_2)\restriction_{\xi_Q}$. By Lemma 7.2 (Dependency Agreement Closure) we have $\delta_1 = \delta_2$ and the desired equivalence, of the application of two apparently different semantic algebras (arising from the nondeterminism of the definition of the denotation of syntactic algebras) to the same value, follows.

Our well-formedness and index sorting rules do not track value-determined indices completely. We leave it to future work to enhance the completeness of syntactic well-formedness. This would likely involve strengthening the coupling of the SMT logic to our system.

Well-typed program terms Appendix Fig. A.51 specifies the denotations of well-typed refined program terms in terms of the denotations of their erasure. The denotation of a refined program term E typed under $(\Theta;\Gamma)$, at refined semantic substitution $\delta \in [\![\Theta;\Gamma]\!]$, is the denotation $[\![|E|]\!]_{|\delta|}$ of the (derivation of the) term's erasure $|E|$ at the erased substitution $|\delta|$. For example,

$$[\![\Theta;\Gamma \vdash e \Leftarrow N]\!] = (\delta \in [\![\Theta;\Gamma]\!]) \mapsto [\![|\Gamma| \vdash |e| \Leftarrow |N|]\!]_{|\delta|}$$

Unrolling Lemmas E.10 and E.9 imply denotation of a refined ADT can be equirecursively unrolled:

Lemma 7.3 (Semantic Unroll)**.** *(Lemma E.11)*

If $\Xi \vdash \{\nu : \mu F \mid \mathscr{M}(F)\}\ \mathsf{type}[\xi]$ *and* $\delta \in [\![\Xi]\!]$

then the set

$$\left\{V \in [\![F]\!]_\delta\,(\mu\,[\![F]\!]_\delta) \;\middle|\; [\![t]\!]_\delta\,([\![\alpha]\!]_\delta\,([\![F]\!]_\delta\,(\mathit{fold}_{[\![F]\!]_\delta}\,[\![\alpha]\!]_\delta)\,V)) = [\![t]\!]_\delta \text{ for all } (\!|\alpha|\!)_F\,\nu\,\mathbf{t} =_\tau t \in \mathscr{M}(F)\right\}$$

is equal to the set

$$\{V \in \mu\,[\![F]\!]_\delta \mid [\![\mathscr{M}(F)]\!]_\delta\,V = \{\bullet\}\}$$

We also prove (appendix Lemma E.20) that unrolling is sound:

Lemma 7.4 (Unrolling Soundness)**.** *(Lemma E.20)*

If $\Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \wr \triangleq {}^{\mathsf{d}}\Theta; R$ *and* $\vdash \delta : \Xi$

then the set of all semantic values $V \in [\![G]\!]_\delta\,(\mu\,[\![F]\!]_\delta)$ *such that*

$$\forall(\beta, (\!|\alpha|\!)_F\ \nu\, \boldsymbol{t} =_\tau t) \in zip(\overrightarrow{\beta})(\mathscr{M}(F)).\ [\![\boldsymbol{t}]\!]_\delta\,([\![\beta]\!]_\delta\,([\![G]\!]_\delta\,(fold_{[\![F]\!]_\delta}\ [\![\alpha]\!]_\delta)\ V)) = [\![t]\!]_\delta$$

is equal to the set $[\![\exists^{\mathsf{d}}\Theta.\, R \wedge {}^{\mathsf{d}}\Theta]\!]_\delta$.

We use the soundness of value-determined indices in $\wr\mathrm{Const}\wr$ and $\wr\mathrm{Id}\wr$ cases of the proof. The hardest case is $\wr\mathrm{Id}\wr$ because we have to prove the "hereditary substitution" or "liftapps" judgment used there preserves denotation. To this end, we define the denotation of the special temporary contexts used only by Id-unrolling, and implicitly use the fact that denotation of $\boldsymbol{u}$ is invariant under weakening its sorting derivation.

$\boxed{[\![\breve{\Xi}]\!]^{\mathsf{fix}}_\delta}$ Semantics for liftapps output contexts

Define $[\![\breve{\Xi}]\!]^{\mathsf{fix}}_\delta$ to be the fixed point of applying $[\![\breve{\Xi}]\!]_{\delta,_}$ initially to $[\![\breve{\Xi}]\!]_\delta$ where

$$[\![\cdot]\!]_\delta = \cdot$$
$$\left[\!\!\left[\breve{\Xi}, \breve{a}^{a(\boldsymbol{u})}\right]\!\!\right]_\delta = [\![\breve{\Xi}]\!]_\delta, ([\![\boldsymbol{u}]\!]_\delta\,\delta(a))\,/\breve{a}^{a(\boldsymbol{u})}$$

Lemma 7.5 (liftapps Sound)**.** *(Lemma E.19)*

If $\Xi; \Xi''; \overrightarrow{(a, (\mathsf{fold}_F\ \alpha)\,\nu\,_ =_\tau _)} \vdash \mathscr{O} \rightsquigarrow \breve{\Xi}; \mathscr{M}'(F); \mathscr{O}'$ *and* $\vdash \delta, \delta_2, \delta_1 : \Xi, \Xi'', \overrightarrow{a \dot{\div} \tau}$

and $dom(\delta_2) = dom(\Xi'')$ *and* $dom(\delta_1) = dom(\overrightarrow{a \dot{\div} \tau}) = \overrightarrow{a} = \pi_1\left(unzip\left(\overrightarrow{(a, (\mathsf{fold}_F\ \alpha)\,\nu\,_ =_\tau _)}\right)\right)$

then $FV(\mathscr{O}') \cap dom(\overrightarrow{a \dot{\div} \tau}) = \emptyset$ *and* $[\![\mathscr{O}]\!]_{\delta,\delta_2,\delta_1} = [\![\mathscr{O}']\!]_{\delta,\delta_2,[\![\breve{\Xi}]\!]^{\mathsf{fix}}_{\delta,\delta_2,\delta_1}}$;

moreover, if $V \in \mu\,[\![F]\!]_\delta$ *then:*

$d_k = (fold_{[\![F]\!]_\delta}\ [\![\alpha_k]\!]_\delta)V$ *for all* $(d_k/a_k, (\mathsf{fold}_F\ \alpha_k)\,\nu\,_ =_{\tau_k} _) \in zip(\delta_1)(\overrightarrow{(\mathsf{fold}_F\ \alpha)\,\nu\,_ =_\tau _})$

implies $[\![\mathscr{M}'(F)]\!]_{\delta,\delta_2,[\![\check{\Xi}]\!]^{\mathsf{fix}}_{\delta,\delta_2,\delta_1}}$ *V holds.*

These two unrolling lemmas (Lemma 7.4 and Lemma 7.3) are used to prove that refined typing is sound in the two cases for refined ADTs (value typing and pattern matching).

Subtyping We prove that subtyping/submeasuring is sound:

Lemma 7.6 (Subtyping Soundness)**.** *(Lemma E.23)*

(1) If $\Theta \vdash A \leq^{\pm} B$ *and* $\delta \in [\![\Theta]\!]$ *then* $[\![A]\!]_\delta \subseteq [\![B]\!]_\delta$.

(2) If $\Xi \vdash \alpha; F \leq_\tau \beta; G$ *and* $\delta \in [\![\Xi]\!]$ *then* $[\![F]\!]_\delta X \subseteq [\![G]\!]_\delta X$ *for any* $X \in$ **Set**.

(3) If $\Xi \vdash \alpha; F \leq_\tau \beta; G$ *and* $\delta \in [\![\Xi]\!]$ *then* $[\![\alpha]\!]_\delta = [\![\beta]\!]_\delta$ *on* $[\![F]\!]_\delta [\![\tau]\!]$.

(4) If $\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)$ *and* $\delta \in [\![\Theta]\!]$
then for all V we know $[\![\mathscr{M}'(F')]\!]_\delta V$ *implies* $[\![\mathscr{M}(F)]\!]_\delta V$.

Part (3) uses the soundness of value-determined indices.

It may be worthwhile in future to investigate increasing the semantic completeness of syntactic subtyping but in this thesis we try to keep subtyping as simple as possible.

Typing soundness Denotational-semantic typing soundness says that if a program term has type A under Θ and Γ, then the mathematical denotation of that program term at any interpretation of (that is, semantic environment for) Θ and Γ is an element of the mathematical denotation of A at that interpretation, that is, the program term denotes a dependent function $\prod_{\delta \in [\![\Theta;\Gamma]\!]} [\![A]\!]_{\lfloor \delta \rfloor}$. This more or less corresponds to proving (operational) typing soundness with respect to a big-step operational semantics. Refined types pick out subsets of values of unrefined types. Therefore, by typing soundness, if a program has a refined

type, then we have learned something more about that program than the unrefined system can verify for us.

Theorem 7.1 (Typing Soundness)**.** *(Thm. E.1)*

Assume $\vdash \delta : \Theta; \Gamma$. *Then:*

(1) If $\Theta; \Gamma \vdash h \Rightarrow P$ *then* $[\![h]\!]_\delta \in [\![P]\!]_{\lfloor\delta\rfloor}$.

(2) If $\Theta; \Gamma \vdash g \Rightarrow N$ *then* $[\![g]\!]_\delta \in [\![N]\!]_{\lfloor\delta\rfloor}$.

(3) If $\Theta; \Gamma \vdash v \Leftarrow P$ *then* $[\![v]\!]_\delta \in [\![P]\!]_{\lfloor\delta\rfloor}$.

(4) If $\Theta; \Gamma \vdash e \Leftarrow N$ *then* $[\![e]\!]_\delta \in [\![N]\!]_{\lfloor\delta\rfloor}$.

(5) If $\Theta; \Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$ *then* $[\![\{r_i \Rightarrow e_i\}_{i\in I}]\!]_\delta \in [\![P]\!]_{\lfloor\delta\rfloor} \Rightarrow [\![N]\!]_{\lfloor\delta\rfloor}$.

(6) If $\Theta; \Gamma; [N] \vdash s \Rightarrow \uparrow P$ *then* $[\![s]\!]_\delta \in [\![N]\!]_{\lfloor\delta\rfloor} \Rightarrow [\![\uparrow P]\!]_{\lfloor\delta\rfloor}$.

(All parts are mutually recursive.) The proof (appendix Thm. E.1) uses the soundness of unrolling and subtyping. The proof is mostly straightforward. The hardest case[3] is the one for recursive expressions in part (4), where we use an *upward closure* lemma—in particular, part (3) below—to show that the fixed point is in the appropriately refined set:

Lemma 7.7 (Upward Closure)**.** *(Lemma E.26)*

Assume $\vdash \delta : \Xi$.

(1) If $\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$ *then* $[\![\alpha]\!]_\delta$ *is monotone.*

(2) If $\Xi \vdash \mathscr{F}$ $\mathsf{functor}[_]$ *and* $\Xi \vdash F$ $\mathsf{functor}[_]$ *and* $k \in \mathbb{N}$
and $V \in [\![\mathscr{F}]\!]_\delta\,([\![F]\!]^k_\delta\,\emptyset)$ *and* $V \sqsubseteq_{[\![|\mathscr{F}|]\!]\,([\![|F|]\!]^k\emptyset)} V'$
then $V' \in [\![\mathscr{F}]\!]_\delta\,([\![F]\!]^k_\delta\,\emptyset)$.

[3] I don't know if this is interesting, but observe that the set $\cup_{k\in\mathbb{N}}\{X_k\}$ (where X_k is defined in the proof) is a filter on the unrefined set.

(3) If $\Xi \vdash A$ type$[_]$ *and* $V \in [\![A]\!]_\delta$ *and* $V \sqsubseteq_{[\![|A|]\!]} V'$ *then* $V' \in [\![A]\!]_\delta$.

The proof of upward closure uses one of the more interesting induction metrics:

Proof. By lexicographic induction, first, on the structure of A or F (parts (1), (2) and (3), mutually), and, second, on $\langle k, \mathscr{F}$ structure$\rangle$ (part (2)), where $\langle \dots \rangle$ denotes lexicographic order. □

This is also the only place, other than subtyping soundness and unrolling soundness, where we use the soundness of value-determined indexes (namely, for the pack case in part (1)).

Substitution soundness We interpret a syntactic substitution (typing derivation) $\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma$ as a function $[\![\sigma]\!] : [\![\Theta_0; \Gamma_0]\!] \to [\![\Theta; \Gamma]\!]$ on semantic substitutions. Or rather, we prove this the case: Lemma E.28 (Substitution Typing Soundness). The denotation of σ at δ is[4]

$$[\![\cdot]\!]_\delta = \cdot$$
$$[\![\sigma, t/a]\!]_\delta = [\![\sigma]\!]_\delta, [\![t]\!]_{\lfloor\delta\rfloor}/a$$
$$[\![\sigma, v : P/x]\!]_\delta = [\![\sigma]\!]_\delta, [\![v]\!]_\delta/x$$

For more details see appendix Def. C.2. Substitution soundness holds (appendix Thm. E.2): if E is a program term typed under Θ and Γ, and $\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma$, then $[\![[\sigma]E]\!] = [\![E]\!] \circ [\![\sigma]\!]$. (Recall we proved a syntactic substitution lemma: Lemma 6.10.) That is, substitution and denotation commute, or (in other words) syntactic substitution and semantic substitution are compatible.

[4]Economou et al. [2023] pre-applied the syntactic substitution prefix because there the substitution operation was sequential not parallel.

Logical consistency, total correctness, and partial correctness Our *semantic* typing soundness result implies that our system is logically consistent and totally correct.

A logically inconsistent type (for example, 0 or $\uparrow 0$ or $\uparrow(1 \wedge \mathsf{ff})$) denotes the empty set, which is uninhabited.

Corollary 7.1 (Logical Consistency)**.** *If* $\cdot;\cdot \vdash e \Leftarrow \uparrow P$*, then* $\uparrow P$ *is* logically consistent*, that is,* $[\![\uparrow P]\!]_{\cdot} \neq \emptyset$*.*

Total correctness means that every closed computation (that is specified as total) returns a value of the specified type:

Corollary 7.2 (Total Correctness)**.**

If $\cdot;\cdot \vdash e \Leftarrow \uparrow P$*, then* $[\![e]\!]_{\cdot} \neq \bot_{[\![\uparrow P]\!]_{\cdot}}$*, that is, e does not diverge semantically speaking.*

Proof.

$$\begin{aligned}
&\vdash \cdot : \cdot;\cdot && \text{By rule}\\
[\![e]\!]_{\cdot} \in [\![\uparrow P]\!]_{\lfloor\cdot\rfloor} &&\text{By Theorem 7.1 (Typing Soundness)}\\
= [\![\uparrow P]\!]_{\cdot} &&\text{By definition of } \lfloor -\rfloor\\
= \{inj_1\, V \mid V \in [\![P]\!]_{\cdot}\} &&\text{By definition of } [\![-]\!]_{\cdot}
\end{aligned}$$

Therefore, $[\![e]\!]_{\cdot} \neq inj_2\, \bot_{\uparrow} = \bot_{[\![\uparrow P]\!]_{\cdot}}$, □

However, in order for this result to imply that expressions deemed total by our refinement type system do not *operationally* or *computationally* diverge, or diverge syntactically speaking—that is, that the e above terminates (and returns a value)—then we must combine it with Theorem 5.3 (Computational Adequacy of the Denotational Semantics)[5] and the fact that erasure of type annotations (the partial inverse of bidirectionalization) preserves denotation.

[5] Which uses a Tait/Plotkin style logical relation.

Our system can be extended to include partiality, simply by adding a *partial* upshift type connective $\uparrow P$ ("partial upshift of P"), with type well-formedness and subtyping rules similar to those of $\uparrow P$, and the following two expression typechecking rules. The first rule introduces the new connective $\uparrow P$; the second rule lacks a termination refinement such as that in Decl$\Leftarrow$rec, so it may yield divergence.

$$\frac{\Theta;\Gamma \vdash v \Leftarrow P}{\Theta;\Gamma \vdash \mathtt{return}\, v \Leftarrow \uparrow P} \qquad \frac{\Theta;\Gamma, x : \downarrow N \vdash e \Leftarrow N}{\Theta;\Gamma \vdash \mathtt{rec}\, x.\, e \Leftarrow N}$$

The denotation of the partial upshift at $\delta \in [\![\Xi]\!]$ is defined as follows:

$$[\![\Xi \vdash \uparrow P\ \mathsf{type}[_]]\!]_\delta = \{d \in [\![\uparrow |P|]\!] \mid \text{if } d \neq \bot \text{ then } d = inj_1\, V \text{ for some } V \in [\![P]\!]_\delta\}$$

It is straightforward to update the metatheory to prove *partial correctness*: If a closed computation (that is specified as partial) terminates, then it returns a value of the specified type. Partial correctness is a corollary of the updated typing soundness result:
if $\cdot;\cdot \vdash e \Leftarrow \uparrow P$ and $[\![e]\!]_. \neq \bot$ then $[\![e]\!]_. = inj_1\, V$ and $V \in [\![P]\!]_.$.

Chapter 8

Algorithmic System

Dunfield and Krishnaswami [2013] prove the completeness of a bidirectional type system for higher-rank polymorphism. Dunfield and Krishnaswami [2019] extended this line of work to a form of indexed types much less general than the one presented in this thesis, although the latter lacks polymorphism. We design our algorithmic system in the spirit of those of Dunfield and Krishnaswami [2013, 2019]: unification/existential variables are created and solved in an algorithmic context, which we say extends with each solution added. Its judgments and presuppositions are summarized in Fig. 8.1 but the last two judgments are only used in metatheory. Mutually defined judgments are grouped together (separated by new lines).

The algorithmic rules closely mirror the declarative rules, except a few key differences:

- Whenever a declarative rule conjures an index term, the corresponding algorithmic rule adds to the input algorithmic context $\hat{\Theta}$ an existential variable (written with a hat: $\hat{a}$; also called evars) to be solved. An algorithmic context $\hat{\Theta}$ is a logical context Θ (which has universal variables or uvars) followed by evars and their solutions.
  - We delay solving evars until all constraints are collected. We solve evars by

$\hat{\Theta}$ algctx	(Fig. A.52)	pre.	no judgment
$\hat{\Xi} \rhd t : \tau\ [\xi_t]$	(Fig. A.54)	pre.	$\hat{\Xi}$ algctx
$\hat{\Xi}; [\tau] \rhd \boldsymbol{t} : \kappa$	(Fig. A.55)	pre.	$\hat{\Xi}$ algctx
$\hat{\Xi} \rhd A$ type$[\xi]$	(Fig. A.56)	pre.	$\hat{\Xi}$ algctx
$\hat{\Xi} \rhd \mathscr{M}(F)$ msmts$[\xi]$	(Fig. A.56)	pre.	$\hat{\Xi}$ algctx
$\hat{\Xi} \rhd \mathscr{F}$ functor$[\xi]$	(Fig. A.57)	pre.	$\hat{\Xi}$ algctx
$\hat{\Xi} \rhd \alpha : F(\tau) \Rightarrow \tau$	(Fig. A.57)	pre.	$\hat{\Xi} \rhd F$ functor$[\xi_F]$
$\hat{\Xi} \vdash {}^{(\forall)}W$ wf$[\xi]$	(Fig. A.58)	pre.	$\hat{\Xi}$ algctx
$\hat{\Xi} \vdash \chi$ Wf$[\xi]$	(Fig. A.58)	pre.	$\hat{\Xi}$ algctx
$\hat{\Theta} \vdash W$ Inst$\blacktriangleright \dashv \hat{\Theta}'$	(Fig. A.61)	pre.	$\hat{\Theta}$ algctx and $\overline{\hat{\Theta}} \vdash W$ wf
$\hat{\Theta} \vdash {}^{(\forall)}W$ Inst $\dashv \hat{\Theta}'$	(Fig. A.61)	pre.	$\hat{\Theta}$ algctx and $\overline{\hat{\Theta}} \vdash {}^{(\forall)}W$ wf
$\hat{\Theta} \vdash W$ fixInst $\dashv \hat{\Theta}'$	(Fig. A.61)	pre.	$\hat{\Theta}$ algctx and $\overline{\hat{\Theta}} \vdash W$ wf$[\xi]$
$\hat{\Theta}$ present	(Fig. A.53)	pre.	$\hat{\Theta}$ algctx
$\Theta \models {}^{(\forall)}W$	(Fig. A.59)	pre.	Θ ctx and $\overline{\Theta} \vdash {}^{(\forall)}W$ wf$[\xi]$
$\hat{\Theta}; \cdot \vdash W$ fixInstChk $\dashv \Omega$	(Fig. A.61)	pre.	$\hat{\Theta}$ ctx and $\overline{\hat{\Theta}} \vdash W$ wf$[\xi]$
$\Theta \vdash A <:^{\pm} B$	(Fig. A.62)	pre.	Θ ctx and $\overline{\Theta} \rhd A$ type$[\xi_A]$ and $\overline{\Theta} \rhd B$ type$[\xi_B]$
$\hat{\Theta} \vdash R <:^{+} P \,/\, {}^{(\forall)}W$	(Fig. A.62)	pre.	$\|\overline{\hat{\Theta}}\| \rhd R$ type$[\xi_R]$ and $\overline{\hat{\Theta}} \rhd P$ type$[\xi_P]$ and $\hat{\Theta}$ present
$\hat{\Theta} \vdash N <:^{-} L \,/\, {}^{(\forall)}W$	(Fig. A.62)	pre.	$\overline{\hat{\Theta}} \rhd N$ type$[\xi_N]$ and $\|\overline{\hat{\Theta}}\| \rhd L$ type$[\xi_L]$ and $\hat{\Theta}$ present
$\hat{\Theta} \vdash \mathscr{M}'(F') \geq \mathscr{M}(F) \,/\, W$	(Fig. A.62)	pre.	$\|\overline{\hat{\Theta}}\| \rhd \mathscr{M}'(F')$ msmts$[\xi']$ and $\overline{\hat{\Theta}} \rhd \mathscr{M}(F)$ msmts$[\xi]$ and $\hat{\Theta}$ present
$\Xi \rhd \alpha; F <:_{\tau} \beta; G$	(Fig. A.64)	pre.	$\Xi \rhd \alpha : F(\tau) \Rightarrow \tau$ and $\Xi \rhd \beta : G(\tau) \Rightarrow \tau$
$\hat{\Xi} \rhd \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Theta; R$	(Fig. A.65)	pre.	$\hat{\Xi} \rhd \mathscr{M}(F)$ msmts$[\xi]$ and $\hat{\Xi} \rhd \overrightarrow{\beta} : G(\mathscr{M}(F)) \Rightarrow \mathscr{M}(F)$ and $\hat{\Xi}$ present
$\Theta; \Gamma \lhd \chi$	(Fig. A.60)	pre.	$\overline{\Theta} \vdash \chi$ Wf$[\xi]$ and Θ algctx and $\overline{\Theta} \vdash \Gamma$ ctx
$\hat{\Theta}; \Gamma \vdash \chi$ fixInstChk $\dashv \Omega$	(Fig. A.61)	pre.	$\overline{\hat{\Theta}} \vdash \chi$ Wf$[\xi]$ and $\hat{\Theta}$ algctx and $\|\overline{\hat{\Theta}}\| \vdash \Gamma$ ctx
$\Theta; \Gamma \rhd h \Rightarrow P$	(Fig. A.66)	pre.	Θ algctx and $\overline{\Theta} \vdash \Gamma$ ctx
$\Theta; \Gamma \rhd g \Rightarrow \uparrow P$	(Fig. A.66)	pre.	Θ algctx and $\overline{\Theta} \vdash \Gamma$ ctx
$\hat{\Theta}; \Gamma \vdash v \Leftarrow P \,/\, \chi \dashv \Delta$	(Fig. A.67)	pre.	$\|\overline{\hat{\Theta}}\| \vdash \Gamma$ ctx and $\overline{\hat{\Theta}} \rhd P$ type$[\xi_P]$ and $\hat{\Theta}$ present
$\Theta; \Gamma \rhd v \Leftarrow P$	(Fig. A.67)	pre.	Θ algctx and $\overline{\Theta} \vdash \Gamma$ ctx and $\overline{\Theta} \rhd P$ type$[\xi_P]$
$\Theta; \Gamma \rhd e \Leftarrow N$	(Fig. A.68)	pre.	Θ algctx and $\overline{\Theta} \vdash \Gamma$ ctx and $\overline{\Theta} \vdash N$ type$[\xi_N]$
$\Theta; \Gamma; [P] \rhd \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$	(Fig. A.69)	pre.	Θ algctx and $\overline{\Theta} \vdash \Gamma$ ctx and $\overline{\Theta} \vdash P$ type$[\xi_P]$ and $\overline{\Theta} \vdash N$ type$[\xi_N]$
$\hat{\Theta}; \Gamma; [N] \vdash s \Rightarrow \uparrow P \,/\, \chi \dashv \Delta$	(Fig. A.70)	pre.	$\|\overline{\Theta}\| \vdash \Gamma$ ctx and $\overline{\hat{\Theta}} \rhd N$ type$[\xi_N]$ and $\hat{\Theta}$ present
$\Theta; \Gamma; [M] \rhd s \Rightarrow \uparrow P$	(Fig. A.70)	pre.	Θ algctx and $\overline{\Theta} \vdash \Gamma$ ctx and $\overline{\Theta} \rhd M$ type$[\xi_M]$
$\hat{\Theta} \longrightarrow \hat{\Theta}'$	(Fig. A.71)	pre.	$\hat{\Theta}$ algctx and $\hat{\Theta}'$ algctx
$\hat{\Theta} \xrightarrow{\mathsf{SMT}} \hat{\Theta}'$	(Fig. A.72)	pre.	$\hat{\Theta}$ algctx and $\hat{\Theta}'$ algctx

Every judgment except submeasuring, unrolling, and algebra well-formedness presupposes that no Id hypothesis occurs in its input context.

Figure 8.1: Algorithmic judgments and their presuppositions

simple matching on constraint equalities that should hold: if $\hat{a} = t$ should hold and the only evars occurring in t will be solved later (that is, they are marked with a $\blacktriangleright$ prefix in the algorithmic context), then we eagerly solve $\hat{a}$ as t. This is done repeatedly until a fixed point is reached, applying the solutions to the constraints. Then we solve the remaining evars, those we already knew could be solved at this point (those we previously said "will be solved later") such that all the constraints hold.

- Whenever a declarative rule checks a constraint ($\Theta \vdash \varphi$ true or ${}^{\mathsf{d}}\Xi \vdash t \equiv u : \tau$ or ${}^{\mathsf{d}}\Xi; [\tau] \vdash \boldsymbol{t} \equiv \boldsymbol{u} : \kappa$ or a sub/typing derivation or such), the algorithm delays its verification until all existential variables are solved (at the end of a focusing stage using the process described in the previous sub-bullet point). Subtyping and expression typechecking constraints are similarly delayed.

Syntactically, objects in the algorithmic system are not much different from corresponding objects of the declarative system. We extend the grammar for index terms with a production of existential variables, written as index variables with a hat $\hat{a}$, $\hat{b}$, or $\hat{c}$.

$$t \ ::= \ \cdots \mid \hat{a}$$

We use this (algorithmic) index grammar everywhere in the algorithmic grammar and system, using the same declarative metavariables.

When a piece of syntax has no existential variables, we say that it is *ground*. The judgment $\mathscr{O}$ ground is defined by $FEV(\mathscr{O}) = \emptyset$ where $FEV(\mathscr{O})$ is defined by structural recursion on $\mathscr{O}$ to collect every free evar of $\mathscr{O}$ in a set. If $FV(\xi) \subseteq dom(\hat{\Theta})$ then we will often get the algorithmic part of ξ by subtracting the logical part: $\xi - \|\hat{\Theta}\|$ (recall $\|-\|$

removes evars or gets the logical part).

Constraints are added to the algorithmic system. Figure 8.2 gives grammars for subtyping and typing constraints. In contrast to DML, the grammar does not include existential constraints. The disjunction constraints are concerning but we need only consider certain forms of it due to invariants of polarized subtyping. While we call all the W constraints "subtyping constraints", we call $\underline{A <:^{\pm} B}$ *literal* subtyping constraints.

$$
\begin{array}{lrcl}
\text{Subtyping constraints} & {}^{(\forall)}W & ::= & {}^{(\supset)}W \mid \forall a \overset{\mathsf{d}}{\div} \tau.\, {}^{(\forall)}W \\
& {}^{(\supset)}W & ::= & W \mid \varphi \supset {}^{(\supset)}W \\
& W & ::= & \varphi \mid u \equiv_{\tau} t \mid \boldsymbol{u} \equiv_{[\tau]} \boldsymbol{t} \mid \bigvee \overrightarrow{W} \\
& & & \mid \underline{R <:^{+} P} \mid \underline{N <:^{-} L} \\
& & & \mid {}^{(\forall)}W \wedge {}^{(\forall)}W \\
\text{Typing constraints} & \chi & ::= & \cdot \mid (e \Leftarrow N), \chi \mid W, \chi
\end{array}
$$

Figure 8.2: Typing and subtyping constraints

Checking constraints boils down to checking propositional validity, $\Theta \vdash \varphi\ \mathsf{true}$, which is analogous to checking *verification conditions* in the tradition of imperative program verification initiated by Floyd [1967] and Hoare [1969] (where programs annotated with Floyd–Hoare assertions are analyzed, generating verification conditions whose validity implies program correctness). These propositional validity constraints are the constraints that can be automatically verified by a theorem prover such as an SMT solver. The (algorithmic) subtyping constraint verification judgment is written $\Theta \models {}^{(\forall)}W$ and means that ${}^{(\forall)}W$ algorithmically holds under Θ. Notice that the only context in the judgment is Θ, which has no existential variables: this reflects the fact that we delay verifying ${}^{(\forall)}W$ until ${}^{(\forall)}W$ has no existential variables (in which case we say ${}^{(\forall)}W$ is *ground*). Similarly, $\Theta; \Gamma \lhd \chi$ is the (algorithmic) typing constraint verification judgment, checking that all of the constraints in χ algorithmically hold under Θ and Γ, and here χ is also ground (it is always possible to

make it so at the end of a focusing stage).

8.1 Algorithmic Contexts and Substitution

An algorithmic context $\hat{\Theta}$ is a logical context Θ followed by a list of solved or unsolved existential variables which may or may not marked as "solved later" with a $\blacktriangleright$ prefix. An algorithmic context is *complete*, and is written Ω, if all evars are solved. We often write an algorithmic context with an empty logical context and only unsolved evars as Δ, especially for the output contexts of algorithmic typing (focusing stages).

$$
\begin{array}{rcl}
\hat{\Theta} & ::= & \Theta \mid \hat{\Theta}, \hat{a} \stackrel{\mathsf{d}\cdot}{\div} \kappa \mid \hat{\Theta}, \hat{a} : \tau{=}t \mid \hat{\Theta}, \blacktriangleright \hat{a} \stackrel{\mathsf{d}\cdot}{\div} \kappa \mid \hat{\Theta}, \blacktriangleright \hat{a} : \tau{=}t \\
{}^{\mathsf{d}}\hat{\Theta} & ::= & {}^{\mathsf{d}}\Theta \mid {}^{\mathsf{d}}\hat{\Theta}, \hat{a} \stackrel{\mathsf{d}\cdot}{\div} \kappa \mid {}^{\mathsf{d}}\hat{\Theta}, \hat{a} : \tau{=}t \mid {}^{\mathsf{d}}\hat{\Theta}, \blacktriangleright \hat{a} \stackrel{\mathsf{d}\cdot}{\div} \kappa \mid {}^{\mathsf{d}}\hat{\Theta}, \blacktriangleright \hat{a} : \tau{=}t \\
\hat{\Xi} & ::= & \Xi \mid \hat{\Xi}, \hat{a} \stackrel{\mathsf{d}\cdot}{\div} \kappa \mid \hat{\Xi}, \hat{a} : \tau{=}t \mid \hat{\Xi}, \blacktriangleright \hat{a} \stackrel{\mathsf{d}\cdot}{\div} \kappa \mid \hat{\Xi}, \blacktriangleright \hat{a} : \tau{=}t \\
{}^{\mathsf{d}}\hat{\Xi} & ::= & {}^{\mathsf{d}}\Xi \mid {}^{\mathsf{d}}\hat{\Xi}, \hat{a} \stackrel{\mathsf{d}\cdot}{\div} \kappa \mid {}^{\mathsf{d}}\hat{\Xi}, \hat{a} : \tau{=}t \mid {}^{\mathsf{d}}\hat{\Xi}, \blacktriangleright \hat{a} \stackrel{\mathsf{d}\cdot}{\div} \kappa \mid {}^{\mathsf{d}}\hat{\Xi}, \blacktriangleright \hat{a} : \tau{=}t \\
\Delta & ::= & \cdot \mid \Delta, \hat{a} \stackrel{\mathsf{d}\cdot}{\div} \kappa \mid \Delta, \blacktriangleright \hat{a} \stackrel{\mathsf{d}\cdot}{\div} \kappa \\
\Omega & ::= & \Theta \mid \Omega, \hat{a} : \tau{=}t \mid \Omega, \blacktriangleright \hat{a} : \tau{=}t
\end{array}
$$

We require solutions t of later evars $\blacktriangleright \hat{a}$ to be well-sorted under (input) logical contexts Θ, which have no existential variables. But later evars may occur in solutions to non-later evars. What makes this possible is the fact that algorithmic subtyping is polarized so that evars never occur in positive subtypes/submeasures or negative supertypes (the *ground sides* of subtyping); further, evars occur ephemerally in refinement algebras (that is, if an evar is introduced to an algebra, it is immediately solved).

We will often treat algorithmic contexts $\hat{\Theta}$ as substitutions of index terms for existential

variables $\hat{a}$ in index terms t (including propositions φ), types A, functors $\mathscr{F}$, algebras α, constraints W and χ, and output logical contexts ${}^{\mathsf{d}}\Theta$ (whose propositions may have existential variables). The definition is similar to the definition of index substitution in Fig. A.6. Since we allow solutions of non-later evars to have later evars, we often apply algorithmic contexts as substitutions twice. If $\mathscr{O}$ only has existential variables from $dom(\Omega)$ then $[\Omega]^2\mathscr{O}$ is ground. By a Barendregtian distribution lemma $[\Omega]^2\mathscr{O} = [[\Omega]\Omega]\mathscr{O}$ where $[\hat{\Theta}]\hat{\Theta}'$ applies $\hat{\Theta}$ as a substitution to solutions in $\hat{\Theta}'$.

8.2 Well-Formedness

Figure A.52 defines the algorithmic context well-formedness judgment $\hat{\Theta}\ \mathsf{algctx}$ in such a way that the order of later and non-later evars does not matter. It uses the operation $\|-\|$ (defined in Fig. A.52) which extracts the underlying logical context of an algorithmic context. We always begin with an algorithmic context $\hat{\Theta}$ in which no $\blacktriangleright$ occurs: this defines the judgment $\hat{\Theta}\,\mathsf{present}$. We only mark evars as $\blacktriangleright$ after collecting all the constraints χ of a focusing stage, based on the (output) dependencies ξ_χ of χ well-formedness: the operation $[\xi]-$ (defined in Fig. A.52) adds a $\blacktriangleright$ prefix to an evar if it occurs in $\mathfrak{D}$ for some $\blacktriangleright\mathfrak{D} \in \xi$ (explained in the next paragraph). Finally, $\hat{\Theta}\ \mathsf{algctx}$ also uses the $\blacktriangleright-$ operation (defined in Fig. A.52) to get the later portion of an algorithmic context. Figure A.53 defines operations $sol(-)$ and $unsol(-)$ on algorithmic contexts which gets the solved or unsolved portion of an algorithmic context. The notation $[\blacktriangleright]\mathscr{O}$ stands for either $\blacktriangleright\mathscr{O}$ or $\mathscr{O}$. Similarly, the notation $\mathscr{O}[{=}t]$ stands for either $\mathscr{O}{=}t$ or $\mathscr{O}$.

Algorithmic index sorting (Fig. A.54 and Fig. A.55) is the same as declarative index sorting except it presupposes a well-formed *algorithmic* context and it accommodates evars

(the notation $\lfloor \hat{a} \rfloor$ for example stands for either $\hat{a}$ or a). The same goes for algorithmic type-/functor/algebra/measurement well-formedness (Fig. A.56 and Fig. A.57). Figure A.58 defines algorithmic constraint well-formedness. The $\xi_{(\forall)W}$ of a subtyping constraint ${}^{(\forall)}W$ is the union of the ξ of each top-level (that is, not under $\supset$ or $\forall$) constraint that should hold. A *disjunction* constraint is only well-formed under specific conditions obtained by polarized subtyping (discussed later):

$$\frac{\begin{array}{l}\text{there exist } \boldsymbol{t}, \tau, t, \kappa, \xi \text{ such that } t \rightsquigarrow \xi \text{ and } \hat{\Xi}; [\tau] \rhd \boldsymbol{t} : \kappa \\ \text{and for all } W_k \in \overrightarrow{W} (\neq \cdot) \text{ we have } \overset{\mathrm{d}}{\div} \hat{\Xi} \vdash W_k \ \mathsf{wf}[_] \\ \quad \text{and there exist } u_k, \boldsymbol{u}_k \text{ such that } W_k = (\boldsymbol{u}_k \equiv_{[\tau]} \boldsymbol{t} \wedge u_k \equiv_\kappa t) \text{ and } (u_k, \boldsymbol{u}_k)\,\mathsf{ground}\end{array}}{\hat{\Xi} \vdash \bigvee \overrightarrow{W} \ \mathsf{wf}[\xi]}$$

where (we again locally redefine the $\rightsquigarrow$ symbol)

$$\frac{t \text{ is not an evar}}{t \rightsquigarrow \cdot} \qquad \frac{t \text{ is an evar}}{t \rightsquigarrow \blacktriangleright\{t\}}$$

The ξ of typing constraint list χ is the union of the ξ of each subtyping constraint in it. We overload the $\lfloor - \rfloor$ operation to translate a typing constraint to a subtyping constraint which is nothing but the conjunction of all the typing constraint's subtyping constraints (defined in Fig. A.58).

8.3 Verifying Constraints and Solving Existentials

Figures A.59 and A.60 define the verification of algorithmic subtyping and typing constraints under a *logical* context Θ. Only *ground* constraints are ever verified.

The judgment $\hat{\Theta};\Gamma \vdash \chi$ fixInstChk $\dashv \Omega$ (defined in Fig. A.61) is used on the constraints χ collected at the end of a focusing stage to solve all the evars and check all the constraints. It is defined mutually with program constraint verification and algorithmic program typing. (We have a similar judgment $\hat{\Theta};\cdot \vdash W$ fixInstChk $\dashv \Omega$ also defined in Fig. A.61 which is defined mutually with subtyping constraint verification and algorithmic subtyping; the following discussion similarly appiles to the subtyping version of the judgment.) The contexts $\hat{\Theta}$ and Γ are inputs where $\hat{\Theta}$ may mention evars newly generated by unrolling and $\blacktriangleright$ evars have already been introduced: $\hat{\Theta} = [\xi_\chi]\hat{\Theta}$. The output context Ω completes $\hat{\Theta}$ such that $[\Omega]^2\chi$ holds under $\hat{\Theta}$ and Γ that is $\overline{\|\hat{\Theta}\|};\Gamma \lhd [\Omega]^2\chi$ (or $\overline{\|\hat{\Theta}\|} \models [\Omega]^2 W$ in the case of subtyping). The judgment $\hat{\Theta};\Gamma \vdash \chi$ fixInstChk $\dashv \Omega$ first solves all the non-later evars via $\hat{\Theta} \vdash \lfloor\chi\rfloor$ fixInst $\dashv \hat{\Theta}'$ whose rules are self-explanatory. Next, an oracle selects constraints $\overrightarrow{W_o}$ from disjunction constraints involving later evars such that using their solutions makes all the constraints hold: $\hat{\Theta}' \vdash \wedge\overrightarrow{W_o}$ Inst$\blacktriangleright$ $\dashv \Omega$ and $\|\hat{\Theta}\|;\Gamma \lhd [\Omega]^2\chi$. In practice, an oracle would not be used, but backtracking would have to be implemented. In this thesis, we try to keep the algorithm very abstract and simple in order to highlight the essentials.

$$\frac{\hat{\Theta} \vdash \lfloor\chi\rfloor \text{ fixInst} \dashv \hat{\Theta}' \qquad \text{select } \overrightarrow{W_o} \text{ from } [\hat{\Theta}']^2\lfloor\chi\rfloor \qquad \hat{\Theta}' \vdash \wedge\overrightarrow{W_o} \text{ Inst}\blacktriangleright \dashv \Omega \qquad \|\hat{\Theta}\|;\Gamma \lhd [\Omega]^2\chi}{\hat{\Theta};\Gamma \vdash \chi \text{ fixInstChk} \dashv \Omega}$$

8.4 Subtyping

Consider the rule(s) for verifying a literal subtyping constraint.

$$\frac{\Theta \vdash A <:^{\pm} B}{\Theta \models \underline{A <:^{\pm} B}}$$

By the grammar for subtyping constraints, this should be read as two rules, one positive and one negative.

$$\frac{\Theta \vdash R <:^{+} P}{\Theta \models \underline{R <:^{+} P}} \qquad \frac{\Theta \vdash N <:^{-} L}{\Theta \models \underline{N <:^{-} L}}$$

The relations $\Theta \vdash R <:^{+} P$ and $\Theta \vdash N <:^{-} L$ are defined mutually with $\Theta \models {}^{(\forall)}W$ and $\hat{\Theta};\cdot \vdash W$ fixInstChk $\dashv \Omega$ and $\hat{\Theta} \vdash R <:^{+} P \;/\; {}^{(\forall)}W$ and $\hat{\Theta} \vdash N <:^{-} L \;/\; {}^{(\forall)}W$ and $\hat{\Theta} \vdash \mathscr{M}'(F') \geq \mathscr{M}(F) \;/\; W$ and $\Xi \rhd \alpha; F <:_{\tau} \beta; G$.

Algorithmic subtyping $\Theta \vdash A <:^{\pm} B$ (Fig. A.62) says that, under logical context Θ, the type A is algorithmically a subtype of B. Each polarity of it has basically one rule. The positive one is

$$\frac{{}^{\mathsf{d}}\Xi \text{ may be } \cdot \qquad \Theta, \widehat{{}^{\mathsf{d}}\Xi} \vdash R <:^{+} [\widehat{{}^{\mathsf{d}}\Xi}/{}^{\mathsf{d}}\Xi]Q \;/\; W \qquad \overline{\Theta}, \widehat{{}^{\mathsf{d}}\Xi} \vdash W \text{ wf}[\xi_W] \qquad \Theta, [\xi_W]\widehat{{}^{\mathsf{d}}\Xi}; \cdot \vdash W \text{ fixInstChk} \dashv \Omega}{\Theta \vdash R <:^{+} \exists^{\mathsf{d}}\Xi.\, Q}$$

which can be read as two rules

$$\frac{\Theta \vdash R <:^{+} Q \;/\; W \qquad \Theta;\cdot \vdash W\ \mathsf{fixInstChk} \dashv \Theta}{\Theta \vdash R <:^{+} Q}$$

$$\frac{{}^{\mathsf{d}}\Xi \neq \cdot \qquad \Theta, \widehat{{}^{\mathsf{d}}\Xi} \vdash R <:^{+} [\widehat{{}^{\mathsf{d}}\Xi}/{}^{\mathsf{d}}\Xi]Q \;/\; W \qquad \overline{\Theta}, \widehat{{}^{\mathsf{d}}\Xi} \vdash W\ \mathsf{wf}[\xi_W] \qquad \Theta, [\xi_W]\widehat{{}^{\mathsf{d}}\Xi};\cdot \vdash W\ \mathsf{fixInstChk} \dashv \Omega}{\Theta \vdash R <:^{+} \exists^{\mathsf{d}}\Xi.\, Q}$$

The operation $\widehat{-}$ (pronounced "enhat") on value-determined index variable binding contexts adds a hat to each variable in the domain:

$$\widehat{\cdot} = \cdot$$

$$\widehat{{}^{\mathsf{d}}\Xi, a \overset{\mathsf{d}_\cdot}{\div} \tau} = \widehat{{}^{\mathsf{d}}\Xi}, \hat{a} \overset{\mathsf{d}_\cdot}{\div} \tau$$

The operation $[\xi]-$ was discussed in Sec. 8.2. Strictly speaking the former rule can simply use $\Theta \models W$ as there are no evars to solve, but I prefer the more efficient economy of presentation.

The judgments $\hat{\Theta} \vdash R <:^{+} Q \;/\; {}^{(\forall)}W$ and $\hat{\Theta} \vdash M <:^{-} L \;/\; {}^{(\forall)}W$ are used by the top level algorithmic subtyping judgments. $\hat{\Theta} \vdash A <:^{\pm} B \;/\; {}^{(\forall)}W$ says that, under algorithmic context $\hat{\Theta}$, the type A is algorithmically a subtype of B if and only if output constraint W holds algorithmically (at suitable solutions for $\hat{\Theta}$), In subtyping, the delayed output constraints W must remember their logical context via $\supset$ and $\forall$. For example, in checking that $\exists a : \mathbb{N}.\, \mathsf{Nat}(a) \wedge (a < 5)$ is a subtype of $\exists a : \mathbb{N}.\, \mathsf{Nat}(a) \wedge (a < 10)$, the output constraint W is $\forall a : \mathbb{N}.\, (a < 5) \supset (a < 10)$.

We don't present all algorithmic subtyping rules here (see appendix Figs. A.62, A.63,

and A.64), but only enough rules to discuss the key design issues.

In algorithmic subtyping, we maintain the invariant that positive subtypes/submeasures and negative supertypes are ground. The rules

$$\frac{^{\mathsf{d}}\Xi \neq \cdot \qquad \Theta, \widehat{^{\mathsf{d}}\Xi} \vdash R <:^{+} [\widehat{^{\mathsf{d}}\Xi}/^{\mathsf{d}}\Xi]Q \;/\; W \qquad \overline{\Theta}, \widehat{^{\mathsf{d}}\Xi} \vdash W\ \mathsf{wf}[\xi_W] \qquad \Theta, [\xi_W]\widehat{^{\mathsf{d}}\Xi}; \cdot \vdash W\ \mathsf{fixInstChk} \dashv \Omega}{\Theta \vdash R <:^{+} \exists^{\mathsf{d}}\Xi.\, Q}$$

$$\frac{^{\mathsf{d}}\Xi \neq \cdot \qquad \Theta, \widehat{^{\mathsf{d}}\Xi} \vdash [\widehat{^{\mathsf{d}}\Xi}/^{\mathsf{d}}\Xi]N <:^{-} L \;/\; W \qquad \overline{\Theta}, \widehat{^{\mathsf{d}}\Xi} \vdash W\ \mathsf{wf}[\xi_W] \qquad \Theta, [\xi_W]\widehat{^{\mathsf{d}}\Xi}; \cdot \vdash W\ \mathsf{fixInstChk} \dashv \Omega}{\Theta \vdash \forall^{\mathsf{d}}\Xi.\, M <:^{-} L}$$

$$\frac{\begin{array}{c} ^{\mathsf{d}}\Xi' \neq \cdot \\ \Xi, ^{\mathsf{d}}\Xi, \overrightarrow{\varphi}, \widehat{^{\mathsf{d}}\Xi'} \vdash R <:^{+} [\widehat{^{\mathsf{d}}\Xi'}/^{\mathsf{d}}\Xi']Q' \;/\; ^{(\forall)}W \\ \Xi, ^{\mathsf{d}}\Xi, \widehat{^{\mathsf{d}}\Xi'} \vdash ^{(\forall)}W\ \mathsf{wf}[\xi] \\ \Xi, ^{\mathsf{d}}\Xi, \overrightarrow{\varphi}, [\xi]\widehat{^{\mathsf{d}}\Xi'}; \cdot \vdash ^{(\forall)}W\ \mathsf{fixInstChk} \dashv \Omega \\ \Xi, ^{\mathsf{d}}\Xi \rhd q \Rightarrow t; \hat{P} <:_{\tau} q' \Rightarrow [\Omega]^2[\widehat{^{\mathsf{d}}\Xi'}/^{\mathsf{d}}\Xi']t'; \hat{P}' \end{array}}{\Xi \rhd (\mathtt{pk}(^{\mathsf{d}}\Xi, \top), q) \Rightarrow t; \underline{\exists^{\mathsf{d}}\Xi.\, R \wedge \overrightarrow{\varphi}} \otimes \hat{P} <:_{\tau} (\mathtt{pk}(^{\mathsf{d}}\Xi', \top), q') \Rightarrow t'; \underline{\exists^{\mathsf{d}}\Xi'.\, Q'} \otimes \hat{P}'}$$

are the only subtyping/submeasuring rules which add existential variables (to the side not necessarily ground) to be solved (whereas the declarative system conjures a solution). We have essentially already explained them in the foregoing discussion. The last one is the most subtle: we don't use the top level subtyping judgment here, because we need access to Ω to solve the evars packed in the superalgebra. Notice that once we apply all the solutions we forget about them and move on (evars in $dom(\widehat{^{\mathsf{d}}\Xi'})$ are ephemeral).

We delay subtyping premises that may generate new evars until we can solve all the

evars already at hand.

$$\frac{P_1 = \exists^{\mathsf{d}} \Xi_1.\, R_1 \wedge \overrightarrow{\varphi_1} \qquad P_2 = \exists^{\mathsf{d}} \Xi_2.\, R_2 \wedge \overrightarrow{\varphi_2}}{\hat{\Theta} \vdash P_1 + P_2 <:^+ P_1' + P_2' \,/\, (\forall^{\mathsf{d}} \Xi_1.\, \overrightarrow{\varphi_1} \supset \underline{R_1 <:^+ P_1'}) \wedge (\forall^{\mathsf{d}} \Xi_2.\, \overrightarrow{\varphi_2} \supset \underline{R_2 <:^+ P_2'})}$$

$$\frac{}{\hat{\Theta} \vdash \downarrow N <:^+ \downarrow(\forall^{\mathsf{d}} \Xi.\, \overrightarrow{\varphi} \supset L) \,/\, \forall^{\mathsf{d}} \Xi.\, \overrightarrow{\varphi} \supset \underline{N <:^- L}}$$

$$\frac{}{\hat{\Theta} \vdash \uparrow(\exists^{\mathsf{d}} \Xi.\, R \wedge \overrightarrow{\varphi}) <:^- \uparrow P \,/\, \forall^{\mathsf{d}} \Xi.\, \overrightarrow{\varphi} \supset \underline{R <:^+ P}}$$

The grammar for subtyping constraints (positive subtypes and negative supertypes must be simple) forces the eager extraction of index information. This works because it corresponds to eagerly applying invertible rules.

The rule

$$\frac{\hat{\Theta} \vdash \mathscr{M}'(F') \geq \mathscr{M}(F) \,/\, W}{\hat{\Theta} \vdash \left\{\nu : \mu F' \;\middle|\; \mathscr{M}'(F')\right\} <:^+ \{\nu : \mu F \mid \mathscr{M}(F)\} \,/\, W}$$

outputs the measurement coverage constraints, which are in turn generated as follows.

$$\frac{\cdot \rhd \underline{\mathsf{tt}}^{(F')}; F' <:_{\mathbb{B}} \underline{\mathsf{tt}}^{(F)}; F}{\hat{\Theta} \vdash \mathscr{M}'(F') \geq \cdot_F \,/\, \mathsf{tt}}$$

$$\frac{\hat{\Theta} \vdash \mathscr{M}'(F') \geq \mathscr{M}(F) \,/\, W \qquad \text{let } \overrightarrow{W} = \left\{ \boldsymbol{t}' \equiv_{[\tau]} \boldsymbol{t} \wedge t' \equiv_\kappa t \;\middle|\; \begin{array}{l} \cdot \rhd \alpha'; F' <:_\tau \alpha; F \text{ and } \hat{\Theta}; [\tau] \rhd \boldsymbol{t}' : \kappa \\ \text{for some } (\mathsf{fold}_{F'}\ \alpha')\, \nu\, \boldsymbol{t}' =_\tau t' \in \mathscr{M}'(F') \end{array} \right\}}{\hat{\Theta} \vdash \mathscr{M}'(F') \geq \mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\, \nu\, \boldsymbol{t} =_\tau t \,/\, W \wedge \left(\bigvee \overrightarrow{W}\right)}$$

For each measurement in the measurements being covered ($\mathscr{M}$), there must be at least one measurement (in $\mathscr{M}'$) covering it. The issue is that the indices $\boldsymbol{t}$ and t may have evars, so we collect all possibilities in a disjunction constraint and deal with these very last. (Note: $\overrightarrow{W}$ implies an order, but the order doesn't matter.) Any evars that can only be solved here will be solvable later when every other evar is solved, though backtracking may be needed to find a combination that works (we abstract away these details). That is because t' is ground due to our groundness invariant: positive subtypes are always ground, so the measurements of a refined ADT subtype are as well.

As can be seen in the measurement covering judgment, algorithmic submeasuring always begins under an empty context. Any evars generated in submeasuring (due to packed supermeasures) are ephemeral because immediately solved away. In this way we maintain the invariant that the context of algorithmic submeasuring is only ever logical. As such, we can eagerly check the index equivalence of algebra bodies as there are no evars in the indices.

8.5 Typing

We now discuss issues specific to algorithmic program typing (Figs. A.66, A.67, A.68, A.69, A.70 excluding the mutually defined typing constraint verification and evar solving judgments already discussed). We take a similar approach as taken in algorithmic subtyping, except we do not delay all typing problems in the same stage and instead collect evars newly generated by unrolling in an output context Δ. Typing problems of a different stage are delayed until all their evars are solved: Alg$\Leftarrow\downarrow$.

Exploiting polarity, we can restrict the flow of index information to the right- and left-focusing stages: $\hat{\Theta};\Gamma \vdash v \Leftarrow P \mid \chi \dashv \Delta'$ and $\hat{\Theta};\Gamma;[M] \vdash s \Rightarrow \uparrow P \mid \chi \dashv \Delta$ are algorithmic value and spine typing judgments, and have top level versions (similarly to subtyping) $\Theta;\Gamma \rhd v \Leftarrow P$ and $\Theta;\Gamma;[N] \rhd s \Rightarrow \uparrow P$. The input types of these non-top-level focusing judgments can have existential variables, and synthesize constraints and newly generated evars to be solved at the end of the focusing stage. However, the algorithmic versions of the other typing judgments do not; we judgmentally distinguish the latter by replacing the "$\vdash$" in the declarative judgments with "$\rhd$" (for example, $\Theta;\Gamma \rhd g \Rightarrow \uparrow P$). Delayed constraints are verified only and immediately after completing a focusing stage, when all their existential variables can be solved.

Consequently, the algorithmic typing judgments for heads, bound expressions, pattern matching, and expressions are essentially the same as their declarative versions, but use algorithmic judgments, in particular the top level algorithmic focusing judgments when possible. To be complete, the rule Alg$\Leftarrow$let must be allowed to backtrack to any type synthesized by the bound expression. For example,

if Γ is $f : \downarrow\forall a : \mathbb{N}.\, \mathsf{Nat}(a) \to \uparrow\mathsf{Nat}(a), y : \{v : \mathsf{Nat} \mid \mathsf{ix}\, v = 0, \mathsf{ix}\, v = 1\}$

then under Γ the expression $\mathtt{let}\, x = f(y); \mathtt{return}\, x$ checks against any type $\uparrow\mathsf{Nat}(n)$ such

that n is SMT-equivalent to 0 or 1. But the algorithm nondeterministically solves a as either 0 or 1.

Alg$\Leftarrow$Var calls (non-top-level) algorithmic subtyping and outputs the subtyping constraint. Algorithmic subtyping beings with a ground positive subtype but the supertype may not be ground. The positive subtype is always ground because types are only added to program contexts in non-focusing stages (for example, Alg$\Leftarrow\lambda$).

Chapter 9

Algorithmic Decidability, Soundness, Completeness

We prove that the algorithmic system (Chapter 8) is decidable, as well as sound and complete with respect to the declarative system (Chapter 6).

The concept of algorithmic context extension helps us prove algorithmic soundness and completeness.

9.1 Context Extension and Well-Formedness

We say that $\hat{\Theta}$ and $\hat{\Theta}'$ *agree on the sorts and later- or* $\blacktriangleright$*-status of evars* if $dom(\hat{\Theta}) = dom(\hat{\Theta}')$ and for all $\hat{a} \in dom(\hat{\Theta})$ we know $\hat{a}$ has sort κ in $\hat{\Theta}$ if and only if $\hat{a}$ has sort κ in $\hat{\Theta}'$ and $\hat{a}$ has a $\blacktriangleright$ prefix in $\hat{\Theta}$ if and only if $\hat{a}$ has a $\blacktriangleright$ prefix in $\hat{\Theta}'$.

The *algorithmic (context) extension* judgment $\hat{\Theta} \longrightarrow \hat{\Theta}'$ presupposes $\hat{\Theta}$ ctx and $\hat{\Theta}'$ ctx and says that $\hat{\Theta}$ and $\hat{\Theta}'$ have the same underlying logical context, that $\hat{\Theta}$ and $\hat{\Theta}'$ agree on the sorts and $\blacktriangleright$-status of evars, and that if an evar is solved in $\hat{\Theta}$ then it has exactly the

same solution in $\hat{\Theta}'$. That is, it is defined by the rule:

$$\frac{\begin{array}{c}\|\hat{\Theta}\| = \|\hat{\Theta}'\| \\ \hat{\Theta} \text{ and } \hat{\Theta}' \text{ agree on the sorts and } \blacktriangleright\text{-status of evars} \\ \text{if } [\blacktriangleright]\hat{a} : \kappa{=}t \in \hat{\Theta} \text{ then } [\blacktriangleright]\hat{a} : \kappa{=}t \in \hat{\Theta}'\end{array}}{\hat{\Theta} \longrightarrow \hat{\Theta}'}$$

The *SMT* or *relaxed (algorithmic) extension* judgment, used in proving algorithmic completeness (to express that the algorithm may solve different but SMT-equivalent existential indices as compared to the given declarative ones), is defined similarly but it allows solutions to be SMT-equivalent:

$$\frac{\begin{array}{c}\|\hat{\Theta}\| = \|\hat{\Theta}'\| \\ \hat{\Theta} \text{ and } \hat{\Theta}' \text{ agree on the sorts and } \blacktriangleright\text{-status of evars} \\ \text{if } [\blacktriangleright]\hat{a} : \kappa{=}t \in \hat{\Theta} \text{ then } [\blacktriangleright]\hat{a} : \kappa{=}t \in \hat{\Theta}' \text{ or } \exists([\blacktriangleright]\hat{a} : \kappa{=}t' \in \hat{\Theta}').\, \|\hat{\Theta}\| \vdash [\hat{\Theta}]t = [\hat{\Theta}']t' \ \mathsf{true}\end{array}}{\hat{\Theta} \xrightarrow{\mathsf{SMT}} \hat{\Theta}'}$$

Notice that only completed (hence ground) solutions are compared for SMT equivalence: $\|\hat{\Theta}\|$ is the logical context underlying $\hat{\Theta}$.

If $\hat{\Theta} \longrightarrow \Omega$ (or $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$) then Ω is said to *complete* $\hat{\Theta}$: it has the solutions of $\hat{\Theta}$ (or SMT-equivalent ones if relaxed), and also solutions to the unsolved evars of $\hat{\Theta}$.

It is straightforward to prove that extension is sound, reflexive, and transitive: Lemma F.4 (Extension Sound), Lemma F.5 (Ext. Reflexive), and Lemma F.6 (Ext. Transitive). Transitivity involving relaxed extension relies on the fact that SMT equivalence $\Theta \vdash - = - \ \mathsf{true}$ is an equivalence relation (hence transitive).

As discussed in the previous chapter, algorithmic sorting and formation rules (and unrolling) are set up as nothing but the corresponding declarative ones but accommodating evars and their solutions.

Just as declarative unrolling generates well-formed declarative types, algorithmic unrolling generates well-formed algorithmic types: Lemma F.3 (Alg. Unrolling Output WF). We must check that all the algorithmic judgments output appropriately well-formed objects, but this is straightforward and we will leave it implicit, not even bothering to write down all the statements. We give one of the most subtle examples (due to the output Δ'): if $\hat{\Theta};\Gamma \vdash v \Leftarrow P \mathbin{/} \chi \dashv \Delta'$ then $\overline{\hat{\Theta}},\Delta' \vdash \chi\ \mathsf{Wf}[\xi_\chi]$ for some unique ξ_χ. We often do not write the formation judgment and simply write ξ_χ for the (unique) "ξ of χ" (which by the way is always the same as $\xi_{\lfloor\chi\rfloor}$: removing the program constraints from a well-formed χ yields a well-formed W having the same ξ).

A solved evar can still be used to derive the sorting for that solved evar. But we can also apply an input algorithmic context as a substitution to the subject(s) of formation or sorting judgment, or one manipulating such judgments: unrolling. These are called "right-hand substitution" lemmas, as in applying the algorithmic context as a substitution to the inputs on the right-hand side of the turnstile: Lemma F.1 (Right-hand Subst) and Lemma F.2 (Right-hand Subst. (Unroll)). The input context of algorithmic sorting or formation or unrolling can be extended: Lemma F.7 (Ext. Weakening (Ixs.)), Lemma F.9 (Ext. Weakening (Types)), and Lemma F.10 (Ext. Weaken (Unroll)). These are the "extension weakening" lemmas. The proof uses the fact that a plain extension or relaxed extension without propositions can be restricted to its value-determined part: Lemma F.8 (Extension Restriction).

The instantiation judgments output a well-formed algorithmic context that extends the input algorithmic context (presupposed well-formed): Lemma F.11 (Inst. Extends). We call

them "extending judgments". The proof uses the reflexivity and transitivity of extension.

Right-hand substitution, extension weakening, and instantiation extension together imply that the last premise in the following rule satisfies its needed presuppositions:

$$\frac{\hat{\Theta} \vdash W\ \mathsf{Inst} \dashv \hat{\Theta}'' \qquad \hat{\Theta}'' \neq \hat{\Theta} \qquad \hat{\Theta}'' \vdash [\hat{\Theta}'']^2 W\ \mathsf{fixInst} \dashv \hat{\Theta}'}{\hat{\Theta} \vdash W\ \mathsf{fixInst} \dashv \hat{\Theta}'}$$

Right-hand substitution and extension weakening can be used to complete algorithmic sorting, well-formedness, and unrolling derivations into declarative ones: Lemma F.12 (Alg. to Decl. WF) and Lemma F.13 (Complete Unroll).

An extension is said to be *present* if its contexts are present (if one context is present, then both are by definition of extension). If a declarative unrolling is obtained by a complete present extension then we can undo the completion to obtain the corresponding algorithmic unrolling which outputs objects completing to the declarative ones: Lemma F.14 (Uncomplete Unrolling). This lemma is helpful in proving the inductive value case of algorithmic completeness.

Finally, if the context of algorithmic unrolling is already applied then applying the context to the outputs has no effect: Lemma F.15 (Unroll Applied).

9.2 Decidability

We have proved that all algorithmic judgments are decidable (appendix Sec. F.3). Algorithmic constraint verification $\Theta \models W$ and $\Theta; \Gamma \lhd \chi$ boils down to verifying propositional validity $\Theta \vdash \varphi\ \mathsf{true}$, which is known to be decidable [Barrett et al., 2009]. Besides that, our decidability proofs rely on fairly simple metrics for the various algorithmic judgments:

most of the time it is simply the structure of the (derivation of the) judgment. We show that, for each algorithmic rule, every premise is smaller than the conclusion, according to the metrics.

Previously [Economou et al., 2023], in order to prove decidability of typing, we used a simple size function and counted the number of subtyping constraints W in typing constraint lists χ. We had to prove lemmas stating that the constraints output by algorithmic judgments decrease in size according to the function. Now the proof is even simpler: the metric is simply the structure of the algorithmic derivation. What led to the simplification is the unification of constraint checking and typing (which are defined by mutual recursion) in the metatheory.

The *non*-checking instantiation judgments output unique contexts: Lemma F.20 (Inst. Succeeds). By contrast, the *checking* instantiation judgment fixInstChk solves the later evars nondeterministically.

9.3 Algorithmic Soundness

We show that the algorithmic system is sound with respect to the declarative system. Since the algorithmic system is designed to mimic the judgmental structure of the declarative system, soundness of the algorithmic system is largely straightforward to prove. Completeness is significantly harder.

Soundness of algorithmic subtyping says that algorithmic subtyping/submeasuring implies declarative subtyping/submeasuring. We decompose it into five mutually recursive parts: there are two parts for positive subtyping, one for submeasuring, and two for negative subtyping. The positive and negative parts are dual as usual so proofs of one are similar to proofs of the other. The top level parts are self-explanatory. The more complicated parts

say that if a subtyping algorithm solves indexes under which its own verification conditions hold, then subtyping holds declaratively under the same solutions. We begin with a (presupposed) present context which extends to Ω so it is not necessary to apply Ω twice for completion. It is convenient to assume that the context is already applied to inputs (positive subtypes and negative supertypes are presupposed ground).

Theorem 9.1 (Soundness of Algorithmic Subtyping)**.** *(Thm. F.3)*

(1) *If* $\hat{\Theta} \vdash R <:^{+} Q \mathbin{/}^{(\forall)} W$ *and* $\hat{\Theta} \longrightarrow \Omega$ *and* $\|\hat{\Theta}\| \models [\Omega]^{(\forall)} W$ *and* $R\,\mathsf{ground}$ *and* $\overline{\hat{\Theta}} \rhd Q\,\mathsf{type}[\xi]$ *and* $[\hat{\Theta}]Q = Q$ *and* $\hat{\Theta}\,\mathsf{present}$ *then* $\|\hat{\Theta}\| \vdash R \leq^{+} [\Omega]Q$.

(2) *If* $\Theta \vdash R <:^{+} P$ *then* $\Theta \vdash R \leq^{+} P$.

(3) *If* $\hat{\Theta} \vdash M <:^{-} L \mathbin{/}^{(\forall)} W$ *and* $\hat{\Theta} \longrightarrow \Omega$ *and* $\|\hat{\Theta}\| \models [\Omega]^{(\forall)} W$ *and* $\overline{\hat{\Theta}} \rhd L\,\mathsf{type}[\Xi]$ *and* $L\,\mathsf{ground}$ *and* $[\hat{\Theta}]M = M$ *and* $\hat{\Theta}\,\mathsf{present}$ *then* $\|\hat{\Theta}\| \vdash [\Omega]M \leq^{-} L$.

(4) *If* $\Theta \vdash N <:^{-} L$ *then* $\Theta \vdash N \leq^{-} L$.

(5) *If* $\Xi \rhd \alpha; F <:_{\tau} \beta; G$ *then* $\Xi \vdash \alpha; F \leq_{\tau} \beta; G$.

Previously [Economou et al., 2023] we proved the soundness of algorithmic subtyping by way of two intermediate (sound) subtyping systems: a declarative system that eagerly extracts under shifts, and a semideclarative system that also eagerly extracts under shifts, but outputs constraints W in the same way as algorithmic subtyping, to be checked by the semideclarative judgment $\Theta \mathrel{\widetilde{\models}} W$ (that we prove sound with respect to the algorithmic $\Theta \models W$). We no longer use an intermediate system to prove algorithmic soundness, but we do use a similar one to prove algorithmic completeness, discussed in the next section. But

since eager extraction is now baked into the refinement type syntax and subtyping rules, we only use one intermediate system (for algorithmic completeness).

The statement of algorithmic typing soundness is similar to that of algorithmic subtyping soundness. As a consequence of focusing, the soundness of head, bound expression, expression, and match typing can be stated relatively simply (without needing to mention evar solutions). The typing soundness of values and spines (which are dual) says that if Ω completes the algorithm's solutions such that the algorithm's constraints hold, then the typing of the value or spine holds declaratively with Ω's solutions.

Theorem 9.2 (Alg. Typing Sound)**.** *(Thm. F.4 in appendix)*

(1) *If* $\Theta;\Gamma \rhd h \Rightarrow P$ *then* $\Theta;\Gamma \vdash h \Rightarrow P$.

(2) *If* $\Theta;\Gamma \rhd g \Rightarrow \uparrow P$ *then* $\Theta;\Gamma \vdash g \Rightarrow \uparrow P$.

(3) *If* $\hat{\Theta};\Gamma \vdash v \Leftarrow P \mathbin{/} \chi \dashv \Delta$ *and* $\hat{\Theta},\Delta \longrightarrow \Omega$ *and* $\|\hat{\Theta}\|;\Gamma \lhd [\Omega]\chi$
and $\overline{\hat{\Theta}} \rhd P\ \mathsf{type}[\xi]$ *and* $[\hat{\Theta}]P = P$ *and* $\hat{\Theta}\ \mathsf{present}$
then $\|\hat{\Theta}\|;\Gamma \vdash [\Omega]v \Leftarrow [\Omega]P$.

(4) *If* $\Theta;\Gamma \rhd v \Leftarrow P$ *then* $\Theta;\Gamma \vdash v \Leftarrow P$.

(5) *If* $\Theta;\Gamma \rhd e \Leftarrow N$ *then* $\Theta;\Gamma \vdash e \Leftarrow N$.

(6) *If* $\Theta;\Gamma;[P] \rhd \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$ *then* $\Theta;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$.

(7) *If* $\hat{\Theta};\Gamma;[M] \vdash s \Rightarrow \uparrow P \mathbin{/} \chi \dashv \Delta$ *and* $\hat{\Theta},\Delta \longrightarrow \Omega$ *and* $\|\hat{\Theta}\|;\Gamma \lhd [\Omega]\chi$
and $\overline{\hat{\Theta}} \rhd M\ \mathsf{type}[\xi]$ *and* $[\hat{\Theta}]M = M$ *and* $\hat{\Theta}\ \mathsf{present}$
then $\|\hat{\Theta}\|;\Gamma;[[\Omega]M] \vdash [\Omega]s \Rightarrow [\Omega]\uparrow P$.

(8) *If* $\Theta;\Gamma;[N] \rhd s \Rightarrow \uparrow P$ *then* $\Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P$.

9.4 Algorithmic Completeness

We show that the algorithmic system is complete with respect to the declarative system.

It is significantly more challenging to prove algorithmic *completeness*. The main challenge is to prove the algorithm solves every existential variable to an index SMT-equivalent to the declaratively given (conjured) index.

The declarative system can conjure index solutions that are different from the algorithm's solutions, but they must be equal according to the logical context $\|\hat{\Theta}\| = \Theta$. We capture this with *relaxed* context extension $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \hat{\Theta}'$ similar to (non-relaxed) context extension $\hat{\Theta} \longrightarrow \hat{\Theta}'$ but solutions in $\hat{\Theta}$ may change to SMT-equal (under Θ) solutions in $\hat{\Theta}'$. We discussed relaxed extension in Sec. 9.1.

We basically take the same approach in proving algorithmic *subtyping* completeness and algorithmic *typing* completeness (the only difference is handling the fact that typechecking rolled values $\texttt{into}(v)$ generates new existentials Δ). In the statement of completeness, given declarative existential solutions (a given complete context Ω) presuppose their algorithmic value-determinedness (based on dependencies involving only evars, no uvars, usually obtained by subtracting the logical part of the algorithmic context: $\xi - \|\hat{\Theta}\|$). We transport the corresponding algorithmic value-determined dependencies over to those of the output constraints in such a way that we can use this information (together with the given (semi)declarative constraint verification) to prove running fixInstChk succeeds in a way SMT-compatible with the declarative solutions (it outputs algorithmic solutions Ω relaxedly extending to the given declarative solutions Ω). Applying the algorithmic solutions instead of the declarative ones does not change the structure or height of typing or subtyping derivations or the structure of types or output constraints that judgmental equivalence cares about. We exploit this fact in the proofs: the induction metric is on the sum of the

height of the given (sub)typing derivation and the height of its output constraint checking derivation (if such exists), and we sometimes need to use the induction hypothesis at SMT-transported (sub)typing or constraint verification derivations.

9.4.1 Semideclarative System

We need to SMT-transport typing derivations but logically we shouldn't have to simultaneously think about the algorithm. It would make more sense to transport declarative derivations instead. But we also need to transport the output constraints of algorithmic typing, so we use a so-called *semideclarative* system, intermediate between the declarative and algorithmic systems. The semideclarative systems are the same as the algorithmic ones except the only difference is they conjure indices just like the declarative systems (why call it semideclarative rather than semialgorithmic? happenstance). We summarize the semideclarative judgments and their presuppositions in Fig. 9.1. As per usual, in (roughly the bottom half of) the figure, we also include judgments used in the intermediate/semideclarative metatheory: namely, judgmental equivalence and semideclarative fixInstChk.

It is straightforward to prove semideclarative (sub)typing is equivalent to declarative (sub)typing.

Lemma 9.1 (Semidecl. Sub. Sound)**.** *(Lemma G.3)*

(1) If $\Theta \mathrel{\widetilde{\vdash}} R <:^{+} P \mathbin{/} {}^{(\forall)}W$ *and* $\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W$ *then* $\Theta \vdash R \leq^{+} P$.

(2) If $\Theta \mathrel{\widetilde{\vdash}} N <:^{-} L \mathbin{/} {}^{(\forall)}W$ *and* $\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W$ *then* $\Theta \vdash N \leq^{-} L$.

(3) If $\Xi \mathrel{\widetilde{\vdash}} \alpha; F <:_{\tau} \alpha'; F'$ *then* $\Xi \vdash \alpha; F \leq_{\tau} \alpha'; F'$.

(4) If $\Theta \mathrel{\widetilde{\vdash}} \mathscr{M}(F) \geq \mathscr{M}'(F') \mathbin{/} W$ *and* $\Theta \mathrel{\widetilde{\models}} W$ *then* $\Theta \vdash \mathscr{M}(F) \geq \mathscr{M}'(F')$.

$\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W$	(Fig. A.80)	pre.	$\overline{\Theta} \vdash {}^{(\forall)}W$ wf$[\xi]$ and Θ ctx
$\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$	(Fig. A.80)	pre.	$\overline{\Theta} \vdash \chi$ Wf$[\xi]$ and $\overline{\Theta} \vdash \Gamma$ ctx and Θ ctx
$\Theta \mathrel{\widetilde{\vdash}} A <:^{\pm} B \mathbin{/} {}^{(\forall)}W$	(Fig. A.74)	pre.	$\overline{\Theta} \vdash A$ type$[\xi_A]$ and $\overline{\Theta} \vdash B$ type$[\xi_B]$ and Θ ctx
$\Theta \mathrel{\widetilde{\vdash}} \mathscr{M}'(F') \geq \mathscr{M}(F) \mathbin{/} W$	(Fig. A.73)	pre.	$\overline{\Theta} \vdash \mathscr{M}'(F')$ msmts$[\xi']$ and $\overline{\Theta} \vdash \mathscr{M}(F)$ msmts$[\xi]$ and Θ ctx
$\Xi \mathrel{\widetilde{\vdash}} \alpha;F <:_{\tau} \beta;G$	(Fig. A.75)	pre.	$\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$ and $\Xi \vdash \beta : G(\tau) \Rightarrow \tau$
$\Theta;\Gamma \mathrel{\widetilde{\vdash}} h \Rightarrow P$	(Fig. A.76)	pre.	Θ ctx and $\overline{\Theta} \vdash \Gamma$ ctx
$\Theta;\Gamma \mathrel{\widetilde{\vdash}} g \Rightarrow \uparrow P$	(Fig. A.76)	pre.	Θ ctx and $\overline{\Theta} \vdash \Gamma$ ctx
$\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow P \mathbin{/} \chi$	(Fig. A.77)	pre.	Θ ctx and $\overline{\Theta} \vdash \Gamma$ ctx and $\overline{\Theta} \vdash P$ type$[\xi_P]$
$\Theta;\Gamma \mathrel{\widetilde{\vdash}} e \Leftarrow N$	(Fig. A.78)	pre.	Θ ctx and $\overline{\Theta} \vdash \Gamma$ ctx and $\overline{\Theta} \vdash N$ type$[\xi_N]$
$\Theta;\Gamma;[P] \mathrel{\widetilde{\vdash}} \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$	(Fig. A.79)	pre.	Θ ctx and $\overline{\Theta} \vdash \Gamma$ ctx and $\overline{\Theta} \vdash P$ type$[\xi_P]$ and $\overline{\Theta} \vdash N$ type$[\xi_N]$
$\Theta;\Gamma;[N] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \mathbin{/} \chi$	(Fig. A.79)	pre.	Θ ctx and $\overline{\Theta} \vdash \Gamma$ ctx and $\overline{\Theta} \vdash N$ type$[\xi_N]$
$\Theta \vdash \mathscr{M}'(F) \equiv \mathscr{M}(F)$	(Fig. A.81)	pre.	$\overline{\Theta} \vdash \mathscr{M}'(F)$ msmts$[\xi]$ and $\overline{\Theta} \vdash \mathscr{M}(F)$ msmts$[\xi']$ and Θ ctx
$\Theta \vdash A \equiv^{\pm} B$	(Fig. A.81)	pre.	$\overline{\Theta} \vdash A$ type$[\xi_A]$ and $\overline{\Theta} \vdash B$ type$[\xi_B]$ and Θ ctx
$\Xi \vdash \alpha;F \equiv_{\tau} \beta;G$	(Fig. A.82)	pre.	$\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$ and $\Xi \vdash \beta : G(\tau) \Rightarrow \tau$
$\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W \leftrightarrow {}^{(\forall)}W'$	(Fig. A.83)	pre.	$\overline{\Theta} \vdash {}^{(\forall)}W$ wf$[\xi]$ and $\overline{\Theta} \vdash {}^{(\forall)}W'$ wf$[\xi']$ and Θ ctx
$\Theta \mathrel{\widetilde{\lhd}} \chi \leftrightarrow \chi'$	(Fig. A.84)	pre.	$\overline{\Theta} \vdash \chi$ Wf$[\xi]$ and $\overline{\Theta} \vdash \chi'$ Wf$[\xi']$ and Θ ctx
$\hat{\Theta};\cdot \mathrel{\widetilde{\vdash}} W$ fixInstChk $\dashv \Omega$	(Fig. A.85)	pre.	$\overline{\hat{\Theta}} \vdash W$ wf$[\xi]$ and $\hat{\Theta}$ algctx
$\hat{\Theta};\Gamma \mathrel{\widetilde{\vdash}} \chi$ fixInstChk $\dashv \Omega$	(Fig. A.85)	pre.	$\overline{\hat{\Theta}} \vdash \chi$ Wf$[\xi]$ and $\|\overline{\hat{\Theta}}\| \vdash \Gamma$ ctx and $\hat{\Theta}$ algctx

The presupposition judgments can be found in Fig. 6.1 and Fig. 8.1.

Figure 9.1: Semideclarative judgments and their presuppositions

Lemma 9.2 (Semidecl. Sub. Complete)**.** *(Lemma G.4)*

(1) If $\Theta \vdash R \leq^{+} P$ *then* $\Theta \mathrel{\widetilde{\vdash}} R <:^{+} P \mathbin{/} {}^{(\forall)}W$ *and* $\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W$.

(2) If $\Theta \vdash N \leq^{-} L$ *then* $\Theta \mathrel{\widetilde{\vdash}} N <:^{-} L \mathbin{/} {}^{(\forall)}W$ *and* $\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W$.

(3) If $\Xi \vdash \alpha;F \leq_{\tau} \beta;G$ *then* $\Xi \mathrel{\widetilde{\vdash}} \alpha;F <:_{\tau} \beta;G$.

(4) *If* $\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)$ *then* $\Theta \mathrel{\widetilde{\vdash}} \mathscr{M}'(F') \geq \mathscr{M}(F) \;/\; W$ *and* $\Theta \mathrel{\widetilde{\models}} W$.

Lemma 9.3 (Semidecl. Typing Sound)**.** *(Lemma G.6)*

(1) *If* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} h \Rightarrow P$ *then* $\Theta;\Gamma \vdash h \Rightarrow P$.

(2) *If* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} g \Rightarrow \uparrow P$ *then* $\Theta;\Gamma \vdash g \Rightarrow \uparrow P$.

(3) *If* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow P \;/\; \chi$ *and* $\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$ *then* $\Theta;\Gamma \vdash v \Leftarrow P$.

(4) *If* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} e \Leftarrow N$ *then* $\Theta;\Gamma \vdash e \Leftarrow N$.

(5) *If* $\Theta;\Gamma;[P] \mathrel{\widetilde{\vdash}} \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$ *then* $\Theta;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$.

(6) *If* $\Theta;\Gamma;[N] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \;/\; \chi$ *and* $\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$ *then* $\Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P$.

Lemma 9.4 (Semidecl. Typing Complete)**.** *(Lemma G.7)*

(1) *If* $\Theta;\Gamma \vdash h \Rightarrow P$ *then* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} h \Rightarrow P$.

(2) *If* $\Theta;\Gamma \vdash g \Rightarrow \uparrow P$ *then* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} g \Rightarrow \uparrow P$.

(3) *If* $\Theta;\Gamma \vdash v \Leftarrow P$ *then there exists* χ *such that* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow P \;/\; \chi$ *and* $\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$.

(4) *If* $\Theta;\Gamma \vdash e \Leftarrow N$ *then* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} e \Leftarrow N$.

(5) *If* $\Theta;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$ *then* $\Theta;\Gamma;[P] \mathrel{\widetilde{\vdash}} \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$.

(6) *If* $\Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P$
then there exists χ *such that* $\Theta;\Gamma;[N] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \;/\; \chi$ *and* $\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$.

When combined with the soundness of algorithmic (sub)typing we get the soundness of algorithmic constraint checking and fixInstChk. We state the typing version.

Lemma 9.5 (Alg. to Semidecl. Chk.)**.** *(Lemma G.8)*

(1) If $\Theta;\Gamma \lhd \chi$ *then* $\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$.

(2) If $\hat{\Theta};\Gamma \vdash \chi$ fixInstChk $\dashv \Omega'$ *then* $\hat{\Theta};\Gamma \mathrel{\widetilde{\vdash}} \chi$ fixInstChk $\dashv \Omega'$.

This algorithmic soundness proof allows us to prove the SMT-transport lemmas purely within the semideclarative setting. (I tried doing it in the algorithmic setting, and it seemed not to work.) The main one is stated as follows. Note the tilde over the turnstile indicating it is a semideclarative derivation. The context $\hat{\Theta}'$ has algorithmic solutions (none where the lemma is used) of the initial $\hat{\Theta}$ but also some given declarative ones. Since the algorithm does not use declarative solutions, we need to prove the solutions it does use are SMT-equivalent.

Lemma 9.6 (fixInstChk Unapply)**.** *(Lemma G.31)*

If $\hat{\Theta}$ present *and* $[\hat{\Theta}]\chi = \chi$

and $\overline{\hat{\Theta}} \vdash \chi$ Wf$[\xi]$ *and* $cl(\xi - \|[\xi]\hat{\Theta}\|)(\emptyset) = unsol([\xi]\hat{\Theta})$

and $\hat{\Theta} \longrightarrow \hat{\Theta}'$ *and* $[\xi]\hat{\Theta}';\Gamma \mathrel{\widetilde{\vdash}} [\hat{\Theta}']\chi$ fixInstChk $\dashv \Omega'$

then there exists a derivation $[\xi]\hat{\Theta};\Gamma \mathrel{\widetilde{\vdash}} \chi$ fixInstChk $\dashv \Omega$ *such that* $\Omega \xrightarrow{\mathsf{SMT}} \Omega'$.

We discuss a sketch of the proof which uses several lemmas. First, the initial run of fixInst (the first thing fixInstChk does) solves everything but the evars already known to be solvable later (those marked ▶ in context).

Lemma 9.7 (Only Evars Solved Later Remain Unsolved)**.** *(Lemma G.29)*

If $\overline{\hat{\Theta}} \vdash W$ wf$[\xi]$ *and* $[\hat{\Theta}]^2 W = W$

and $cl(\xi - \|[\xi]\hat{\Theta}\|)(\emptyset) = unsol([\xi]\hat{\Theta})$ *and* $[\xi]\hat{\Theta} \vdash W$ fixInst $\dashv \hat{\Theta}'$

then $unsol(\hat{\Theta}') = \blacktriangleright unsol([\xi]\hat{\Theta})$.

Then we make a selection of later evar solutions (with their accompanying index spine equivalence constraints) that is SMT-compatible with the given solutions. We can then prove that all the solutions are SMT-equivalent: If W holds at Ω' and running fixInst on W solves all but the later evars and the later evar solutions of Ω are SMT-compatible with those of Ω' then all the evar solutions of Ω must be SMT-compatible with those of Ω'. The point is that fixInst only instantiates by simple matching on equalites that should hold, and SMT-equal solutions can be freely swapped (allowing us to deal with the later evars being SMT-equal) using part (1) of Lemma 9.10 (Equiv. Solutions), discussed a bit later. The main part of the following is part (2). In proving part (2) we use part (1).

Lemma 9.8 (True Inst. Preserves Relaxed Ext.)**.** *(Lemma G.30)*

Assume $\|\Omega\|, \blacktriangleright\Omega, \hat{\Theta} - \|\Omega\| - \blacktriangleright\Omega \xrightarrow{\mathsf{SMT}} \Omega'$.

(1) If $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} [\Omega']^2({}^{(\forall)}W)$ *and* $\hat{\Theta} \vdash {}^{(\forall)}W \ \mathsf{Inst} \dashv \hat{\Theta}''$
and $\hat{\Theta}'' \longrightarrow \Omega$ *and* $[\hat{\Theta}]({}^{(\forall)}W) = {}^{(\forall)}W$
then $\|\hat{\Theta}''\|, \blacktriangleright\Omega, \hat{\Theta}'' - \|\hat{\Theta}''\| - \blacktriangleright\hat{\Theta}'' \xrightarrow{\mathsf{SMT}} \Omega'$.

(2) If $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} [\Omega']^2 W$ *and* $\hat{\Theta} \vdash W \ \mathsf{fixInst} \dashv \hat{\Theta}_0$
and $\hat{\Theta}_0 \longrightarrow \Omega$ *and* $unsol(\hat{\Theta}_0) = \blacktriangleright unsol(\hat{\Theta})$ *and* $[\hat{\Theta}]^2 W = W$
then $\Omega \xrightarrow{\mathsf{SMT}} \Omega'$.

We then take this resulting SMT-compatibility $\Omega \xrightarrow{\mathsf{SMT}} \Omega'$ and transport the given constraint equivalence of Lemma 9.5 (Alg. to Semidecl. Chk) along it. This is a so-called sandwich lemma (we sandwich the new Ω between the old Ω' and the constraint) which uses judgmental equivalence so let's talk about that.

9.4.2 Judgmental Equivalence

Basically, two types/functors/algebras/measurements/constraints are judgmentally equivalent if their respective, structural subparts are equivalent and the other (index) parts are SMT-equivalent. Refined inductive types are equivalent only if they use syntactically the same algebras and functors (up to alpha renaming).

All equivalence judgments satisfy weakening, reflexivity (Lemmas G.11, G.14, and G.17), symmetry (Lemmas G.12, G.15, and G.18), and transitivity (Lemmas G.13, G.16, and G.19).

Judgmental equivalence is stable under substitution: Lemma G.10 (Tp./Meas. Equiv. Syn. Subs). We only checked this for types/functors/measures as we only needed it there, but constraint equivalence is probably also stable under substitution.

We define a straightforward subtyping constraint equivalence judgment $\Theta \mathrel{\widetilde{\models}} W \leftrightarrow W'$, which uses the declarative index equivalence introduced in Chapter 6, as well as judgmental type equivalence, to transport semideclarative constraint verification derivations: if $\Theta \mathrel{\widetilde{\models}} W$ and $\Theta \mathrel{\widetilde{\models}} W \leftrightarrow W'$, then $\Theta \mathrel{\widetilde{\models}} W'$ by a derivation of equal structure/height (see appendix Lemma G.22). To simplify[1] the proof of this, we prove that type equivalence implies subtyping (Lemma G.20). To prove that, we use the fact that if Θ_1 is logically equivalent to Θ_2 under their prefix context Θ (judgment $\Theta \vdash \Theta_1 \equiv \Theta_2$) then we can replace Θ_1 with Θ_2 (and vice versa) in derivations (Lemma C.71). Conversely, mutual subtyping does not imply type equivalence: $\vdash 1 \wedge \mathtt{tt} \leq 1$ and $\vdash 1 \leq 1 \wedge \mathtt{tt}$ but $\not\vdash 1 \equiv 1 \wedge \mathtt{tt}$ because the unit type is structurally (at the level of types) distinct from an asserting type.

We define a similar typing constraint equivalence judgment $\Theta \mathrel{\widetilde{\lhd}} \chi \leftrightarrow \chi'$ which uses subtyping constraint equivalence. We prove a similar transport lemma for typing.

[1] However, this doesn't save us much work because we still need to prove Lemma G.23 (Subtyping Respects Equiv).

Lemma 9.9 (Typing Respects Equiv.). *(Lemma G.26)*

Assume $\Theta \vdash \Gamma \equiv^{+} \Gamma'$.

(1) If $\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$ *and* $\Theta \mathrel{\widetilde{\lhd}} \chi \leftrightarrow \chi'$

then $\Theta;\Gamma' \mathrel{\widetilde{\lhd}} \chi'$ *by a derivation of equal height and structure.*

(2) If $\Theta;\Gamma \mathrel{\widetilde{\vdash}} h \Rightarrow P$ *then there exists* P' *such that* $\Theta \vdash P \equiv^{+} P'$

and $\Theta;\Gamma' \mathrel{\widetilde{\vdash}} h \Rightarrow P'$ *by a derivation of equal height and structure.*

(3) If $\Theta;\Gamma \mathrel{\widetilde{\vdash}} g \Rightarrow \uparrow P$ *then there exists* P' *such that* $\Theta \vdash \uparrow P \equiv^{-} \uparrow P'$

and $\Theta;\Gamma' \mathrel{\widetilde{\vdash}} g \Rightarrow \uparrow P'$ *by a derivation of equal height and structure.*

(4) If $\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow P \mathbin{/} \chi$ *and* $\Theta \vdash P \equiv^{+} P'$ *then there exists* χ' *such that* $\Theta \mathrel{\widetilde{\lhd}} \chi \leftrightarrow \chi'$

and $\Theta;\Gamma' \mathrel{\widetilde{\vdash}} v \Leftarrow P' \mathbin{/} \chi'$ *by a derivation of equal height and structure.*

(5) If $\Theta;\Gamma \mathrel{\widetilde{\vdash}} e \Leftarrow N$ *and* $\Theta \vdash N \equiv^{-} N'$

then $\Theta;\Gamma' \mathrel{\widetilde{\vdash}} e \Leftarrow M$ *by a derivation of equal height and structure.*

(6) If $\Theta;\Gamma;[P] \mathrel{\widetilde{\vdash}} \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$ *and* $\Theta \vdash P \equiv^{+} P'$ *and* $\Theta \vdash N \equiv^{-} N'$

then $\Theta;\Gamma';[P'] \mathrel{\widetilde{\vdash}} \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N'$

by a derivation of equal height and structure.

(7) If $\Theta;\Gamma;[N] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \mathbin{/} \chi$ *and* $\Theta \vdash N \equiv^{-} N'$

then there exist P' *and* χ' *such that* $\Theta \vdash \uparrow P \equiv^{-} \uparrow P'$ *and* $\Theta \mathrel{\widetilde{\lhd}} \chi \leftrightarrow \chi'$

and $\Theta;\Gamma';[N'] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P' \mathbin{/} \chi'$ *by a derivation of equal height and structure.*

The statement of the subtyping version would look similar to this one if we don't use the "equivalence implies subtyping" simplification.

Anyway, we can swap SMT-equal indices without affecting any structure registered by judgmental equivalence (or SMT equality).

Lemma 9.10 (Equiv. Solutions)**.** *(Lemma G.27)*

Assume $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$.

(1) If $\overline{\hat{\Theta}} \rhd t : \kappa$ *then* $\|\hat{\Theta}\| \vdash [\Omega]^2 t = [\Omega]^2[\hat{\Theta}]^2 t$ true.

(2) If $\overline{\hat{\Theta}} \rhd u : \tau$ *then* $\|\hat{\Theta}\| \vdash [\Omega]^2 u \equiv [\Omega]^2[\hat{\Theta}]^2 u : \tau$.

(3) If $\overline{\hat{\Theta}}; [\tau] \rhd \boldsymbol{t} : \kappa$ *then* $\|\hat{\Theta}\|; [\tau] \vdash [\Omega]^2 \boldsymbol{t} \equiv [\Omega]^2[\hat{\Theta}]^2 \boldsymbol{t} : \kappa$.

(4) If $\overline{\hat{\Theta}} \rhd A$ type$[_]$ *then* $\|\hat{\Theta}\| \vdash [\Omega]^2 A \equiv^{\pm} [\Omega]^2[\hat{\Theta}]^2 A$.

(5) If $\overline{\hat{\Theta}} \rhd \mathscr{M}(F)$ msmts$[_]$ *then* $\|\hat{\Theta}\| \vdash [\Omega]^2 \mathscr{M}([\Omega]^2 F) \equiv [\Omega]^2[\hat{\Theta}]^2 \mathscr{M}([\Omega]^2[\hat{\Theta}]^2 F)$.

(6) If $\overline{\hat{\Theta}} \rhd \alpha : F(\tau) \Rightarrow \tau$ *then* $\|\hat{\Theta}\| \vdash [\Omega]^2 \alpha; [\Omega]^2 F \equiv_\tau [\Omega]^2[\hat{\Theta}]^2 \alpha; [\Omega]^2[\hat{\Theta}]^2 F$.

(7) If $\overline{\hat{\Theta}} \vdash {}^{(\forall)}W$ wf *then* $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} [\Omega]^2({}^{(\forall)}W) \leftrightarrow [\Omega]^2[\hat{\Theta}]^2({}^{(\forall)}W)$.

(8) If $\overline{\hat{\Theta}} \vdash \chi$ Wf *then* $\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} [\Omega]^2 \chi \leftrightarrow [\Omega]^2[\hat{\Theta}]^2 \chi$.

Combined with the fact that (sub)typing respects judgmental equivalence, we get the so-called sandwich lemma for constraint checking.

Lemma 9.11 (Constraint Checking Sandwich)**.** *(Lemma G.28)*

(1) If $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$ *and* $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} [\Omega]^2({}^{(\forall)}W)$ *and* $\overline{\hat{\Theta}} \vdash {}^{(\forall)}W$ wf
then $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} [\Omega]^2[\hat{\Theta}]^2({}^{(\forall)}W)$ *by a derivation of equal height and structure.*

(2) If $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$ *and* $\|\hat{\Theta}\|; \Gamma \mathrel{\widetilde{\lhd}} [\Omega]^2 \chi$ *and* $\overline{\hat{\Theta}} \vdash \chi$ Wf
then $\|\hat{\Theta}\|; \Gamma \mathrel{\widetilde{\lhd}} [\Omega]^2[\hat{\Theta}]^2 \chi$ *by a derivation of equal height and structure.*

When $\hat{\Theta}$ is complete Ω' we get say $\|\hat{\Theta}\|; \Gamma \mathrel{\widetilde{\lhd}} [\Omega]^2[\Omega']^2 \chi$ which is nothing but $\|\hat{\Theta}\|; \Gamma \mathrel{\widetilde{\lhd}} [\Omega']^2 \chi$ since $FEV([\Omega']^2 \chi) = \emptyset$ where $FEV(-)$ gets the set of free evars.

9.4.3 Dependency Mediation

Dependency mediation (Fig. G.1) is an abstraction describing how algorithmic[2] input type dependencies transform into algorithmic output constraint dependencies. The dependency mediation judgment $\Delta \vdash \xi' \angle \xi$ presupposes that only evars occur in ξ' and ξ, and that no evar from Δ occurs in ξ', and makes an assertion as defined by exactly one rule:

$$\frac{dom(\Delta) \subseteq cl(\xi)(FV(\xi) - \Delta) \quad \text{for all } \widetilde{\mathfrak{D}} \to \hat{c} \in \xi' \text{ there exists } \widetilde{\mathfrak{B}} \to \hat{c} \in \xi \text{ such that } \widetilde{\mathfrak{B}} \subseteq \widetilde{\mathfrak{D}} \cup \Delta \text{ and } \widetilde{\mathfrak{B}} \cap \Delta \subseteq cl(\xi)(\widetilde{\mathfrak{D}})}{\Delta \vdash \xi' \angle \xi}$$

The first premise, a condition on *all* of Δ, says all the newly generated evars Δ are value-determined, assuming everything except Δ is. The second premise says that for every dependency in the input type there is a similar dependency in the output constraint except it may depend on *part* of Δ, but that part of Δ is value-determined, assuming the original dependencies are. The judgment $\Delta \vdash \xi' \angle \xi$ can be read "Δ mediates ξ' in ξ".

A basic consequence of dependency mediation is that whatever is value-determined according to ξ' (of input type) is also value-determined according to ξ (of output constraints).

Lemma 9.12 (Admissible Premise)**.** *(Lemma G.32)*

If $\Delta \vdash \xi' \angle \xi$ *then* $cl(\xi')(\emptyset) \subseteq cl(\xi)(\emptyset)$.

Dependency mediation is also reflexive (under empty Δ) and composable: Lemma G.33 (No Δ Mediates Reflexive) and Lemma G.34 (Compose Mediates). Composability is useful in the product typechecking case, for example. Not only composability of dependency

[2] A dependency is *algorithmic* if it only mentions evars.

mediation but also fixInstChk: Lemma G.2 (Inst. Compose).

We state the lemma relating the dependencies of input types and output constraints of algorithmic (sub)typing as follows. The newly generated evars mediate the input type dependencies in the output constraint dependencies.

Lemma 9.13 (Main Complete)**.** *(Lemma G.35)*

(1) If $\hat{\Theta} \vdash R' <:^{+} Q \mathbin{/} W$ *and* $R'\,\mathsf{ground}$ *and* $\overline{\hat{\Theta}} \vdash Q\ \mathsf{type}[\xi_Q]$ *and* $[\hat{\Theta}]Q = Q$
then $\overline{\hat{\Theta}} \vdash W\ \mathsf{wf}[\xi_W]$ *and* $\cdot \vdash \xi_Q - \|\hat{\Theta}\| \angle \xi_W - \|\hat{\Theta}\|$;
moreover, if $Q = R$ *then* $\blacktriangleright(pos(\xi_Q - \|\hat{\Theta}\|)) \subseteq \xi_W - \|\hat{\Theta}\|$.

(2) If $\overline{\hat{\Theta}} \rhd \mathscr{M}(F)\ \mathsf{type}[\xi']$ *and* $\nexists x.\, v = \overrightarrow{\mathrm{inj}_{k_i}}^{i}\left(\overrightarrow{\langle __j, - \rangle}^{j} x\right)$
and $\overset{\mathsf{d}}{\div}\hat{\Theta} \rhd \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \wr \triangleq {}^{\mathsf{d}}\Theta; R$ *and* $\hat{\Theta}; \Gamma \vdash v \Leftarrow \exists^{\mathsf{d}}\Theta.\, R \wedge {}^{\mathsf{d}}\Theta \mathbin{/} \chi \dashv \Delta$
and $[\hat{\Theta}](F, \mathscr{M}, G, \overrightarrow{\beta}) = (F, \mathscr{M}, G, \beta)$
then $\overline{\hat{\Theta}}, \Delta \vdash \lfloor \chi \rfloor\ \mathsf{wf}[\xi_\chi]$ *and* $\Delta \vdash \xi' - \|\hat{\Theta}\| \angle \xi_\chi - \|\hat{\Theta}\|$.

(3) If $\hat{\Theta}; \Gamma \vdash v \Leftarrow Q \mathbin{/} \chi \dashv \Delta$ *and* $\overline{\hat{\Theta}} \rhd Q\ \mathsf{type}[\xi_Q]$ *and* $[\hat{\Theta}]Q = Q$
then $\overline{\hat{\Theta}}, \Delta \vdash \lfloor \chi \rfloor\ \mathsf{wf}[\xi_\chi]$ *and* $\Delta \vdash \xi_Q - \|\hat{\Theta}\| \angle \xi_\chi - \|\hat{\Theta}\|$.

(4) If $\hat{\Theta}; \Gamma; [M] \vdash s \Rightarrow \uparrow P \mathbin{/} \chi \dashv \Delta$
and $\overline{\hat{\Theta}} \rhd M\ \mathsf{type}[\xi_M]$ *and* $[\hat{\Theta}]M = M$
then $\overline{\hat{\Theta}}, \Delta \vdash \lfloor \chi \rfloor\ \mathsf{wf}[\xi_\chi]$ *and* $\Delta \vdash \xi_M - \|\hat{\Theta}\| \angle \xi_\chi - \|\hat{\Theta}\|$.

At the start of focusing stages (the "top level" judgments) we know all the existentials $FV(\xi')$ in dependencies are value-determined, we know newly generated evars Δ mediate input type dependencies in output constraint dependencies, and we know output constraints can only mention evars from the input type and Δ. As a consequence, we know that *according to the output constraint dependencies* all the original existentials $FV(\xi')$ as well as the new ones Δ are value-determined.

Lemma 9.14 (Det. to Mediated Det.)**.** *(Lemma G.37)*

If $\xi' \vdash FV(\xi')\ \mathsf{det}$ *and* $\Delta \vdash \xi' \angle \xi$ *and* $FV(\xi) \subseteq FV(\xi') \cup dom(\Delta)$

then $\xi \vdash FV(\xi'), \Delta\ \mathsf{det}$.

9.4.4 Proving Completeness

Algorithmic completeness says our subtyping algorithm verifies any declarative subtyping. We state the main lemma of which algorithmic subtyping completeness is a corollary (if one only cared about subtyping). This main lemma is used to prove algorithmic typing completeness which is the main result we care about.

Lemma 9.15 (Aux. Alg. Sub. Complete)**.** *(Lemma G.38)*

(1) *If* $\Theta \mathrel{\widetilde{\models}} W$ *then* $\Theta \models W$.

(2) *If* $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$ *and* $\|\hat{\Theta}\| \mathrel{\widetilde{\vdash}} R <:^{+} [\Omega]Q \,/\, W$ *and* $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W$
and $\hat{\Theta}\ \mathsf{present}$ *and* $[\hat{\Theta}]Q = Q$ *and* $R\,\mathsf{ground}$ *and* $\overline{\hat{\Theta}} \rhd Q\ \mathsf{type}[\xi]$
then there exists W' *such that* $\hat{\Theta} \vdash R <:^{+} Q \,/\, W'$
and $\|\hat{\Theta}\|; \cdot \vdash [\Omega]W'\ \mathsf{fixInstChk} \dashv \|\hat{\Theta}\|$ *and* $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W \leftrightarrow [\Omega]W'$.

(3) *If* $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$ *and* $\|\hat{\Theta}\| \mathrel{\widetilde{\vdash}} [\Omega]M <:^{-} L \,/\, W$ *and* $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W$
and $\hat{\Theta}\ \mathsf{present}$ *and* $[\hat{\Theta}]M = M$ *and* $L\,\mathsf{ground}$ *and* $\overline{\hat{\Theta}} \rhd M\ \mathsf{type}[\xi]$
then there exists W' *such that* $\hat{\Theta} \vdash L <:^{-} M \,/\, W'$
and $\|\hat{\Theta}\|; \cdot \vdash [\Omega]W'\ \mathsf{fixInstChk} \dashv \|\hat{\Theta}\|$ *and* $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W \leftrightarrow [\Omega]W'$.

(4) *If* $\Xi \mathrel{\widetilde{\vdash}} \alpha; F <:_{\tau} \alpha'; F'$ *then* $\Xi \rhd \alpha; F <:_{\tau} \alpha'; F'$.

(5) *If* $\Theta \mathrel{\widetilde{\vdash}} R <:^{+} P \,/\, W$ *and* $\Theta \mathrel{\widetilde{\models}} W$ *then* $\Theta \vdash R <:^{+} P$.

(6) *If* $\Theta \mathrel{\widetilde{\vdash}} N <:^{-} L \,/\, W$ *and* $\Theta \mathrel{\widetilde{\models}} W$ *then* $\Theta \vdash N <:^{-} L$.

(7) *If* $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$ *and* $\|\hat{\Theta}\| \mathrel{\widetilde{\vdash}} \mathscr{M}'(F') \geq [\Omega](\mathscr{M}(F))$ / W *and* $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W$
and $\hat{\Theta}$ present *and* $[\hat{\Theta}](\mathscr{M}(F)) = \mathscr{M}(F)$ *and* $\mathscr{M}'(F')$ ground *and* $\overline{\hat{\Theta}} \triangleright \mathscr{M}(F)$ msmts$[\xi]$
then there exists W' *such that* $\hat{\Theta} \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)$ / W'
and $\|\hat{\Theta}\|;\cdot \vdash [\Omega]W'$ fixInstChk $\dashv \|\hat{\Theta}\|$ *and* $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W \leftrightarrow [\Omega]W'$.

Finally, we prove the completeness of algorithmic typing. Like algorithmic typing soundness, again due to focusing, the head, bound expression, expression, and pattern matching parts are straightforward to state (and prove). But, because algorithmic function application may instantiate indexes different but logically equal to those conjured (semi)declaratively, bound expressions may algorithmically synthesize a type (judgmentally) equivalent to the type it synthesizes declaratively.

We introduced logical context equivalence in Sec. 9.4.2. Other than in proving that type equivalence implies subtyping, logical context equivalence is used in proving the completeness of algorithmic typing (in particular, we effectively use appendix Lemma C.71 to swap logically equivalent logical contexts in (semi)declarative typing derivations). The type $[\Omega, \Omega']^2 P'$ in the output of the algorithm in part (8) below can have different index solutions (from output Ω') that are logically equal (under Θ) to the solutions which appear in the declaratively synthesized P. However, P and P' necessarily have the same structure, so $\Theta \vdash P \equiv^+ [\Omega, \Omega']^2 P'$. Therefore, a bound expression (of a let-binding) may (semi)declaratively synthesize a type that is judgmentally equivalent to the type synthesized algorithmically: see part (3). We then extract different but logically equivalent logical contexts from the (equivalent) types synthesized by a bound expression. We replace (Lemma 9.9 (Typing Respects Equiv)) the given semideclarative subderivation of the body of the let-binding with an equivalent typing but at the algorithmic solutions, obtained by the induction hypothesis on the bound expression in the proof of the let-binding case of

part (6), before applying the induction hypothesis on the body, which is legal because the structure/height of the original does not change.

As such, the main algorithmic typing completeness lemma is stated as follows:

Lemma 9.16 (Aux. Alg. Typing Complete)**.** *(Lemma G.39)*

(1) *If* $\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$ *then* $\Theta;\Gamma \lhd \chi$.

(2) *If* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} h \Rightarrow P$ *then* $\Theta;\Gamma \rhd h \Rightarrow P$.

(3) *If* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} g \Rightarrow \uparrow P$

then there exists P' *such that* $\Theta;\Gamma \rhd g \Rightarrow \uparrow P'$ *and* $\Theta \vdash P \equiv^{+} P'$.

(4) *If* $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$ *and* $\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow [\Omega]P \mathbin{/} \chi$ *and* $\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\lhd}} \chi$

and $\hat{\Theta}\ \mathsf{present}$ *and* $[\hat{\Theta}]P = P$ *and* $\overline{\hat{\Theta}} \rhd P\ \mathsf{type}[\xi]$

then there exist χ', Δ', *and* Ω' *such that* $\hat{\Theta};\Gamma \vdash v \Leftarrow P \mathbin{/} \chi' \dashv \Delta'$

and $\overline{\hat{\Theta}}, \Delta' \vdash \chi'\ \mathsf{Wf}[\xi']$ *and* $\|\hat{\Theta}\|, [\xi']\Delta'; \Gamma \vdash [\Omega]\chi'\ \mathsf{fixInstChk} \dashv \|\hat{\Theta}\|, \Omega'$

and $\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} \chi \leftrightarrow [\Omega, \Omega']^2\chi'$.

(5) *If* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow P \mathbin{/} \chi$ *and* $\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$ *then* $\Theta;\Gamma \rhd v \Leftarrow P$.

(6) *If* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} e \Leftarrow N$ *then* $\Theta;\Gamma \rhd e \Leftarrow N$.

(7) *If* $\Theta;\Gamma;[P] \mathrel{\widetilde{\vdash}} \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$ *then* $\Theta;\Gamma;[P] \rhd \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$.

(8) *If* $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$ *and* $\|\hat{\Theta}\|;\Gamma;[[\Omega]M] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \mathbin{/} \chi$ *and* $\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\lhd}} \chi$

and $\hat{\Theta}\ \mathsf{present}$ *and* $\overline{\hat{\Theta}} \rhd M\ \mathsf{type}[\xi]$ *and* $[\hat{\Theta}]M = M$

then there exist P', χ', Δ', *and* Ω' *such that* $\hat{\Theta};\Gamma;[M] \vdash s \Rightarrow \uparrow P' \mathbin{/} \chi' \dashv \Delta'$

and $\overline{\hat{\Theta}}, \Delta' \vdash \chi'\ \mathsf{Wf}[\xi']$ *and* $\|\hat{\Theta}\|, [\xi']\Delta'; \Gamma \vdash [\Omega]\chi'\ \mathsf{fixInstChk} \dashv \|\hat{\Theta}\|, \Omega'$

and $\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} \chi \leftrightarrow [\Omega, \Omega']^2\chi'$ *and* $\|\hat{\Theta}\| \vdash P \equiv^{+} [\Omega, \Omega']^2 P'$.

(9) *If* $\Theta;\Gamma;[N] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \,/\, \chi$ *and* $\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$

then there exists P' *such that* $\Theta;\Gamma;[N] \rhd s \Rightarrow \uparrow P'$ *and* $\Theta \vdash P \equiv^{+} P'$.

Algorithmic typing completeness follows:

Theorem 9.3 (Algorithmic Typing Completeness)**.** *(Thm. G.1)*

(1) *If* $\Theta;\Gamma \vdash h \Rightarrow P$ *then* $\Theta;\Gamma \rhd h \Rightarrow P$.

(2) *If* $\Theta;\Gamma \vdash g \Rightarrow \uparrow P$ *then* $\Theta;\Gamma \rhd g \Rightarrow \uparrow P'$ *and* $\Theta \vdash P \equiv^{+} P'$ *for some* P'.

(3) *If* $\Theta;\Gamma \vdash v \Leftarrow P$ *then* $\Theta;\Gamma \rhd v \Leftarrow P$.

(4) *If* $\Theta;\Gamma \vdash e \Leftarrow N$ *then* $\Theta;\Gamma \rhd e \Leftarrow N$.

(5) *If* $\Theta;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$ *then* $\Theta;\Gamma;[P] \rhd \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$.

(6) *If* $\hat{\Theta};\Gamma;[N] \vdash s \Rightarrow \uparrow P$ *then* $\Theta;\Gamma;[N] \rhd s \Rightarrow \uparrow P'$ *and* $\Theta \vdash P \equiv^{+} P'$ *for some* P'.

Chapter 10

Conclusion

"The petty done, the undone vast."

—Robert Browning

10.1 Summary

We have presented a declarative system for modular recursive index refinement typing that is semantically sound and logically designed with basic principles. We proved our declarative system is sound for a domain-theoretic denotational semantics and an equivalent operational semantics, implying our system is logically consistent and totally correct. We have also presented an implementable algorithmic system and proved it is decidable, as well as sound and complete for the declarative system. Focusing yields CBPV, which already has clear denotational semantics, and refining it by an index domain (paying attention to value-determinedness) facilitates a semantics in line with the perspective of Melliès and Zeilberger [2015]. But focusing (in combination with value-determinedness) also allows for a relatively easy proof of the completeness of a decidable typing algorithm. The relative ease with which we demonstrate both the semantic and algorithmic correctness of a rich typing system essentially flowed from a single, proof-theoretic technique: focusing.

10.2 Future Work

Researchers of liquid typing have laid out an impressive and extensive research program providing many useful features which would be very interesting to study in our setting. But the most glaring absence in this thesis is polymorphism. We plan to add type polymorphism in future work, along the lines of previous work [Dunfield and Krishnaswami, 2013], which would go well with refinement abstraction [Vazou et al., 2013] or refinement polymorphism. We expect a form of predicative polymorphism would be straightforward to add, at least moreso than impredicative polymorphism, and there are many interesting questions to explore in the relation between the two (predicative and impredicative) and refinements in our setting. It would also be interesting to study other features of liquid typing in our setting, like extending refinement inference with templates and refinement reflection, though arguably the latter is more closely related to dependent typing.

In this thesis we only consider value-determined refinements. For simplicity we also minimize the coupling of the type system to the SMT logic. In future work it would be interesting to increase this coupling and to study carefully the semantic (in)completeness of value-determined indices in syntactic well-formedness rules. (Relatedly, it could be interesting to study the completeness of subtyping but in this thesis we wanted to keep it very simple but with huge expressive benefit.) It would also be desirable to add refinements which are not value-determined, that is, invaluable refinements [Dunfield, 2007b]. It may make sense to add invaluable refinements along with polymorphism and abstract refinements as semantically invaluable refinements seem to involve parametricity [Kennedy, 2010].

In future work, we hope to apply our type refinement system (or future extensions of it) to various domains, from static time complexity analysis [Wang et al., 2017] to resource

analysis [Handley et al., 2019]. Eventually, we hope to be able to express, for example, that a program terminates within a worst-case amount of time and space. Our system is parametric in the index domain, provided it satisfies some basic properties. Different index domains may be suitable for different applications. We also hope to add more effects, such as input/output and mutable reference cells. CBPV is built for effects, but our refinement layer may result in interesting interactions between effects and indexes.

Objects (in the sense of object-oriented programming) or coinductive types are dual to inductive types in that, semantically, they are final coalgebras of endofunctors [Cook, 2009]. A consideration of categorical duality leads us to a natural (perhaps naïve) question: if we can build a well-behaved system that refines algebraic data types by algebras, could it mean anything to refine objects by coalgebras? We would expect the most direct model of coinductive types would be via negative types, but working out the details is potential future work.

Our system may at first seem complicated, but its metatheoretic proofs are largely straightforward, if lengthy (at least as presented) and tedious. A source of this complexity is the proliferation of judgments. However, having various judgments helps us organize different forms of knowledge [Martin-Löf, 1996] or (from a Curry–Howard perspective) stages or parts of an implementation (such as pattern-matching, processing an argument list, and so on). We are always on the lookout for simplifying proofs which takes us closer to the essence of our systems. It would be especially good if we could simplify the proof of algorithmic completeness, for example, which has tedious and long but intuitively correct technical lemmas. We would also like to simplify (and expand on) unrolling identity functors and its associated metatheory, such as its interaction with subtyping.

Adding more expressive features and computational effects tends to significantly affect the metatheory and the techniques used to prove it. We hope to reflect on the development of our proofs (including those for systems with polymorphism [Dunfield and Krishnaswami, 2013, 2019]) in search of abstractions which may help designers of practical, general-purpose functional languages to establish crucial metatheoretic properties. In this respect, synthetic semantics might be a helpful tool to avoid having to (re)prove hard technical details such as those involved with domain models, especially for things like computational adequacy [Niu et al., 2024]. To borrow a metaphor from Grothendieck [McLarty, 2003], in this thesis I have relied too much on the chisel; the problem that is the metatheory of modular refinement typing could use more soaking in some liquid.

Mechanizing our metatheory would improve our understanding of and confidence in our system but then one has to deal with limitations of proof assistants, which can be especially problematic when the system and its metatheory is a work in progress. There are more extensions and simplifications to be made; hopefully by then proof assistants will have even more convenient frameworks in which we can prove and maintain both our semantic and algorithmic results more easily. Regardless, it is convenient to be able to reason about the metatheory of the language just within the ambient mathematical milieu. This was basically the approach of this thesis: apply logical principles and use standard mathematical tools, and things seem to work out quite alright. Sure, there is much more work to be done, and the difficulty might escalate rapidly, but I hope this is a decent start that will make it easier to understand and overcome difficulties as they inevitably arise.

Bibliography

Andreas Abel. Parallel hereditary substitutions. Unpublished draft, `https://www.cse.chalmers.se/~abela//notes/ParallelHereditarySubstitution.pdf`, 2019.

Andreas Abel, Thierry Coquand, and Peter Dybjer. Verifying a semantic $\beta\eta$-conversion test for Martin-Löf type theory. In *Mathematics of Program Construction (MPC'08)*, volume 5133 of *LNCS*, pages 29–56. Springer, 2008.

Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. Logic and Computation*, 2(3):297–347, 1992.

Carlo Angiuli and Daniel Gratzer. Principles of dependent type theory, April 2024. URL `https://carloangiuli.com/courses/b619-sp24/notes.pdf`.

Robert Atkey, Patricia Johann, and Neil Ghani. Refining Inductive Types. *Logical Methods in Computer Science*, Volume 8, Issue 2, 2012. URL `https://lmcs.episciences.org/957`.

Lennart Augustsson. Cayenne—a language with dependent types. In *ICFP*, pages 239–250, 1998.

J. Baez and M. Stay. *Physics, Topology, Logic and Computation: A Rosetta Stone*, pages

95–172. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-12821-9. URL https://doi.org/10.1007/978-3-642-12821-9_2.

Ondrej Baranovič. LTR, 2023. https://github.com/nulano/LTR.

H. P. Barendregt. *Lambda calculi with types*, pages 117–309. Oxford University Press, Inc., USA, 1993. ISBN 0198537611.

Hendrik Pieter Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. Sole distributors for the U.S.A. and Canada, Elsevier Science Pub. Co., New York, N.Y., 1981.

Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *J. Symbolic Logic*, 48(4):931–940, 1983.

Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. *Satisfiability modulo theories*, pages 825–885. Number 1 in Frontiers in Artificial Intelligence and Applications. IOS Press, 1st edition, 2009. ISBN 9781586039295.

João Filipe Belo, Michael Greenberg, Atsushi Igarashi, and Benjamin C. Pierce. Polymorphic Contracts. In Gilles Barthe, editor, *Proceedings of the 20th European Symposium on Programming*, volume 6602 of *Lecture Notes in Computer Science*, pages 18–37. Springer International Publishing, 2011. ISBN 978-3-642-19717-8.

Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media, 2013.

Richard Bird and Phil Wadler. *Introduction to Functional Programming*. International Series in Computer Science. Prentice Hall, 1988. ISBN 0-13-484189-1.

Susanne Bobzien. Stoic sequent logic and proof theory. *History and Philosophy of Logic*, 40(3):234–265, 2019.

Taus Brock-Nannestad, Nicolas Guenot, and Daniel Gustafsson. Computation in Focused Intuitionistic Logic. In *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming*, PPDP '15, pages 43–54, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450335164. URL https://doi.org/10.1145/2790449.2790528.

R. M. Burstall. Proving properties of programs by structural induction. *Computer Journal*, 12:41–48, 1969.

Luca Cardelli. Type Systems. *ACM Comput. Surv.*, 28(1):263–264, 1996. ISSN 0360-0300. URL https://doi.org/10.1145/234313.234418.

Iliano Cervesato and Frank Pfenning. A linear spine calculus. *J. Logic and Computation*, 13(5):639–688, 2003.

Chiyan Chen and Hongwei Xi. Combining programming with theorem proving. In *ICFP*, pages 66–77, 2005.

James Cheney and Ralf Hinze. First-class phantom types. Technical report, Cornell University, 2003.

Robert L. Constable. Mathematics as programming. In Edmund M. Clarke and Dexter Kozen, editors, *Logics of Programs, Workshop, Carnegie Mellon University, Pittsburgh, PA, USA, June 6-8, 1983, Proceedings*, volume 164 of *Lecture Notes in Computer Science*, pages 116–128. Springer, 1983. URL https://doi.org/10.1007/3-540-12896-4_359.

William R. Cook. On understanding data abstraction, revisited. In *Proceedings of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '09, pages 557–572, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605587660.

Thierry Coquand. An algorithm for type-checking dependent types. *Science of Computer Programming*, 26(1–3):167–177, 1996.

Pierre-Evariste Dagand and Conor McBride. Transporting functions across ornaments. In *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming*, ICFP '12, pages 103–114, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450310543. URL `https://doi.org/10.1145/2364527.2364544`.

Luis Damas and Robin Milner. Principal type-schemes for functional programs. In *POPL*, pages 207–212. ACM Press, 1982.

Rowan Davies and Frank Pfenning. Intersection types and computational effects. In *ICFP*, pages 198–208. ACM Press, 2000.

Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008. ISBN 978-3-540-78799-0.

Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25*, pages 378–388, Cham, 2015. Springer International Publishing. ISBN 978-3-319-21401-6.

Kosta Došen. *Logical Consequence: A Turn in Style*, pages 289–311. Springer Netherlands, Dordrecht, 1997.

Gilles Dowek. The undecidability of typability in the lambda-pi-calculus. In *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, TLCA '93, pages 139–145, Berlin, Heidelberg, 1993. Springer-Verlag. ISBN 3540565175.

Paul Downen. *Sequent Calculus: A Logic and a Language for Computation and Duality*. PhD thesis, College of Arts and Sciences, University of Oregon, 2017. Available at `https://www.cs.uoregon.edu/Reports/PHD-201706-Downen.pdf`.

Michael Dummett. *The logical basis of metaphysics*, volume 5. Harvard University Press, 1991.

Jana Dunfield. Refined typechecking with Stardust. In *Programming Languages meets Programming Verification (PLPV '07)*, pages 21–32. ACM Press, 2007a.

Jana Dunfield. *A Unified System of Type Refinements*. PhD thesis, Carnegie Mellon University, 2007b. CMU-CS-07-129.

Jana Dunfield and Neel Krishnaswami. Bidirectional typing. *ACM Comput. Surv.*, 54(5), June 2021. ISSN 0360-0300. URL `https://doi.org/10.1145/3450952`.

Jana Dunfield and Neelakantan R. Krishnaswami. Complete and easy bidirectional typechecking for higher-rank polymorphism. In *ICFP*, pages 429–442. ACM Press, 2013. arXiv:`1306.6032 [cs.PL]`.

Jana Dunfield and Neelakantan R. Krishnaswami. Sound and complete bidirectional typechecking for higher-rank polymorphism with existentials and indexed types. *Proc.*

ACM Program. Lang., 3(POPL):9:1–9:28, 2019. URL https://doi.org/10.1145/3290322.

Jana Dunfield and Frank Pfenning. Type assignment for intersections and unions in call-by-value languages. In *FoSSaCS*, pages 250–266. Springer, 2003.

Zakir Durumeric, J. Kasten, David Adrian, J. A. Halderman, M. Bailey, Frank H. Li, N. Weaver, J. Amann, J. Beekman, M. Payer, and V. Paxson. The Matter of Heartbleed. *Proceedings of the 2014 Conference on Internet Measurement Conference*, 2014.

Dimitrios J. Economou, Neel Krishnaswami, and Jana Dunfield. Focusing on refinement typing. *ACM Trans. Program. Lang. Syst.*, 45(4), December 2023. ISSN 0164-0925. URL https://doi.org/10.1145/3610408.

José Espírito Santo. The Polarized λ-calculus. *Electronic Notes in Theoretical Computer Science*, 332:149–168, 2017. ISSN 1571-0661. URL http://www.sciencedirect.com/science/article/pii/S1571066117300221. LSFA 2016 - 11th Workshop on Logical and Semantic Frameworks with Applications (LSFA).

Robert Bruce Findler and Matthias Felleisen. Contracts for higher-order functions. In *ICFP*, pages 48–59, 2002.

Cormac Flanagan, Amr Sabry, Bruce F. Duba, and Matthias Felleisen. The essence of compiling with continuations. In *PLDI*, pages 237–247, 1993.

Robert W. Floyd. Assigning Meanings to Programs. *Proceedings of Symposium on Applied Mathematics*, 19:19–32, 1967.

Tim Freeman and Frank Pfenning. Refinement types for ML. In *PLDI*, pages 268–277. ACM Press, 1991.

Gerhard Gentzen. Investigations into logical deduction. In M. Szabo, editor, *Collected papers of Gerhard Gentzen*, pages 68–131. North-Holland, 1935.

Jean-Yves Girard. A new constructive logic: classical logic. *Mathematical Structures in Computer Science*, 1(3):255–296, 1991.

Jean-Yves Girard. A fixpoint theorem in linear logic. Post to Linear Logic mailing list, `http://www.seas.upenn.edu/~sweirich/types/archive/1992/msg00030.html`, 1992.

Jean-Yves Girard. On the unity of logic. *Annals of Pure and Applied Logic*, 59(3):201–217, 1993.

Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*. Number 7 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1989.

J. A. Goguen, J. W. Thatcher, E. G. Wagner, and J. B. Wright. Initial algebra semantics and continuous algebras. *J. ACM*, 24(1):68–95, January 1977. ISSN 0004-5411. URL `https://doi.org/10.1145/321992.321997`.

Michael Greenberg, Benjamin C. Pierce, and Stephanie Weirich. Contracts made manifest. In *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '10, pages 353–364, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605584799. URL `https://doi.org/10.1145/1706299.1706341`.

Carl A. Gunter. *Semantics of programming languages - structures and techniques*. Foundations of computing. MIT Press, 1993. ISBN 978-0-262-07143-7.

Martin A. T. Handley, Niki Vazou, and Graham Hutton. Liquidate your assets: Reasoning about resource usage in Liquid Haskell. *Proc. ACM Program. Lang.*, 4(POPL), December 2019. URL https://doi.org/10.1145/3371092.

Bob Harper. The Holy Trinity, 2011. http://existentialtype.wordpress.com/2011/03/27/the-holy-trinity/.

Bob Harper and Mark Lillibridge. ML with callcc is unsound. Post to TYPES mailing list, 8 July 1991, archived at https://web.archive.org/web/20220121203611/https://www.seas.upenn.edu/~sweirich/types/archive/1991/msg00034.html, 1991.

Robert Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, 2nd edition, 2016.

Robert Harper, John C. Mitchell, and Eugenio Moggi. Higher-order modules and the phase distinction. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 341–354, San Francisco, California, USA, 1990. ACM Press. ISBN 0-89791-343-4.

Bastiaan Heeren, Jurriaan Hage, and S. Doaitse Swierstra. Generalizing Hindley-Milner Type Inference Algorithms. Technical Report UU-CS-2002-031, Department of Information and Computing Sciences, Utrecht University, 2002. URL http://www.cs.uu.nl/research/techreps/repo/CS-2002/2002-031.pdf.

R. Hindley. The principal type-scheme of an object in combinatory logic. *Trans. Amer. Math. Soc.*, 146:29–60, 1969.

C. A. R. Hoare. An Axiomatic Basis for Computer Programming. *Commun. ACM*, 12(10): 576–580, 1969. ISSN 0001-0782. URL https://doi.org/10.1145/363235.363259.

Alfred Horn. On sentences which are true of direct unions of algebras. *The Journal of Symbolic Logic*, 16(1):14–21, 1951. ISSN 00224812. URL `http://www.jstor.org/stable/2268661`.

William A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980. Original paper manuscript from 1969; `http://www.cs.cmu.edu/~crary/819-f09/Howard80.pdf`.

Ming Kawaguchi, Patrick Rondon, and Ranjit Jhala. Type-Based Data Structure Verification. In *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '09, pages 304–315, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605583921. URL `https://doi.org/10.1145/1542476.1542510`.

Andrew Kennedy. Dimension types. In *European Symposium on Programming (ESOP '94)*, volume 788, pages 348–362. Springer, 1994.

Andrew Kennedy. *Types for Units-of-Measure: Theory and Practice*, pages 268–305. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-17685-2.

Oiva Ketonen. *Untersuchungen zum Prädikatenkalkül*. PhD thesis, University of Helsinki, 1944.

Neelakantan R. Krishnaswami. Focusing on pattern matching. In *POPL*, pages 366–378. ACM Press, 2009.

Olivier Laurent. A proof of the focalization property of linear logic. Unpublished draft, `https://perso.ens-lyon.fr/olivier.laurent/llfoc.pdf`, 2004.

Nico Lehmann, Adam T. Geller, Niki Vazou, and Ranjit Jhala. Flux: Liquid Types for Rust. *Proc. ACM Program. Lang.*, 7(PLDI), June 2023. URL `https://doi.org/10.1145/3591283`.

N.G. Leveson and C.S. Turner. An investigation of the Therac-25 accidents. *Computer*, 26 (7):18–41, 1993.

Paul Blain Levy. *Call-By-Push-Value: A Functional/Imperative Synthesis (Semantics Structures in Computation, V. 2)*. Kluwer Academic Publishers, Norwell, MA, US, 2004. ISBN 1-4020-1730-8.

Chuck Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoretical Computer Science*, 410(46):4747–4768, 2009. ISSN 0304-3975. Abstract Interpretation and Logic Programming: In honor of professor Giorgio Levi.

Daniel R. Licata and Robert Harper. A formulation of Dependent ML with explicit equality proofs. Technical Report CMU-CS-05-178, Carnegie Mellon University, 2005.

Martijn Linssen. *Gospels, Epistles, Old Testament - The order of books according to Jesus Chri st*. December 2023.

Per Martin-Löf. A theory of types. Manuscript, Stockholm University. `https://raw.githubusercontent.com/michaelt/martin-lof/master/pdfs/martin-loef1971%20-%20A%20Theory%20of%20Types.pdf`, 1971.

Per Martin-Löf. An intuitionistic theory of types: Predicative part. In H. E. Rose and J. C. Shepherdson, editors, *Logic Colloquium '73*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73–118. Elsevier, 1975.

Per Martin-Löf. Constructive Mathematics and Computer Programming. In *6th International Congress for Logic, Methodology and Philosophy of Science*, pages 153–175, 1979. Published by North Holland, Amsterdam. 1982.

Per Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in Proof Theory*. Bibliopolis, 1984. ISBN 88-7088-105-9.

Per Martin-Löf. Analytic and synthetic judgements in type theory. In Paolo Parrini, editor, *Kant and Contemporary Epistemology*, pages 87–99. Springer Netherlands, 1994. ISBN 978-94-011-0834-8.

Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic*, 1(1):11–60, 1996.

Per Martin-Löf. How did 'judgement' come to be a term of logic?, 2011. URL `https://www.youtube.com/watch?v=FGUzgcLXNuk`. Transcript at `https://pml.flu.cas.cz/uploads/PML-Paris14Oct11.pdf`.

Conor McBride. Ornamental algebras, algebraic ornaments. 2011. URL `https://personal.cis.strath.ac.uk/conor.mcbride/pub/OAAO/LitOrn.pdf`.

Conor McBride and James McKinna. The view from the left. *J. Functional Programming*, 14(1):69–111, 2004.

Colin McLarty. The rising sea: Grothendieck on simplicity and generality. In *The History of Algebra in the Nineteenth and Twentieth Centuries*, 2003.

Lambert Meertens. First steps towards the theory of rose trees. `https://www.academia.edu/29461542/First_steps_towards_the_theory_of_rose_trees`, 1988.

Paul-André Melliès and Noam Zeilberger. Functors are type refinement systems. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 3–16, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450333009. URL https://doi.org/10.1145/2676726.2676970.

Robin Milner. A theory of type polymorphism in programming. *J. Computer and System Sciences*, 17(3):348–375, 1978.

Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997.

Eugenio Moggi. A category-theoretic account of program modules. In *Category Theory and Computer Science*, pages 101–117, Berlin, Heidelberg, 1989a. Springer-Verlag. ISBN 3-540-51662-X.

Eugenio Moggi. Computational lambda-calculus and monads. In *Logic in Computer Science (LICS '89)*, pages 14–23, 1989b.

Enrico Moriconi and Laura Tesconi. On inversion principles. *History and Philosophy of Logic*, 29(2):103–113, 2008. doi: 10.1080/01445340701830334.

Yue Niu, Jonathan Sterling, and Robert Harper. Cost-sensitive computational adequacy of higher-order recursion in synthetic domain theory, 2024. URL https://arxiv.org/abs/2404.00212.

nLab authors. computational trilogy. https://ncatlab.org/nlab/show/computational+trilogy, 2024. Revision 40.

Ulf Norell. Dependently Typed Programming in Agda. In *Proceedings of the 6th International Conference on Advanced Functional Programming*, AFP'08, pages 230–266, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3642046517.

James Oberg. Why the Mars probe went off course. *IEEE Spectrum*, 36(12), 1999. `http://www.jamesoberg.com/mars/loss.html`.

Martin Odersky, Matthias Zenger, and Christoph Zenger. Colored local type inference. In *POPL*, pages 41–53. ACM Press, 2001.

Chris Okasaki. *Purely Functional Data Structures*. Cambridge, 1998.

Uwe Petersen. Is Cantor's theorem a dialetheia? Variations on a paraconsistent approach to Cantor's theorem. *The Review of Symbolic Logic*, 2023. doi: 10.1017/S1755020323000187.

Simon Peyton Jones, Dimitrios Vytiniotis, Stephanie Weirich, and Mark Shields. Practical type inference for arbitrary-rank types. *J. Functional Programming*, 17(1):1–82, 2007.

Frank Pfenning. Structural cut elimination I: Intuitionistic and classical logic. *Information and Computation*, 157(1–2):84–141, 2000.

Frank Pfenning. Church and Curry: Combining intrinsic and extrinsic typing. In *Reasoning in Simple Type Theory: Festschrift in Honor of Peter B. Andrews on His 70th Birthday*. College Publications, 2008.

Brigitte Pientka. A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In *POPL*, pages 371–382. ACM Press, 2008.

Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

Benjamin C. Pierce and David N. Turner. Local type inference. *ACM Trans. Prog. Lang. Sys.*, 22:1–44, 2000.

Ruzica Piskac and Viktor Kuncak. Decision procedures for multisets with cardinality constraints. In Francesco Logozzo, Doron A. Peled, and Lenore D. Zuck, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 218–232, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-78163-9.

G. D. Plotkin. A powerdomain construction. *SIAM Journal on Computing*, 5(3):452–487, 1976. URL `https://doi.org/10.1137/0205035`.

G.D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5(3):223–255, 1977. ISSN 0304-3975.

Gordon Plotkin. Lambda-definability and logical relations. `https://www.cl.cam.ac.uk/~nk480/plotkin-logical-relations.pdf`, 1973.

Dag Prawitz. *Natural Deduction*. Almqvist & Wiksells, 1965.

John C. Reynolds. *Theories of programming languages*. Cambridge University Press, 1998. ISBN 978-0-521-59414-1.

Henry Gordon Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74(2):358–366, 1953. ISSN 00029947. URL `http://www.jstor.org/stable/1990888`.

Nick Rioux and Steve Zdancewic. Computation focusing. *Proc. ACM Program. Lang.*, 4 (ICFP), August 2020. URL `https://doi.org/10.1145/3408977`.

David Ripley. On the 'transitivity' of consequence relations. *Journal of Logic and Computation*, 28(2):433–450, 12 2017.

Patrick Rondon, Ming Kawaguchi, and Ranjit Jhala. Liquid types. In *PLDI*, pages 159–169. ACM Press, 2008.

Patrick Rondon, Alexander Bakst, Ming Kawaguchi, and Ranjit Jhala. CSolve: Verifying C with Liquid Types. In P. Madhusudan and Sanjit A. Seshia, editors, *Computer Aided Verification*, pages 744–750, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-31424-7.

Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. *Lisp Symb. Comput.*, 6(3–4):289–360, November 1993. ISSN 0892-4635. URL `https://doi.org/10.1007/BF01019462`.

Gabriel Scherer. *Which types have a unique inhabitant? Focusing on pure program equivalence*. PhD thesis, Doctorat d'Informatique Université Paris-Diderot, 2016. Available at `https://www.irif.fr/~scherer/phd_thesis/`.

Peter Schroeder-Heister. Definitional reflection and the completion. In *Extensions of Logic Programming*, LNCS, pages 333–347. Springer, 1994.

Taro Sekiyama and Atsushi Igarashi. Reasoning about polymorphic manifest contracts, 2018. URL `https://arxiv.org/abs/1806.07041`.

Taro Sekiyama, Yuki Nishida, and Atsushi Igarashi. Manifest Contracts for Datatypes. In *Proceedings of the 42nd Symposium on Principles of Programming Languages*, pages 195–207. ACM, 2015. ISBN 978-1-4503-3300-9.

Taro Sekiyama, Atsushi Igarashi, and Michael Greenberg. Polymorphic manifest contracts, revised and resolved. *ACM Trans. Program. Lang. Syst.*, 39(1), February 2017. ISSN 0164-0925. URL https://doi.org/10.1145/2994594.

Robert J. Simmons. Structural focalization. *ACM Trans. Comput. Logic*, 15(3), 2014. ISSN 1529-3785. URL https://doi.org/10.1145/2629678.

Raymond M. Smullyan. Analytic cut. *Journal of Symbolic Logic*, 33:560–564, 1968.

Alley Stoughton. Substitution revisited. *Theoretical Computer Science*, 59(3):317–325, 1988.

Sandro Stucki and Paolo G. Giarrusso. A theory of higher-order subtyping with type intervals. *Proc. ACM Program. Lang.*, 5(ICFP), August 2021. URL https://doi.org/10.1145/3473574.

Nikhil Swamy, Cătălin Hriţcu, Chantal Keller, Aseem Rastogi, Antoine Delignat-Lavaud, Simon Forest, Karthikeyan Bhargavan, Cédric Fournet, Pierre-Yves Strub, Markulf Kohlweiss, Jean-Karim Zinzindohoue, and Santiago Zanella-Béguelin. Dependent types and multi-monadic effects in F*. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '16, pages 256–270, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450335492. URL https://doi.org/10.1145/2837614.2837655.

W. W. Tait. Intensional Interpretations of Functionals of Finite Type I. *J. Symbolic Logic*, 32(2):198–212, 1967. ISSN 00224812. URL http://www.jstor.org/stable/2271658.

A. S. Troelstra. *History of constructivism in the 20th century*, pages 150–179. Lecture Notes in Logic. Cambridge University Press, 2011.

Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.

Niki Vazou, Patrick M. Rondon, and Ranjit Jhala. Abstract refinement types. In Matthias Felleisen and Philippa Gardner, editors, *Programming Languages and Systems*, pages 209–228, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-37036-6.

Niki Vazou, Eric L. Seidel, Ranjit Jhala, Dimitrios Vytiniotis, and Simon Peyton-Jones. Refinement Types for Haskell. In *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming*, ICFP '14, pages 269–282, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450328739. URL `https://doi.org/10.1145/2628136.2628161`.

Niki Vazou, Anish Tondwalkar, Vikraman Choudhury, Ryan G. Scott, Ryan R. Newton, Philip Wadler, and Ranjit Jhala. Refinement reflection: Complete verification with SMT. *Proc. ACM Program. Lang.*, 2(POPL), December 2017. URL `https://doi.org/10.1145/3158141`.

Peng Wang, Di Wang, and Adam Chlipala. TiML: A Functional Language for Practical Complexity Analysis with Invariants. *Proc. ACM Program. Lang.*, 1(OOPSLA), October 2017. URL `https://doi.org/10.1145/3133903`.

David H. D. Warren. *Applied logic: its use and implementation as a programming tool.*

PhD thesis, University of Edinburgh, UK, 1978. URL http://hdl.handle.net/1842/6648.

Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework: The propositional fragment. In *Types for Proofs and Programs*, pages 355–377. Springer LNCS 3085, 2004.

J.B. Wells. Typability and type checking in System F are equivalent and undecidable. *Annals of Pure and Applied Logic*, 98(1):111–156, 1999. ISSN 0168-0072. URL https://www.sciencedirect.com/science/article/pii/S0168007298000475.

Marcel Wild. The joy of implications, aka pure Horn formulas. *Theor. Comput. Sci.*, 658 (PB):264–292, January 2017. ISSN 0304-3975. URL https://doi.org/10.1016/j.tcs.2016.03.018.

Thomas Williams and Didier Rémy. A Principled Approach to Ornamentation in ML. *Proc. ACM Program. Lang.*, 2(POPL), 2017. URL https://doi.org/10.1145/3158109.

A.K. Wright and M. Felleisen. A syntactic approach to type soundness. *Inf. Comput.*, 115 (1):38–94, 1994. ISSN 0890-5401. URL http://dx.doi.org/10.1006/inco.1994.1093.

Andrew K. Wright. Simple imperative polymorphism. *Lisp and Symbolic Computation*, 8 (4):343–355, 1995.

Hongwei Xi. *Dependent Types in Practical Programming*. PhD thesis, Carnegie Mellon University, 1998.

Hongwei Xi. Dependent types for program termination verification. *Journal of Higher-Order and Symbolic Computation*, 15:91–131, 2002.

Hongwei Xi. Applied Type System (extended abstract). In *TYPES 2003*, LNCS, pages 394–408. Springer, 2004.

Hongwei Xi and Frank Pfenning. Eliminating array bound checking through dependent types. In *PLDI*, pages 249–257, 1998.

Hongwei Xi and Frank Pfenning. Dependent types in practical programming. In *POPL*, pages 214–227. ACM Press, 1999.

Hongwei Xi, Chiyan Chen, and Gang Chen. Guarded recursive datatype constructors. In *POPL*, pages 224–235. ACM Press, 2003.

Noam Zeilberger. On the unity of duality. *Annals of Pure and Applied Logic*, 153(1): 66–96, 2008a. ISSN 0168-0072. URL `http://www.sciencedirect.com/science/article/pii/S0168007208000080`.

Noam Zeilberger. Focusing and Higher-Order Abstract Syntax. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '08, pages 359–369, New York, NY, USA, 2008b. Association for Computing Machinery. ISBN 9781595936899. URL `https://doi.org/10.1145/1328438.1328482`.

Noam Zeilberger. Refinement types and computational duality. In *Programming Languages meets Programming Verification (PLPV '09)*, pages 15–26. ACM Press, 2009.

Appendix A

Definitions and Figures

Appendix A.1 Syntax

Program variables	x, y, z		
Expressions	e	$::=$	$\mathtt{return}\, v \mid \lambda x.e \mid \mathtt{rec}\, x : N.\, e \mid \mathtt{unreachable}$
			$\mid \mathtt{let}\, x = g;\, e \mid \mathtt{match}\, h\, \{r_i \Rightarrow e_i\}_{i \in I}$
Values	v	$::=$	$x \mid \langle\rangle \mid \langle v, v\rangle \mid \mathtt{inj}_1\, v \mid \mathtt{inj}_2\, v \mid \mathtt{into}(v) \mid \{e\}$
Heads	h	$::=$	$x \mid (v : P)$
Bound expressions	g	$::=$	$h(s) \mid (e : \uparrow P)$
Spines	s	$::=$	$\cdot \mid v, s$
Patterns	r	$::=$	$\mathtt{into}(x) \mid \langle\rangle \mid \langle x, y\rangle \mid \mathtt{inj}_1\, x \mid \mathtt{inj}_2\, x$
Types	A, B, C	$::=$	$P \mid N$
Positive types	P	$::=$	$Q \mid \exists a : \kappa.\, P$
	Q	$::=$	$R \mid Q \wedge \varphi$
	R	$::=$	$1 \mid R \times R \mid 0 \mid P + P \mid \downarrow N \mid \{\nu : \mu F \mid \mathscr{M}(F)\}$
Negative types	N	$::=$	$M \mid \forall a : \kappa.\, N$
	M	$::=$	$L \mid \varphi \supset M$
	L	$::=$	$R \to L \mid \uparrow P$
Measurements on $\nu : \mu F$	$\mathscr{M}(F)$	$::=$	$\cdot_F \mid \mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\, \nu\, \boldsymbol{u} =_\tau t$
Functors	F, G, H	$::=$	$\hat{P} \mid F \oplus F$ $\qquad$ $\mathscr{F}, \mathscr{G}, \mathscr{H} ::= F \mid \hat{B}$
	$\hat{P}$	$::=$	$\hat{I} \mid \underline{P} \otimes \hat{P}$
	$\hat{I}$	$::=$	$I \mid \mathrm{Id} \otimes \hat{I}$ $\qquad$ $\hat{B} ::= \underline{P} \mid \mathrm{Id}$
Ix. vars. and sets	$a, b, c, d, \breve{a}^{a(\boldsymbol{u})}$		$\mathfrak{A}, \mathfrak{B}, \mathfrak{C}, \mathfrak{D}$
Index terms	t, u, φ, ψ	$::=$	$a \mid n \mid t + t \mid t - t \mid (t, t) \mid \lambda a.t \mid a(\boldsymbol{t})$
			$\mid t = t \mid t \leq t \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \mathsf{tt} \mid \mathsf{ff}$
Index spines	$\boldsymbol{t}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$	$::=$	$\cdot \mid t, \boldsymbol{t} \mid .1, \boldsymbol{t} \mid .2, \boldsymbol{t}$
Algebras	α, β, γ	$::=$	$\cdot \mid (p \Rightarrow t \mid \alpha)$
Sum algebra patterns	p	$::=$	$q \mid \mathtt{inj}_1\, p \mid \mathtt{inj}_2\, p$
Product algebra patterns	q	$::=$	$() \mid (o, q)$
Base algebra patterns	o	$::=$	$\top \mid a \mid \mathtt{pk}(a, o)$
Sorts	τ, ω	$::=$	$\mathbb{B} \mid \mathbb{N} \mid \mathbb{Z} \mid \tau \times \tau \mid \kappa \Rightarrow \tau$
First-order sorts	κ	$::=$	$\mathbb{B} \mid \mathbb{N} \mid \mathbb{Z} \mid \kappa \times \kappa$

Figure A.1: Syntax

$$
\begin{array}{lrcl}
\text{Program contexts} & \Gamma & ::= & \cdot \mid \Gamma, x : R \\
\text{Logical contexts} & \Theta & ::= & \cdot \mid \Theta, a \div \tau \mid \Theta, a \overset{\mathsf{d}\cdot}{\div} \tau \mid \Theta, a\,\mathrm{Id} \mid \Theta, \varphi \\
& {}^{\mathsf{d}}\Theta & ::= & \cdot \mid {}^{\mathsf{d}}\Theta, a \overset{\mathsf{d}\cdot}{\div} \tau \mid {}^{\mathsf{d}}\Theta, a\,\mathrm{Id} \mid {}^{\mathsf{d}}\Theta, \varphi \\
& \Xi & ::= & \cdot \mid \Xi, a \div \tau \mid \Xi, a \overset{\mathsf{d}\cdot}{\div} \tau \mid \Xi, a\,\mathrm{Id} \\
& {}^{\mathsf{d}}\Xi & ::= & \cdot \mid \Xi, a \overset{\mathsf{d}\cdot}{\div} \tau \mid \Xi, a\,\mathrm{Id} \\
\text{Value-det. dependencies} & \xi & ::= & \cdot \mid \xi, \mathfrak{B} \to a \mid \xi, \blacktriangleright \mathfrak{D} \\
\text{Algorithmic contexts} & \hat{\Theta}, {}^{\mathsf{d}}\hat{\Theta}, \hat{\Xi}, {}^{\mathsf{d}}\hat{\Xi} & ::= & \Theta \mid \hat{\Theta}, \hat{a} \overset{\mathsf{d}\cdot}{\div} \kappa \mid \hat{\Theta}, \hat{a} : \kappa{=}t \\
& & & \mid \hat{\Theta}, \blacktriangleright \hat{a} \overset{\mathsf{d}\cdot}{\div} \kappa \mid \hat{\Theta}, \blacktriangleright \hat{a} : \kappa{=}t \\
& \Delta & ::= & \cdot \mid \Delta, \hat{a} \overset{\mathsf{d}\cdot}{\div} \kappa \mid \Delta, \blacktriangleright \hat{a} \overset{\mathsf{d}\cdot}{\div} \kappa \\
\text{Complete algo. contexts} & \Omega & ::= & \Theta \mid \Omega, \hat{a} : \kappa{=}t \mid \Omega, \blacktriangleright \hat{a} : \kappa{=}t \\
\text{Typing constraints} & \chi & ::= & \cdot \mid (e \Leftarrow N), \chi \mid W, \chi \\
\text{Subtyping constraints} & {}^{(\forall)}W & ::= & {}^{(\supset)}W \mid \forall a \overset{\mathsf{d}\cdot}{\div} \tau.\, {}^{(\forall)}W \\
& {}^{(\supset)}W & ::= & W \mid \varphi \supset {}^{(\supset)}W \\
& W & ::= & \varphi \mid u \equiv_\tau t \mid \boldsymbol{u} \equiv_{[\tau]} \boldsymbol{t} \mid \bigvee \overrightarrow{W} \\
& & & \mid \underline{R <:^+ P} \mid \underline{N <:^- L} \mid {}^{(\forall)}W \wedge {}^{(\forall)}W
\end{array}
$$

Figure A.2: Syntax continued

Appendix A.2 Judgments and Their Presuppositions

$\xi \vdash \mathfrak{D}\ \mathsf{det}$	(Fig. A.5)	pre.	no judgment
$\Theta\ \mathsf{ctx}$	(Fig. A.8)	pre.	no judgment
$\Theta \subseteq \Theta'$	(Fig. A.9)	pre.	$\Theta\ \mathsf{ctx}$ and $\Theta'\ \mathsf{ctx}$
$\Xi \vdash t : \tau\ [\xi_t]$	(Fig. A.11)	pre.	$\Xi\ \mathsf{ctx}$
$\Xi;[\tau] \vdash \boldsymbol{t} : \kappa$	(Fig. A.12)	pre.	$\Xi\ \mathsf{ctx}$
$\alpha \circ \mathsf{inj}_k \mathrel{\hat{=}} \alpha_k$	(Fig. A.13)	pre.	no judgment
$\Xi \vdash A\ \mathsf{type}[\xi_A]$	(Fig. A.15)	pre.	$\Xi\ \mathsf{ctx}$
$\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi]$	(Fig. A.15)	pre.	$\Xi\ \mathsf{ctx}$
$\Xi \vdash \mathscr{F}\ \mathsf{functor}[\xi_{\mathscr{F}}]$	(Fig. A.16)	pre.	$\Xi\ \mathsf{ctx}$
$\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$	(Fig. A.16)	pre.	$\Xi \vdash F\ \mathsf{functor}[\xi_F]$
$\Xi \vdash \Gamma\ \mathsf{ctx}$	(Fig. A.17)	pre.	$\Xi\ \mathsf{ctx}$
$\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma$	(Fig. A.19)	pre.	$\Theta_0\ \mathsf{ctx}$ and $\Theta\ \mathsf{ctx}$ and $\overline{\Theta_0} \vdash \Gamma_0\ \mathsf{ctx}$ and $\overline{\Theta} \vdash \Gamma\ \mathsf{ctx}$
$\Theta \vdash \varphi\ \mathsf{true}$	(Fig. A.20)	pre.	$\overline{\Theta} \vdash \varphi : \mathbb{B}$ and $\Theta\ \mathsf{ctx}$
$\Theta \vdash u \equiv t : \tau$	(Fig. A.21)	pre.	$\overline{\Theta} \vdash u : \tau$ and $\overline{\Theta} \vdash t : \tau$ and $\Theta\ \mathsf{ctx}$
$\Theta;[\tau] \vdash \boldsymbol{u} \equiv \boldsymbol{t} : \kappa$	(Fig. A.22)	pre.	$\overline{\Theta};[\tau] \vdash \boldsymbol{u} : \kappa$ and $\overline{\Theta};[\tau] \vdash \boldsymbol{t} : \kappa$ and $\Theta\ \mathsf{ctx}$
$\Theta \vdash \Theta_1 \equiv \Theta_2\ \mathsf{ctx}$	(Fig. A.23)	pre.	$(\Theta,\Theta_1)\ \mathsf{ctx}$ and $(\Theta,\Theta_2)\ \mathsf{ctx}$
$\Theta \vdash A \equiv^{\pm} B$	(Fig. A.81)	pre.	$\overline{\Theta} \vdash A\ \mathsf{type}[\xi_A]$ and $\overline{\Theta} \vdash B\ \mathsf{type}[\xi_B]$
$\Theta \vdash \mathscr{M}'(F) \equiv \mathscr{M}(F)$	(Fig. A.81)	pre.	$\overline{\Theta} \vdash \mathscr{M}'(F)\ \mathsf{msmts}[\xi]$ and $\overline{\Theta} \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi']$
$\Xi \vdash \alpha;F \equiv_\tau \beta;G$	(Fig. A.82)	pre.	$\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$ and $\Xi \vdash \beta : G(\tau) \Rightarrow \tau$
$\Theta \vdash A \leq^{\pm} B$	(Fig. A.24)	pre.	$\overline{\Theta} \vdash A\ \mathsf{type}[\xi_A]$ and $\overline{\Theta} \vdash B\ \mathsf{type}[\xi_B]$ and $\Theta\ \mathsf{ctx}$
$\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)$	(Fig. A.25)	pre.	$\overline{\Theta} \vdash \mathscr{M}'(F')\ \mathsf{msmts}[\xi']$ and $\overline{\Theta} \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi]$ and $\Theta\ \mathsf{ctx}$
$\Xi \vdash \alpha;F \leq_\tau \beta;G$	(Fig. A.25)	pre.	$\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$ and $\Xi \vdash \beta : G(\tau) \Rightarrow \tau$
$\Xi \vdash \wr \overrightarrow{\beta};G;\mathscr{M}(F) \wr \mathrel{\hat{=}} {}^{\mathsf{d}}\Theta;R$	(Fig. A.26)	pre.	$\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi]$ and $\Xi \vdash \overrightarrow{\beta} : G(\mathscr{M}(F)) \Rightarrow \mathscr{M}(F)$
$\Theta;\Gamma \vdash h \Rightarrow P$	(Fig. A.28)	pre.	$\Theta\ \mathsf{ctx}$ and $\overline{\Theta} \vdash \Gamma\ \mathsf{ctx}$
$\Theta;\Gamma \vdash g \Rightarrow \uparrow P$	(Fig. A.28)	pre.	$\Theta\ \mathsf{ctx}$ and $\overline{\Theta} \vdash \Gamma\ \mathsf{ctx}$
$\Theta;\Gamma \vdash v \Leftarrow P$	(Fig. A.29)	pre.	$\Theta\ \mathsf{ctx}$ and $\overline{\Theta} \vdash \Gamma\ \mathsf{ctx}$ and $\overline{\Theta} \vdash P\ \mathsf{type}[\xi_P]$
$\Theta;\Gamma \vdash e \Leftarrow N$	(Fig. A.30)	pre.	$\Theta\ \mathsf{ctx}$ and $\overline{\Theta} \vdash \Gamma\ \mathsf{ctx}$ and $\overline{\Theta} \vdash N\ \mathsf{type}[\xi_N]$
$\Theta;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$	(Fig. A.31)	pre.	$\Theta\ \mathsf{ctx}$ and $\overline{\Theta} \vdash \Gamma\ \mathsf{ctx}$ and $\overline{\Theta} \vdash P\ \mathsf{type}[\xi_P]$ and $\overline{\Theta} \vdash N\ \mathsf{type}[\xi_N]$
$\Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P$	(Fig. A.32)	pre.	$\Theta\ \mathsf{ctx}$ and $\overline{\Theta} \vdash \Gamma\ \mathsf{ctx}$ and $\overline{\Theta} \vdash N\ \mathsf{type}[\xi_N]$
$\vdash \delta : \Theta;\Gamma$	(Fig. A.46)	pre.	$\overline{\Theta} \vdash \Gamma\ \mathsf{ctx}$
$\hat{\Theta}\ \mathsf{algctx}$	(Fig. A.52)	pre.	no judgment
$\hat{\Xi} \rhd t : \tau\ [\xi_t]$	(Fig. A.54)	pre.	$\hat{\Xi}\ \mathsf{algctx}$
$\hat{\Xi};[\tau] \rhd \boldsymbol{t} : \kappa$	(Fig. A.55)	pre.	$\hat{\Xi}\ \mathsf{algctx}$
$\hat{\Xi} \rhd A\ \mathsf{type}[\xi]$	(Fig. A.56)	pre.	$\hat{\Xi}\ \mathsf{algctx}$
$\hat{\Xi} \rhd \mathscr{M}(F)\ \mathsf{msmts}[\xi]$	(Fig. A.15)	pre.	$\hat{\Xi}\ \mathsf{algctx}$
$\hat{\Xi} \rhd \mathscr{F}\ \mathsf{functor}[\xi]$	(Fig. A.57)	pre.	$\hat{\Xi}\ \mathsf{algctx}$
$\hat{\Xi} \rhd \alpha : F(\tau) \Rightarrow \tau$	(Fig. A.57)	pre.	$\hat{\Xi} \rhd F\ \mathsf{functor}[\xi_F]$
$\hat{\Xi} \vdash {}^{(\forall)}W\ \mathsf{wf}[\xi]$	(Fig. A.58)	pre.	$\hat{\Xi}\ \mathsf{algctx}$
$\hat{\Xi} \vdash \chi\ \mathsf{Wf}[\xi]$	(Fig. A.58)	pre.	$\hat{\Xi}\ \mathsf{algctx}$
$\Theta \models {}^{(\forall)}W$	(Fig. A.59)	pre.	$\overline{\Theta} \vdash {}^{(\forall)}W\ \mathsf{wf}[\xi]$ and $\Theta\ \mathsf{ctx}$
$\Theta;\Gamma \lhd \chi$	(Fig. A.60)	pre.	$\overline{\Theta} \vdash \chi\ \mathsf{Wf}[\xi]$ and $\overline{\Theta} \vdash \Gamma\ \mathsf{ctx}$ and $\Theta\ \mathsf{ctx}$
$\hat{\Theta} \vdash W\ \mathsf{Inst}\blacktriangleright \dashv \hat{\Theta}'$	(Fig. A.61)	pre.	$\overline{\hat{\Theta}} \vdash W\ \mathsf{wf}$ and $\hat{\Theta}\ \mathsf{algctx}$
$\hat{\Theta} \vdash {}^{(\forall)}W\ \mathsf{Inst} \dashv \hat{\Theta}'$	(Fig. A.61)	pre.	$\overline{\hat{\Theta}} \vdash {}^{(\forall)}W\ \mathsf{wf}$ and $\hat{\Theta}\ \mathsf{algctx}$
$\hat{\Theta} \vdash W\ \mathsf{fixInst} \dashv \hat{\Theta}'$	(Fig. A.61)	pre.	$\overline{\hat{\Theta}} \vdash W\ \mathsf{wf}[\xi]$ and $\hat{\Theta}\ \mathsf{algctx}$
$\hat{\Theta};\Gamma \vdash \chi\ \mathsf{fixInstChk} \dashv \Omega$	(Fig. A.61)	pre.	$\overline{\hat{\Theta}} \vdash \chi\ \mathsf{Wf}[\xi]$ and $\hat{\Theta}\ \mathsf{ctx}$ and $\|\overline{\hat{\Theta}}\| \vdash \Gamma\ \mathsf{ctx}$
$\hat{\Theta} \vdash R <:^{+} P \,/\, {}^{(\forall)}W$	(Fig. A.62)	pre.	$\|\overline{\hat{\Theta}}\| \vdash R\ \mathsf{type}[\xi_R]$ and $\overline{\hat{\Theta}} \rhd P\ \mathsf{type}[\xi_P]$ and $\hat{\Theta}\ \mathsf{present}$
$\hat{\Theta} \vdash N <:^{-} L \,/\, {}^{(\forall)}W$	(Fig. A.62)	pre.	$\overline{\hat{\Theta}} \rhd N\ \mathsf{type}[\xi_N]$ and $\|\overline{\hat{\Theta}}\| \vdash L\ \mathsf{type}[\xi_L]$ and $\hat{\Theta}\ \mathsf{present}$
$\Xi \rhd \alpha;F <:_\tau \beta;G$	(Fig. A.64)	pre.	$\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$ and $\Xi \vdash \beta : G(\tau) \Rightarrow \tau$
$\hat{\Xi} \rhd \wr \overrightarrow{\beta};G;\mathscr{M}(F) \wr \mathrel{\hat{=}} {}^{\mathsf{d}}\Theta;R$	(Fig. A.65)	pre.	$\hat{\Xi} \rhd \mathscr{M}(F)\ \mathsf{msmts}[\xi]$ and $\hat{\Xi} \rhd \overrightarrow{\beta} : G(\mathscr{M}(F)) \Rightarrow \mathscr{M}(F)$ and $\hat{\Xi}\ \mathsf{present}$
(algorithmic typing judgment presuppositions similar to above)			
$\hat{\Theta} \longrightarrow \hat{\Theta}'$	(Fig. A.71)	pre.	$\hat{\Theta}\ \mathsf{algctx}$ and $\hat{\Theta}'\ \mathsf{algctx}$
$\hat{\Theta} \xrightarrow{\mathsf{SMT}} \hat{\Theta}'$	(Fig. A.72)	pre.	$\hat{\Theta}\ \mathsf{algctx}$ and $\hat{\Theta}'\ \mathsf{algctx}$
(intermediate judgment presuppositions in Sec. A.8 similar to declarative/algorithmic versions)			

Figure A.3: Judgments and their (judgmental) presuppositions ("pre." for "presupposes")

Every judgment except submeasuring, unrolling, and algebra well-formedness presupposes that no $a\,\mathrm{Id}$ hypothesis occurs in its input context.

Appendix A.3 Declarative System

Definition A.1 (Get Value-Determined Indices ("Get Det.")). For any Θ ctx, define $\overset{\mathsf{d}}{\div}(\Theta)$ by:

$$\begin{aligned}
\overset{\mathsf{d}}{\div}\cdot &= \cdot \\
\overset{\mathsf{d}}{\div}(\Theta, a \div \tau) &= \overset{\mathsf{d}}{\div}\Theta \\
\overset{\mathsf{d}}{\div}(\Theta, a \overset{\mathsf{d}}{\div} \tau) &= \overset{\mathsf{d}}{\div}\Theta, a \overset{\mathsf{d}}{\div} \tau \\
\overset{\mathsf{d}}{\div}(\Theta, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id}) &= \overset{\mathsf{d}}{\div}\Theta, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id} \\
\overset{\mathsf{d}}{\div}(\Theta, \varphi) &= \overset{\mathsf{d}}{\div}\Theta
\end{aligned}$$

Definition A.2 (Remove Propositions). For any Θ ctx, define $\overline{\Theta}$ by:

$$\begin{aligned}
\overline{\cdot} &= \cdot \\
\overline{\Theta, a \div \tau} &= \overline{\Theta}, a \div \tau \\
\overline{\Theta, a \overset{\mathsf{d}}{\div} \tau} &= \overline{\Theta}, a \overset{\mathsf{d}}{\div} \tau \\
\overline{\Theta, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id}} &= \overline{\Theta}, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id} \\
\overline{\Theta, \varphi} &= \overline{\Theta}
\end{aligned}$$

Definition A.3 (Remove Id Variables). For any Θ ctx, define $\Theta - \mathrm{Id}$ by:

$$\begin{aligned}
\cdot - \mathrm{Id} &= \cdot \\
(\Theta, a \div \tau) - \mathrm{Id} &= (\Theta - \mathrm{Id}), a \div \tau \\
(\Theta, a \overset{\mathsf{d}}{\div} \tau) - \mathrm{Id} &= (\Theta - \mathrm{Id}), a \overset{\mathsf{d}}{\div} \tau \\
(\Theta, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id}) - \mathrm{Id} &= \Theta - \mathrm{Id} \\
(\Theta, \varphi) - \mathrm{Id} &= (\Theta - \mathrm{Id}), \varphi
\end{aligned}$$

Definition A.4 (Remove Program Entries). For any substitution σ, define $\lfloor \sigma \rfloor$ by:

$$\begin{aligned}
\lfloor \cdot \rfloor &= \cdot \\
\lfloor \sigma, t/a \rfloor &= \lfloor \sigma \rfloor, t/a \\
\lfloor \sigma, v : R/x \rfloor &= \lfloor \sigma \rfloor
\end{aligned}$$

Figure A.4: Miscellaneous operations

$\boxed{\xi \vdash a\ \mathsf{det}}$ Under (input) ξ, (input) index variable a is value-determined

$$\frac{\emptyset{\to}a \in \xi}{\xi \vdash a\ \mathsf{det}}\ \text{DetUnit} \qquad \frac{\xi \vdash c\ \mathsf{det} \qquad \xi \cup \mathfrak{C}{\to}b \vdash a\ \mathsf{det}}{\xi, (\mathfrak{C}, c){\to}b \vdash a\ \mathsf{det}}\ \text{DetCut}$$

$$\xi \vdash \mathfrak{A}\ \mathsf{det}\ \text{iff}\ \xi \vdash a\ \mathsf{det}\ \text{for all}\ a \in \mathfrak{A}$$
$$\xi \vdash {}^{\mathsf{d}}\Theta\ \mathsf{det}\ \text{iff}\ \xi \vdash \mathit{dom}({}^{\mathsf{d}}\Theta)\ \mathsf{det}$$

$$\emptyset{\to}\mathfrak{D} \triangleq \cup_{a \in \mathfrak{D}}\, \emptyset{\to}a$$

$$\cdot - a \triangleq \cdot$$
$$(\xi, \mathfrak{D}{\to}c) - a \triangleq \begin{cases} \xi - a & \text{if } c = a \\ (\xi - a) \cup ((\mathfrak{D} - a){\to}c) & \text{else} \end{cases}$$

$$\mathit{units}(\xi) \triangleq \{a \mid (\emptyset{\to}a) \in \xi\}$$
$$\mathit{pos}(\xi) \triangleq \cup_{(\mathfrak{D}{\to}a) \in \xi}\, \{a\}$$
$$\mathit{neg}(\xi) \triangleq \cup_{(\mathfrak{D}{\to}a) \in \xi}\, \mathfrak{D}$$
$$\mathit{FV}(\xi) \triangleq \cup_{(\mathfrak{D}{\to}a) \in \xi}\, \mathfrak{D} \cup \{a\}$$

The definitions in this figure extend in the obvious way
to permit evars $\hat{a}$ and to treat $\blacktriangleright\mathfrak{D}$ as $\emptyset{\to}\mathfrak{D}$.

Figure A.5: A judgment, definitions, and operations pertaining to ξ

$$
\begin{aligned}
\langle t \mid \cdot \rangle &= t \\
\langle b \mid \boldsymbol{u} \rangle &= b(\boldsymbol{u}) \quad \text{if } \boldsymbol{u} \neq \cdot \\
\langle \lambda b.t \mid u_0, \boldsymbol{u} \rangle &= \langle [u_0/b]t \mid \boldsymbol{u} \rangle \\
\langle (t_1, t_2) \mid .k, \boldsymbol{u} \rangle &= \langle t_k \mid \boldsymbol{u} \rangle \quad \text{if } k \in \{1,2\} \\
\langle a(\boldsymbol{t}) \mid \boldsymbol{u} \rangle &= \langle a \mid \boldsymbol{t}, \boldsymbol{u} \rangle \\
\langle t \mid \boldsymbol{u} \rangle &\text{ is undefined} \quad \text{for inputs } t \text{ and } \boldsymbol{u} \text{ not matching the above patterns} \\
(\langle t \mid \boldsymbol{u} \rangle &\text{ is defined if } \Xi \vdash t : \tau \text{ and } \Xi; [\tau] \vdash \boldsymbol{u} : \kappa \text{ by Lemma C.17})
\end{aligned}
$$

$$
\begin{aligned}
[\sigma](a(\boldsymbol{u})) &= \begin{cases} u & \text{if } \langle \sigma(a) \mid [\sigma]\boldsymbol{u} \rangle = u \\ a([\sigma]\boldsymbol{u}) & \text{if } a \notin dom(\sigma) \end{cases} \\
[\sigma](a(\boldsymbol{u})) &\text{ is undefined} \quad \text{if } a \in dom(\sigma) \text{ and } \langle \sigma(a) \mid [\sigma]\boldsymbol{u} \rangle \text{ is undefined} \\
([\sigma](a(\boldsymbol{u})) &\text{ is def. if } a \in dom(\sigma),\ \Xi_0 \vdash \sigma : \Xi,\ \Xi; [\tau] \vdash \boldsymbol{u} : \kappa \text{ by Lemma C.17}) \\
[\sigma]a &= \begin{cases} \sigma(a) & \text{if } a \in dom(\sigma) \\ a & \text{else} \end{cases} \\
[\sigma](t_1 + t_2) &= [\sigma]t_1 + [\sigma]t_2 \\
&\vdots \\
[\sigma]\mathsf{tt} &= \mathsf{tt} \\
[\sigma](\neg\varphi) &= \neg([\sigma]\varphi) \\
&\vdots \\
[\sigma]\{\nu : \mu F \mid \mathscr{M}(F)\} &= \{\nu : \mu[\sigma]F \mid [\sigma]\mathscr{M}([\sigma]F)\} \\
[\sigma](R \to L) &= [\sigma]R \to [\sigma]L \\
&\vdots \\
[\sigma](F_1 \oplus F_2) &= [\sigma]F_1 \oplus [\sigma]F_2 \\
&\vdots \\
[\sigma]\cdot &= \cdot \\
[\sigma](p \Rightarrow u \mid \alpha) &= p \Rightarrow [\sigma]u \mid [\sigma]\alpha \\
&\vdots \\
[\sigma](\cdot_F) &= \cdot_{[\sigma]F} \\
[\sigma](\mathscr{M}(F), (\mathsf{fold}_G\ \beta)\, \nu\, \boldsymbol{u} =_\omega u) &= [\sigma](\mathscr{M}(F)), (\mathsf{fold}_{[\sigma]G}\ [\sigma]\beta)\, \nu\, [\sigma]\boldsymbol{u} =_\omega [\sigma]u
\end{aligned}
$$

$$
\begin{aligned}
[\sigma]\cdot &= \cdot \\
[\sigma](\check{\Xi}, \check{a}^{a(\boldsymbol{u})} \mathbin{\dot{-}} \kappa) &= [\sigma]\check{\Xi}, \check{a}^{a([\sigma]\boldsymbol{u})} \mathbin{\dot{-}} \kappa
\end{aligned}
$$

Figure A.6: Index substitution and hereditary reduction

$$
\begin{aligned}
[\sigma]^{\mathsf{h}} x &= \begin{cases} x & \text{if } x \notin dom(\sigma) \text{ or } \sigma(x) = (x : P) \\ \sigma(x) & \text{else} \end{cases} \\
[\sigma]^{\mathsf{h}}(v : P) &= ([\sigma]v : [\lfloor\sigma\rfloor]P) \\
\\
[\sigma](h(s)) &= ([\sigma]^{\mathsf{h}} h)([\sigma]s) \\
[\sigma](e : \uparrow P) &= ([\sigma]e : [\lfloor\sigma\rfloor](\uparrow P)) \\
\\
[\sigma]x &= \begin{cases} x & \text{if } x \notin dom(\sigma) \\ v & \text{if } \sigma(x) = (v : P) \end{cases} \\
[\sigma]\langle v_1, v_2\rangle &= \langle [\sigma]v_1, [\sigma]v_2\rangle \\
&\vdots \\
[\sigma](\mathtt{match}\ h\ \{r_i \Rightarrow e_i\}_{i\in I}) &= \mathtt{match}\ \left([\sigma]^{\mathsf{h}} h\right)\ ([\sigma]\{r_i \Rightarrow e_i\}_{i\in I}) \\
&\vdots \\
[\sigma](\lambda x.e) &= \lambda x.[\sigma]e \\
[\sigma](\mathtt{rec}\, x : (\forall a : \mathbb{N}.N).\ e) &= \mathtt{rec}\, x : [\lfloor\sigma\rfloor](\forall a : \mathbb{N}.N).\ [\sigma]e \\
&\vdots
\end{aligned}
$$

Figure A.7: Definition of syntactic substitution on program terms

$\boxed{\Theta\ \mathsf{ctx}}$ Input logical context Θ is well-formed

$$\frac{}{\cdot\ \mathsf{ctx}}\ \text{LogCtxEmpty} \qquad \frac{\Theta\ \mathsf{ctx} \qquad a \notin dom(\Theta)}{\begin{array}{c}(\Theta, a \div \tau)\ \mathsf{ctx} \\ (\Theta, a \overset{\mathsf{d}}{\div} \tau)\ \mathsf{ctx} \\ (\Theta, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id})\ \mathsf{ctx}\end{array}}\ \text{LogCtxVar}$$

$$\frac{\Theta\ \mathsf{ctx} \qquad \overline{\Theta} \vdash \varphi : \mathbb{B}}{(\Theta, \varphi)\ \mathsf{ctx}}\ \text{LogCtxProp}$$

Figure A.8: Declarative logical context well-formedness

$\boxed{\Theta \subseteq \Theta'}$ Input logical context Θ is a subcontext of input Θ'

$$\frac{}{\cdot \subseteq \cdot} \qquad \frac{\Theta \subseteq \Theta'}{\Theta \subseteq \Theta', a : \tau} \qquad \frac{\Theta \subseteq \Theta'}{\Theta \subseteq \Theta', \varphi} \qquad \frac{\Theta \subseteq \Theta'}{\begin{array}{c}\Theta, a \div \tau \subseteq \Theta', a \div \tau \\ \Theta, a \overset{\mathrm{d}}{\div} \tau \subseteq \Theta', a \overset{\mathrm{d}}{\div} \tau \\ \Theta, a \overset{\mathrm{d}}{\div} \tau, a\,\mathrm{Id} \subseteq \Theta', a \overset{\mathrm{d}}{\div} \tau, a\,\mathrm{Id}\end{array}}$$

$$\frac{\Theta \subseteq \Theta'}{\Theta, \varphi \subseteq \Theta', \varphi}$$

Figure A.9: Logical subcontext

For each base sort κ, we define the set $\mathscr{K}_\kappa$ of constant terms of sort κ:

$$\begin{aligned}\mathscr{K} &: \{\mathbb{B}, \mathbb{N}, \mathbb{Z}\} \to \mathrm{Set} \\ \mathscr{K}_{\mathbb{B}} &= \{\mathsf{tt}, \mathsf{ff}\} \\ \mathscr{K}_{\mathbb{N}} &= \{0, 1, 2, \dots\} \\ \mathscr{K}_{\mathbb{Z}} &= \{\dots, -2, -1, 0, 1, 2, \dots\}\end{aligned}$$

Figure A.10: Constant index terms

$\boxed{\Xi \vdash t : \tau\ [\xi_t]}$ Under input Ξ, input index t has input sort τ and output value-determined dependencies ξ_t
Note that $\Xi \vdash t : \tau$ abbreviates $\Xi \vdash t : \tau\ [_]$ where "_" means "don't care"

$$\frac{(a : \tau) \in \Xi}{\Xi \vdash a : \tau\ [\cdot]}\ \text{IxVar} \qquad \frac{\Xi \vdash t : \tau\ [\xi'] \qquad \xi \subsetneq \xi'}{\Xi \vdash t : \tau\ [\xi]}\ \text{IxSub} \qquad \frac{t \in \mathscr{K}_\kappa}{\Xi \vdash t : \kappa\ [\cdot]}\ \text{IxConst}$$

$$\frac{\kappa \in \{\mathbb{N}, \mathbb{Z}\} \qquad \Xi \vdash t_1 : \kappa\ [\xi_1] \qquad \Xi \vdash t_2 : \kappa\ [\xi_2]}{\Xi \vdash t_1 + t_2 : \kappa\ [\cdot]}\ \text{Ix}+$$

$$\frac{\kappa \in \{\mathbb{N}, \mathbb{Z}\} \qquad \Xi \vdash t_1 : \kappa\ [\xi_1] \qquad \Xi \vdash t_2 : \kappa\ [\xi_2]}{\Xi \vdash t_1 - t_2 : \kappa\ [\cdot]}\ \text{Ix}-$$

$$\frac{\Xi \vdash t_1 : \tau_1\ [\xi_1] \qquad \Xi \vdash t_2 : \tau_2\ [\xi_2]}{\Xi \vdash (t_1, t_2) : \tau_1 \times \tau_2\ [\cdot]}\ \text{Ix}\times$$

$$\frac{\Xi, a \div \kappa \vdash t : \tau\ [\xi_t]}{\Xi \vdash \lambda a.t : \kappa \Rightarrow \tau\ [\cdot]}\ \text{Ix}\lambda \qquad \frac{(a : \tau) \in \Xi \qquad \Xi; [\tau] \vdash \boldsymbol{t} : \kappa}{\Xi \vdash a(\boldsymbol{t}) : \kappa\ [\cdot]}\ \text{IxApp}$$

$$\frac{(a \overset{\mathrm{d}}{\div} \kappa) \in \Xi \qquad (t \overset{\mathrm{d}}{\div} \kappa) \notin \Xi \qquad \overset{\mathrm{d}}{\div}\Xi \vdash t : \kappa\ [\xi_t]}{\Xi \vdash a = t : \mathbb{B}\ [FV(t) \to a]}\ \text{Ix=L}$$

$$\frac{(a \overset{\mathrm{d}}{\div} \kappa) \in \Xi \qquad (t \overset{\mathrm{d}}{\div} \kappa) \notin \Xi \qquad \overset{\mathrm{d}}{\div}\Xi \vdash t : \kappa\ [\xi_t]}{\Xi \vdash t = a : \mathbb{B}\ [FV(t) \to a]}\ \text{Ix=R}$$

$$\frac{(a \overset{\mathrm{d}}{\div} \kappa) \in \Xi \qquad (b \overset{\mathrm{d}}{\div} \kappa) \in \Xi}{\Xi \vdash a = b : \mathbb{B}\ [a \to b, b \to a]}\ \text{Ix=LR}$$

$$\frac{\Xi \vdash u_1 = t_1 : \mathbb{B}\ [\xi_1] \qquad \Xi \vdash u_2 = t_2 : \mathbb{B}\ [\xi_2]}{\Xi \vdash (u_1, u_2) = (t_1, t_2) : \mathbb{B}\ [\xi_1 \cup \xi_2]}\ \text{Ix=}\times$$

$$\frac{\text{no other rule applies} \qquad \Xi \vdash t_1 : \kappa\ [\xi_1] \qquad \Xi \vdash t_2 : \kappa\ [\xi_2]}{\Xi \vdash t_1 = t_2 : \mathbb{B}\ [\cdot]}\ \text{Ix=}$$

$$\frac{\Xi \vdash \varphi_1 : \mathbb{B}\ [\xi_1] \qquad \Xi \vdash \varphi_2 : \mathbb{B}\ [\xi_2]}{\Xi \vdash \varphi_1 \wedge \varphi_2 : \mathbb{B}\ [\xi_1 \cup \xi_2]}\ \text{Ix}\wedge \qquad \frac{\Xi \vdash \varphi_1 : \mathbb{B}\ [\xi_1] \qquad \Xi \vdash \varphi_2 : \mathbb{B}\ [\xi_2]}{\Xi \vdash \varphi_1 \vee \varphi_2 : \mathbb{B}\ [\cdot]}\ \text{Ix}\vee$$

$$\frac{\Xi \vdash \varphi : \mathbb{B}\ [\xi_\varphi]}{\Xi \vdash \neg\varphi : \mathbb{B}\ [\cdot]}\ \text{Ix}\neg \qquad \frac{\kappa \in \{\mathbb{N}, \mathbb{Z}\} \qquad \Xi \vdash t_1 : \kappa\ [\xi_1] \qquad \Xi \vdash t_2 : \kappa\ [\xi_2]}{\Xi \vdash t_1 \leq t_2 : \mathbb{B}\ [\cdot]}\ \text{Ix}\leq$$

Figure A.11: Declarative index sorting

$\boxed{\Xi;[\tau] \vdash \boldsymbol{t} : \kappa}$ Under input Ξ, fully applying an index of sort τ to $\boldsymbol{t}$ (inputs) yields an index of sort κ (output)

$$\frac{}{\Xi;[\kappa] \vdash \cdot : \kappa}\ \text{IxSpineNil} \qquad \frac{\Xi \vdash t_0 : \kappa_0 \qquad \Xi;[\tau] \vdash \boldsymbol{t} : \kappa}{\Xi;[\kappa_0 \Rightarrow \tau] \vdash t_0, \boldsymbol{t} : \kappa}\ \text{IxSpineEntry}$$

$$\frac{k \in \{1,2\} \qquad \Xi;[\tau_k] \vdash \boldsymbol{t} : \kappa}{\Xi;[\tau_1 \times \tau_2] \vdash .k, \boldsymbol{t} : \kappa}\ \text{IxSpineProj}_k$$

Figure A.12: Declarative index spine sorting

$\boxed{\begin{array}{l}\alpha \circ \mathsf{inj}_1 \mathrel{\hat{=}} \alpha_1 \\ \alpha \circ \mathsf{inj}_2 \mathrel{\hat{=}} \alpha_2\end{array}}$ The left pattern of algebra α (input) is α_1 (output)
The right pattern of algebra α (input) is α_2 (output)

$$\frac{}{\cdot \circ \mathsf{inj}_1 \mathrel{\hat{=}} \cdot}\ \text{DeclPatNil}_1 \qquad \frac{\alpha \circ \mathsf{inj}_1 \mathrel{\hat{=}} \beta}{(\mathsf{inj}_2\, p \Rightarrow t \mid \alpha) \circ \mathsf{inj}_1 \mathrel{\hat{=}} \beta}\ \text{DeclPatThere}_1$$

$$\frac{\alpha \circ \mathsf{inj}_1 \mathrel{\hat{=}} \beta}{(\mathsf{inj}_1\, p \Rightarrow t \mid \alpha) \circ \mathsf{inj}_1 \mathrel{\hat{=}} (p \Rightarrow t \mid \beta)}\ \text{DeclPatHere}_1$$

$$\frac{}{\cdot \circ \mathsf{inj}_2 \mathrel{\hat{=}} \cdot}\ \text{DeclPatNil}_2 \qquad \frac{\alpha \circ \mathsf{inj}_2 \mathrel{\hat{=}} \beta}{(\mathsf{inj}_1\, p \Rightarrow t \mid \alpha) \circ \mathsf{inj}_2 \mathrel{\hat{=}} \beta}\ \text{DeclPatThere}_2$$

$$\frac{\alpha \circ \mathsf{inj}_2 \mathrel{\hat{=}} \beta}{(\mathsf{inj}_2\, p \Rightarrow t \mid \alpha) \circ \mathsf{inj}_2 \mathrel{\hat{=}} (p \Rightarrow t \mid \beta)}\ \text{DeclPatHere}_2$$

Figure A.13: Algebra pattern selection

$$\begin{aligned}
\exists \cdot . P &= P \\
\exists (\Theta, a \div \kappa) . P &= \exists \Theta . P \\
\exists (\Theta, a \stackrel{\mathsf{d}}{\div} \kappa) . P &= \exists \Theta . (\exists a \stackrel{\mathsf{d}}{\div} \kappa . P) \\
\exists (\Theta, \varphi) . P &= \exists \Theta . P
\end{aligned}
\qquad
\begin{aligned}
Q \wedge \cdot &= Q \\
Q \wedge (\Theta, a : \kappa) &= Q \wedge \Theta \\
Q \wedge (\Theta, \varphi) &= (Q \wedge \varphi) \wedge \Theta
\end{aligned}$$

$$\begin{aligned}
\forall \cdot . N &= N \\
\forall (\Theta, a \div \kappa) . N &= \forall \Theta . N \\
\forall (\Theta, a \stackrel{\mathsf{d}}{\div} \kappa) . N &= \forall \Theta . (\forall a \stackrel{\mathsf{d}}{\div} \kappa . N) \\
\forall (\Theta, \varphi) . N &= \forall \Theta . N
\end{aligned}
\qquad
\begin{aligned}
\cdot \supset M &= M \\
(\Theta, a : \kappa) \supset M &= \Theta \supset M \\
(\Theta, \varphi) \supset M &= \Theta \supset (\varphi \supset M)
\end{aligned}$$

Figure A.14: Metaoperations for forming types over logical contexts

$\boxed{\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi_{\mathscr{M}(F)}]}$ Under input Ξ, input measurements $\mathscr{M}(F)$ are well-formed, with output value-determined dependencies $\xi_{\mathscr{M}(F)}$

$$\frac{\cdot \vdash F\ \mathsf{functor}[_]}{\Xi \vdash \cdot_F\ \mathsf{msmts}[\cdot]}$$

$$\frac{\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi] \qquad \cdot \vdash \alpha : F(\tau) \Rightarrow \tau \qquad {}^{\mathsf{d}\div}\Xi;[\tau] \vdash \boldsymbol{t} : \kappa \qquad (t\ {}^{\mathsf{d}\div}\ \kappa) \in \Xi}{\Xi \vdash \mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\ \nu\, \boldsymbol{t} =_\tau t\ \mathsf{msmts}[\xi \cup FV(\boldsymbol{t}) \to t]}$$

$$\frac{\begin{array}{c}\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi] \\ \cdot \vdash \alpha : F(\tau) \Rightarrow \tau \qquad {}^{\mathsf{d}\div}\Xi;[\tau] \vdash \boldsymbol{t} : \kappa \qquad (t\ {}^{\mathsf{d}\div}\ \kappa) \notin \Xi \qquad {}^{\mathsf{d}\div}\Xi \vdash t : \kappa\end{array}}{\Xi \vdash \mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\ \nu\, \boldsymbol{t} =_\tau t\ \mathsf{msmts}[\xi]}$$

$\boxed{\Xi \vdash A\ \mathsf{type}[\xi_A]}$ Under Ξ (input), type A (input) is well-formed, with (output) value-determined dependencies ξ_A

$$\frac{\Xi, {}^{\mathsf{d}}\Xi \vdash Q\ \mathsf{type}[\xi_Q] \qquad \xi_Q - {}^{\mathsf{d}\div}\Xi \vdash {}^{\mathsf{d}}\Xi\ \mathsf{det}}{\Xi \vdash \exists {}^{\mathsf{d}}\Xi.\, Q\ \mathsf{type}[\xi_Q - {}^{\mathsf{d}}\Xi]}\ \mathrm{DeclTp}\exists$$

$$\frac{\Xi \vdash R\ \mathsf{type}[\xi_R] \qquad \Xi \vdash \overrightarrow{\varphi} : \mathbb{B}\ [\xi_{\overrightarrow{\varphi}}]}{\Xi \vdash R \wedge \overrightarrow{\varphi}\ \mathsf{type}[\xi_R \cup \xi_{\overrightarrow{\varphi}}]}\ \mathrm{DeclTp}\wedge$$

$$\frac{\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi]}{\Xi \vdash \{\nu : \mu F \mid \mathscr{M}(F)\}\ \mathsf{type}[\xi]}\ \mathrm{DeclTp}\mu$$

$$\frac{\Xi \vdash P_1\ \mathsf{type}[\xi_1] \qquad \Xi \vdash P_2\ \mathsf{type}[\xi_2]}{\Xi \vdash P_1 + P_2\ \mathsf{type}[\cdot]}\ \mathrm{DeclTp}+$$

$$\frac{\Xi \vdash R_1\ \mathsf{type}[\xi_1] \qquad \Xi \vdash R_2\ \mathsf{type}[\xi_2]}{\Xi \vdash R_1 \times R_2\ \mathsf{type}[\xi_1 \cup \xi_2]}\ \mathrm{DeclTp}\times \qquad \frac{}{\Xi \vdash 0\ \mathsf{type}[\cdot]}\ \mathrm{DeclTp0}$$

$$\frac{}{\Xi \vdash 1\ \mathsf{type}[\cdot]}\ \mathrm{DeclTp1} \qquad \frac{\Xi \vdash N\ \mathsf{type}[\xi_N]}{\Xi \vdash \downarrow N\ \mathsf{type}[\cdot]}\ \mathrm{DeclTp}\downarrow \qquad \frac{\Xi \vdash P\ \mathsf{type}[\xi_P]}{\Xi \vdash \uparrow P\ \mathsf{type}[\cdot]}\ \mathrm{DeclTp}\uparrow$$

$$\frac{\Xi \vdash R\ \mathsf{type}[\xi_R] \qquad \Xi \vdash L\ \mathsf{type}[\xi_L]}{\Xi \vdash R \to L\ \mathsf{type}[\xi_R \cup \xi_L]}\ \mathrm{DeclTp}\to$$

$$\frac{\Xi \vdash L\ \mathsf{type}[\xi_L] \qquad \Xi \vdash \overrightarrow{\varphi} : \mathbb{B}\ [\xi_{\overrightarrow{\varphi}}]}{\Xi \vdash \overrightarrow{\varphi} \supset L\ \mathsf{type}[\xi_L \cup \xi_{\overrightarrow{\varphi}}]}\ \mathrm{DeclTp}\supset$$

$$\frac{\Xi, {}^{\mathsf{d}}\Xi \vdash M\ \mathsf{type}[\xi_M] \qquad \xi_M - {}^{\mathsf{d}\div}\Xi \vdash {}^{\mathsf{d}}\Xi\ \mathsf{det}}{\Xi \vdash \forall {}^{\mathsf{d}}\Xi.\, M\ \mathsf{type}[\xi_M - {}^{\mathsf{d}}\Xi]}\ \mathrm{DeclTp}\forall$$

Figure A.15: Declarative well-formedness of types (and measurements)

$\boxed{\Xi \vdash \mathscr{F}\ \mathsf{functor}[\xi_{\mathscr{F}}]}$ Under Ξ (input), functor $\mathscr{F}$ (input) is well-formed, with (output) value-determined dependencies $\xi_{\mathscr{F}}$

$$\frac{\Xi \vdash P\ \mathsf{type}[\xi]}{\Xi \vdash \underline{P}\ \mathsf{functor}[\xi]}\ \text{DeclFuncConst} \qquad \frac{}{\Xi \vdash \mathrm{Id}\ \mathsf{functor}[\cdot]}\ \text{DeclFuncId}$$

$$\frac{}{\Xi \vdash I\ \mathsf{functor}[\cdot]}\ \text{DeclFunc}I \qquad \frac{\Xi \vdash \hat{B}\ \mathsf{functor}[\xi_1] \qquad \Xi \vdash \hat{P}\ \mathsf{functor}[\xi_2]}{\Xi \vdash \hat{B} \otimes \hat{P}\ \mathsf{functor}[\xi_1 \cup \xi_2]}\ \text{DeclFunc}\otimes$$

$$\frac{\Xi \vdash F_1\ \mathsf{functor}[\xi_1] \qquad \Xi \vdash F_2\ \mathsf{functor}[\xi_2]}{\Xi \vdash F_1 \oplus F_2\ \mathsf{functor}[\cdot]}\ \text{DeclFunc}\oplus$$

$\boxed{\Xi \vdash \alpha : F(\tau) \Rightarrow \tau}$ Under Ξ (input), α (input) is a well-formed algebra of kind $F(\tau) \Rightarrow \tau$ (inputs: F and τ)

$$\frac{\begin{array}{ll}\alpha \circ \mathsf{inj}_1 \mathrel{\mathring{=}} \alpha_1 & \Xi \vdash \alpha_1 : F_1(\tau) \Rightarrow \tau \\ \alpha \circ \mathsf{inj}_2 \mathrel{\mathring{=}} \alpha_2 & \Xi \vdash \alpha_2 : F_2(\tau) \Rightarrow \tau\end{array}}{\Xi \vdash \alpha : (F_1 \oplus F_2)(\tau) \Rightarrow \tau}\ \text{DeclAlg}\oplus$$

$$\frac{\Xi \vdash Q\ \mathsf{type}[\xi_Q] \qquad \Xi \vdash q \Rightarrow t : \hat{P}(\tau) \Rightarrow \tau}{\Xi \vdash (\top, q) \Rightarrow t : (\underline{Q} \otimes \hat{P})(\tau) \Rightarrow \tau}\ \text{DeclAlgConst}$$

$$\frac{\Xi, {}^{\mathsf{d}}\Xi' \vdash (\top, q) \Rightarrow t : (\underline{Q} \otimes \hat{P})(\tau) \Rightarrow \tau}{\Xi \vdash (\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q) \Rightarrow t : (\underline{\exists^{\mathsf{d}}\Xi'.\, Q} \otimes \hat{P})(\tau) \Rightarrow \tau}\ \text{DeclAlg}\exists$$

$$\frac{\Xi, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id} \vdash q \Rightarrow t : \hat{I}(\tau) \Rightarrow \tau}{\Xi \vdash (a, q) \Rightarrow t : (\mathrm{Id} \otimes \hat{I})(\tau) \Rightarrow \tau}\ \text{DeclAlgId}$$

$$\frac{\overset{\mathsf{d}}{\div}\Xi \vdash t : \tau}{\Xi \vdash () \Rightarrow t : I(\tau) \Rightarrow \tau}\ \text{DeclAlg}I$$

$\boxed{\Xi \vdash \alpha : (F)\mathbb{B} \Rightarrow \mathbb{B} = _ \mapsto \mathsf{tt}}$ $\alpha : F(\mathbb{B}) \Rightarrow \mathbb{B}$ is constantly true, often writing α as $\underline{\mathsf{tt}}^{(F)}$ (inputs: α and F)

$$\frac{\Xi \vdash \alpha_1 : (F_1)\mathbb{B} \Rightarrow \mathbb{B} = _ \mapsto \mathsf{tt} \qquad \Xi \vdash \alpha_2 : (F_2)\mathbb{B} \Rightarrow \mathbb{B} = _ \mapsto \mathsf{tt}}{\Xi \vdash \mathsf{inj}_1\,\alpha_1 \mid \mathsf{inj}_2\,\alpha_2 : (F_1 \oplus F_2)\mathbb{B} \Rightarrow \mathbb{B} = _ \mapsto \mathsf{tt}}$$

$$\frac{\Xi \vdash q \Rightarrow t : (\hat{P})\mathbb{B} \Rightarrow \mathbb{B} = _ \mapsto \mathsf{tt}}{\Xi \vdash (\mathtt{pk}({}^{\mathsf{d}}\Xi, \top), q) \Rightarrow t : (\underline{\exists^{\mathsf{d}}\Xi.\, Q} \otimes \hat{P})\mathbb{B} \Rightarrow \mathbb{B} = _ \mapsto \mathsf{tt}}$$

$$\frac{\Xi \vdash q \Rightarrow t : (\hat{I})\mathbb{B} \Rightarrow \mathbb{B} = _ \mapsto \mathsf{tt}}{\Xi \vdash (a, q) \Rightarrow t : (\mathrm{Id} \otimes \hat{I})\mathbb{B} \Rightarrow \mathbb{B} = _ \mapsto \mathsf{tt}} \qquad \frac{}{\Xi \vdash () \Rightarrow \mathsf{tt} : (I)\mathbb{B} \Rightarrow \mathbb{B} = _ \mapsto \mathsf{tt}}$$

Figure A.16: Declarative well-formedness of functors and algebras

$\boxed{\Xi \vdash \Gamma\ \mathsf{ctx}}$ Under logical context Ξ (input), program context Γ (input) is well-formed

$$\frac{}{\Xi \vdash \cdot\ \mathsf{ctx}}\ \text{ProgCtxEmpty} \qquad \frac{\Xi \vdash \Gamma\ \mathsf{ctx} \qquad \Xi \vdash R\ \mathsf{type}[\xi] \qquad x \notin dom(\Gamma)}{\Xi \vdash \Gamma, x : R\ \mathsf{ctx}}\ \text{ProgCtxVar}$$

Figure A.17: Program context well-formedness

$$\text{Syntactic substitutions} \quad \sigma \ ::= \ \cdot \mid \sigma, t/a \mid \sigma, v : R/x$$

Figure A.18: Syntactic substitution

$\boxed{\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma}$ Under Θ_0 and Γ_0, σ is a syntactic substitution for variables in Θ and Γ (Inputs: $\Theta_0, \Gamma_0, \sigma, \Theta, \Gamma$)

$$\frac{}{\Theta_0; \Gamma_0 \vdash \cdot : \cdot; \cdot}\ \text{SubstEmpty}$$

$$\frac{\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma \qquad \overline{\Theta_0} \vdash t : \tau \qquad a \notin dom(\Theta)}{\Theta_0; \Gamma_0 \vdash \sigma, t/a : \Theta, a \div \tau; \Gamma}\ \text{SubstIx}$$

$$\frac{\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma \qquad \overset{\mathsf{d}}{\div}\Theta_0 \vdash t : \tau \qquad a \notin dom(\Theta)}{\Theta_0; \Gamma_0 \vdash \sigma, t/a : \Theta, a \overset{\mathsf{d}}{\div} \tau[, a\,\mathrm{Id}]; \Gamma}\ \text{SubstIxDet}$$

$$\frac{\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma \qquad \Theta_0 \vdash [\lfloor \sigma \rfloor] \varphi\ \mathsf{true}}{\Theta_0; \Gamma_0 \vdash \sigma : \Theta, \varphi; \Gamma}\ \text{SubstProp}$$

$$\frac{\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma \qquad \Theta_0; \Gamma_0 \vdash v \Leftarrow [\lfloor \sigma \rfloor] R \qquad x \notin dom(\Gamma)}{\Theta_0; \Gamma_0 \vdash \sigma, v : [\lfloor \sigma \rfloor] R/x : \Theta; \Gamma, x : R}\ \text{SubstVal}$$

Figure A.19: Syntactic substitution

$\boxed{\Theta \vdash \varphi\ \mathsf{true}}$ Under input Θ, input index φ is true

$$\frac{[\![\varphi]\!]_\delta = \{\bullet\} \text{ for all } \delta \in [\![\Theta]\!]}{\Theta \vdash \varphi\ \mathsf{true}}\ \text{PropTrue}$$

Figure A.20: Index (of boolean sort) validity or truth

$\boxed{\Theta \vdash u \equiv t : \tau}$ Under Θ, index terms u and t are equivalent and have sort τ (inputs: Θ, u, t, τ)

$$\frac{\Theta - \mathrm{Id} \vdash u = t \ \mathsf{true}}{\Theta \vdash u \equiv t : \kappa}\ \text{Ix}{\equiv}\text{SMT} \qquad \frac{(a : \tau) \in \Theta}{\Theta \vdash a \equiv a : \tau}\ \text{Ix}{\equiv}\text{Var}$$

$$\frac{\begin{array}{c}\kappa \in \{\mathbb{N}, \mathbb{Z}\} \\ FV(t_1, t_2, t_1', t_2') \nsubseteq dom(\Theta - \mathrm{Id}) \qquad \Theta \vdash t_1 \equiv t_1' : \kappa \qquad \Theta \vdash t_2 \equiv t_2' : \kappa\end{array}}{\Theta \vdash t_1 + t_2 \equiv t_1' + t_2' : \kappa}\ \text{Ix}{\equiv}\text{Plus}$$

$$\frac{\begin{array}{c}\kappa \in \{\mathbb{N}, \mathbb{Z}\} \\ FV(t_1, t_2, t_1', t_2') \nsubseteq dom(\Theta - \mathrm{Id}) \qquad \Theta \vdash t_1 \equiv t_1' : \kappa \qquad \Theta \vdash t_2 \equiv t_2' : \kappa\end{array}}{\Theta \vdash t_1 - t_2 \equiv t_1' - t_2' : \kappa}\ \text{Ix}{\equiv}\text{Minus}$$

$$\frac{\begin{array}{c}FV(u_1, u_2, t_1, t_2) \nsubseteq dom(\Theta - \mathrm{Id}) \text{ or } \tau_1 \times \tau_2 \neq \kappa \\ \Theta \vdash u_1 \equiv t_1 : \tau_1 \qquad \Theta \vdash u_2 \equiv t_2 : \tau_2\end{array}}{\Theta \vdash (u_1, u_2) \equiv (t_1, t_2) : \tau_1 \times \tau_2}\ \text{Ix}{\equiv}\text{Prod}$$

$$\frac{FV(u, t) \nsubseteq dom((\Theta, a \div \kappa) - \mathrm{Id}) \qquad \Theta, a \div \kappa \vdash u \equiv t : \tau}{\Theta \vdash \lambda a.u \equiv \lambda a.t : \kappa \Rightarrow \tau}\ \text{Ix}{\equiv}\lambda$$

$$\frac{FV(a, \boldsymbol{t}, \boldsymbol{t}') \nsubseteq dom(\Theta - \mathrm{Id}) \qquad (a \overset{\mathrm{d}}{\div} \tau) \in \Theta \qquad \Theta; [\tau] \vdash \boldsymbol{t} \equiv \boldsymbol{t}' : \kappa}{\Theta \vdash a(\boldsymbol{t}) \equiv a(\boldsymbol{t}') : \kappa}\ \text{Ix}{\equiv}\text{App}$$

$$\frac{\begin{array}{c}FV(t_1, t_2, t_1', t_2') \nsubseteq dom(\Theta - \mathrm{Id}) \\ \overline{\Theta} \vdash t_1 : \kappa \qquad \overline{\Theta} \vdash t_1' : \kappa \qquad \Theta \vdash t_1 \equiv t_1' : \kappa \qquad \Theta \vdash t_2 \equiv t_2' : \kappa\end{array}}{\Theta \vdash t_1 = t_2 \equiv t_1' = t_2' : \mathbb{B}}\ \text{Ix}{\equiv}{=}$$

$$\frac{FV(\varphi_1, \varphi_2, \psi_1, \psi_2) \nsubseteq dom(\Theta - \mathrm{Id}) \qquad \Theta \vdash \varphi_1 \equiv \psi_1 : \mathbb{B} \qquad \Theta \vdash \varphi_2 \equiv \psi_2 : \mathbb{B}}{\Theta \vdash \varphi_1 \wedge \varphi_2 \equiv \psi_1 \wedge \psi_2 : \mathbb{B}}\ \text{Ix}{\equiv}\wedge$$

$$\frac{FV(\varphi_1, \varphi_2, \psi_1, \psi_2) \nsubseteq dom(\Theta - \mathrm{Id}) \qquad \Theta \vdash \varphi_1 \equiv \psi_1 : \mathbb{B} \qquad \Theta \vdash \varphi_2 \equiv \psi_2 : \mathbb{B}}{\Theta \vdash \varphi_1 \vee \varphi_2 \equiv \psi_1 \vee \psi_2 : \mathbb{B}}\ \text{Ix}{\equiv}\vee$$

$$\frac{FV(\varphi, \psi) \nsubseteq dom(\Theta - \mathrm{Id}) \qquad \Theta \vdash \varphi \equiv \psi : \mathbb{B}}{\Theta \vdash \neg\varphi \equiv \neg\psi : \mathbb{B}}\ \text{Ix}{\equiv}\neg$$

$$\frac{\begin{array}{c}FV(t_1, t_2, t_1', t_2') \nsubseteq dom(\Theta - \mathrm{Id}) \\ \kappa \in \{\mathbb{N}, \mathbb{Z}\} \qquad \Theta \vdash t_1 \equiv t_1' : \kappa \qquad \Theta \vdash t_2 \equiv t_2' : \kappa\end{array}}{\Theta \vdash t_1 \leq t_2 \equiv t_1' \leq t_2' : \mathbb{B}}\ \text{Ix}{\equiv}{\leq}$$

Figure A.21: Declarative index equivalence

$\boxed{\Theta;[\tau] \vdash \boldsymbol{t} \equiv \boldsymbol{t}' : \kappa}$ Under Θ, index spines $\boldsymbol{t}$ and $\boldsymbol{t}'$ are equivalent and have sort τ returning κ (Inputs: $\Theta, \tau, \boldsymbol{t}, \boldsymbol{t}'$; outputs: κ)

$$\frac{}{\Theta;[\kappa] \vdash \cdot \equiv \cdot : \kappa}\ \text{IxSpine}{\equiv}\text{Nil} \qquad \frac{\Theta \vdash t_0 \equiv t_0' : \kappa_0 \qquad \Theta;[\tau] \vdash \boldsymbol{t} \equiv \boldsymbol{t}' : \kappa}{\Theta;[\kappa_0 \Rightarrow \tau] \vdash t_0, \boldsymbol{t} \equiv t_0', \boldsymbol{t}' : \kappa}\ \text{IxSpine}{\equiv}\text{Entry}$$

$$\frac{k \in \{1,2\} \qquad \Theta;[\tau_k] \vdash \boldsymbol{t} \equiv \boldsymbol{t}' : \kappa}{\Theta;[\tau_1 \times \tau_2] \vdash .k, \boldsymbol{t} \equiv .k, \boldsymbol{t}' : \kappa}\ \text{IxSpine}{\equiv}\text{Proj}_k$$

Figure A.22: Declarative index spine equivalence

$\boxed{\Theta \vdash \Theta_1 \equiv \Theta_2\ \mathsf{ctx}}$ Under input Θ, input logical contexts Θ_1 and Θ_2 are equivalent

$$\frac{}{\Theta \vdash \cdot \equiv \cdot\ \mathsf{ctx}}\ \text{Ctx}{\equiv}\text{Empty} \qquad \frac{\Theta \vdash \Theta_1 \equiv \Theta_2\ \mathsf{ctx}}{\begin{array}{l}\Theta \vdash \Theta_1, a \div \tau \equiv \Theta_2, a \div \tau\ \mathsf{ctx} \\ \Theta \vdash \Theta_1, a \overset{\mathrm{d}}{\div} \tau \equiv \Theta_2, a \overset{\mathrm{d}}{\div} \tau\ \mathsf{ctx}\end{array}}\ \text{Ctx}{\equiv}\text{Var}$$

$$\frac{\Theta \vdash \Theta_1 \equiv \Theta_2\ \mathsf{ctx} \qquad \Theta, \overline{\Theta_1} \vdash \varphi_1 \equiv \varphi_2 : \mathbb{B}}{\Theta \vdash \Theta_1, \varphi_1 \equiv \Theta_2, \varphi_2\ \mathsf{ctx}}\ \text{Ctx}{\equiv}\text{Prop}$$

Figure A.23: Declarative logical context equivalence

$\boxed{\Theta \vdash A \leq^{\pm} B}$ Under input Θ, input type A is a subtype of input B

$$\frac{}{\Theta \vdash 1 \leq^{+} 1}\ \leq^{+}1 \qquad \frac{}{\Theta \vdash 0 \leq^{+} 0}\ \leq^{+}0$$

$$\frac{\Theta \vdash R_1 \leq^{+} R_1' \qquad \Theta \vdash R_2 \leq^{+} R_2'}{\Theta \vdash R_1 \times R_2 \leq^{+} R_1' \times R_2'}\ \leq^{+}\times \qquad \frac{\Theta \vdash P_1 \leq^{+} P_1' \qquad \Theta \vdash P_2 \leq^{+} P_2'}{\Theta \vdash P_1 + P_2 \leq^{+} P_1' + P_2'}\ \leq^{+}+$$

$$\frac{\Theta, \overrightarrow{\varphi} \vdash R \leq^{+} P}{\Theta \vdash R \wedge \overrightarrow{\varphi} \leq^{+} P}\ \leq^{+}\wedge\mathrm{L} \qquad \frac{\Theta, {}^{\mathsf{d}}\Xi \vdash Q \leq^{+} P}{\Theta \vdash \exists^{\mathsf{d}}\Xi.Q \leq^{+} P}\ \leq^{+}\exists\mathrm{L}$$

$$\frac{\Theta \vdash R \leq^{+} R' \qquad \Theta \vdash \overrightarrow{\varphi}\,\mathsf{true}}{\Theta \vdash R \leq^{+} R' \wedge \overrightarrow{\varphi}}\ \leq^{+}\wedge\mathrm{R} \qquad \frac{\dot{\div}\Theta \vdash \sigma : {}^{\mathsf{d}}\Xi \qquad \Theta \vdash R \leq^{+} [\sigma]Q}{\Theta \vdash R \leq^{+} \exists^{\mathsf{d}}\Xi.Q}\ \leq^{+}\exists\mathrm{R}$$

$$\frac{\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)}{\Theta \vdash \left\{\nu : \mu F' \;\middle|\; \mathscr{M}'(F')\right\} \leq^{+} \left\{\nu : \mu F \mid \mathscr{M}(F)\right\}}\ \leq^{+}\mu$$

$$\frac{\Theta \vdash N \leq^{-} N'}{\Theta \vdash \downarrow N \leq^{+} \downarrow N'}\ \leq^{+}\downarrow \qquad \frac{\Theta \vdash P \leq^{+} P'}{\Theta \vdash \uparrow P \leq^{-} \uparrow P'}\ \leq^{-}\uparrow$$

$$\frac{\Theta \vdash L' \leq^{-} L \qquad \Theta \vdash \overrightarrow{\varphi}\,\mathsf{true}}{\Theta \vdash \overrightarrow{\varphi} \supset L' \leq^{-} L}\ \leq^{-}\supset\mathrm{L} \qquad \frac{\dot{\div}\Theta \vdash \sigma : {}^{\mathsf{d}}\Xi \qquad \Theta \vdash [\sigma]M \leq^{-} L}{\Theta \vdash \forall^{\mathsf{d}}\Xi.M \leq^{-} L}\ \leq^{-}\forall\mathrm{L}$$

$$\frac{\Theta, \overrightarrow{\varphi} \vdash N \leq^{-} L}{\Theta \vdash N \leq^{-} \overrightarrow{\varphi} \supset L}\ \leq^{-}\supset\mathrm{R} \qquad \frac{\Theta, {}^{\mathsf{d}}\Xi \vdash N \leq^{-} M}{\Theta \vdash N \leq^{-} \forall^{\mathsf{d}}\Xi.M}\ \leq^{-}\forall\mathrm{R}$$

$$\frac{\Theta \vdash R' \leq^{+} R \qquad \Theta \vdash L \leq^{-} L'}{\Theta \vdash R \to L \leq^{-} R' \to L'}\ \leq^{-}\to$$

Figure A.24: Declarative subtyping

$\boxed{\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)}$ Under Θ, measurement list $\mathscr{M}'(F')$ covers $\mathscr{M}(F)$ (all inputs)

$$\frac{\cdot \vdash \underline{\mathsf{tt}}^{(F')}; F' \leq_{\mathbb{B}} \underline{\mathsf{tt}}^{(F)}; F}{\Theta \vdash \mathscr{M}'(F') \geq \cdot_F}$$

$$\frac{\begin{array}{c}\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F) \qquad (\mathsf{fold}_{F'}\ \alpha')\ v\, t' =_\tau t' \in \mathscr{M}'(F') \\ \cdot \vdash \alpha'; F' \leq_\tau \alpha; F \qquad \overset{\mathsf{d}}{\div}\Theta; [\tau] \vdash t' \equiv t : \kappa \qquad \overset{\mathsf{d}}{\div}\Theta \vdash t' = t\ \mathsf{true}\end{array}}{\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\ v\, t =_\tau t}$$

$\boxed{\Xi \vdash \alpha; F \leq_\tau \beta; G}$ Under Ξ, algebra $\alpha : F(\tau) \Rightarrow \tau$ is a submeasure of $\beta : G(\tau) \Rightarrow \tau$ (inputs: $\Xi, \alpha, F, \tau, \beta, G$)

$$\frac{\begin{array}{ccc}\alpha \circ \mathsf{inj}_1 \doteq \alpha_1 & \beta \circ \mathsf{inj}_1 \doteq \beta_1 & \Xi \vdash \alpha_1; F_1 \leq_\tau \beta_1; G_1 \\ \alpha \circ \mathsf{inj}_2 \doteq \alpha_2 & \beta \circ \mathsf{inj}_2 \doteq \beta_2 & \Xi \vdash \alpha_2; F_2 \leq_\tau \beta_2; G_2\end{array}}{\Xi \vdash \alpha; F_1 \oplus F_2 \leq_\tau \beta; G_1 \oplus G_2}\ \text{Meas}{\leq}\oplus$$

$$\frac{\Xi, {}^{\mathsf{d}}\Xi' \vdash (\top, q) \Rightarrow t; \underline{Q} \otimes \hat{P} \leq_\tau (o', q') \Rightarrow t'; \underline{P} \otimes \hat{P}'}{\Xi \vdash (\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q) \Rightarrow t; \underline{\exists^{\mathsf{d}}\Xi'.\, Q} \otimes \hat{P} \leq_\tau (o', q') \Rightarrow t'; \underline{P} \otimes \hat{P}'}\ \text{Meas}{\leq}\exists\text{L}$$

$$\frac{\overset{\mathsf{d}}{\div}\Xi \vdash \sigma : {}^{\mathsf{d}}\Xi' \qquad \Xi \vdash (\top, q) \Rightarrow t; \underline{Q} \otimes \hat{P} \leq_\tau (\top, q') \Rightarrow [\sigma]t'; \underline{[\sigma]Q'} \otimes \hat{P}'}{\Xi \vdash (\top, q) \Rightarrow t; \underline{Q} \otimes \hat{P} \leq_\tau (\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q') \Rightarrow t'; \underline{\exists^{\mathsf{d}}\Xi'.\, Q'} \otimes \hat{P}'}\ \text{Meas}{\leq}\exists\text{R}$$

$$\frac{\Xi \vdash Q \leq^+ Q' \qquad \Xi \vdash q \Rightarrow t; \hat{P} \leq_\tau q' \Rightarrow t'; \hat{P}'}{\Xi \vdash (\top, q) \Rightarrow t; \underline{Q} \otimes \hat{P} \leq_\tau (\top, q') \Rightarrow t'; \underline{Q'} \otimes \hat{P}'}\ \text{Meas}{\leq}\text{Const}$$

$$\frac{\Xi, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id} \vdash q \Rightarrow t; \hat{I} \leq_\tau q' \Rightarrow t'; \hat{I}}{\Xi \vdash (a, q) \Rightarrow t; \mathrm{Id} \otimes \hat{I} \leq_\tau (a, q') \Rightarrow t'; \mathrm{Id} \otimes \hat{I}}\ \text{Meas}{\leq}\text{Id}$$

$$\frac{\overset{\mathsf{d}}{\div}\Xi \vdash u \equiv t : \tau}{\Xi \vdash () \Rightarrow u; I \leq_\tau () \Rightarrow t; I}\ \text{Meas}{\leq}I$$

$\boxed{\Theta \vdash \mathscr{M}'(F) \geq_\equiv \mathscr{M}(F)}$ $\Theta \vdash \mathscr{M}'(F) \geq \mathscr{M}(F)$ and $\#\mathscr{M}' = \#\mathscr{M}$ and $\mathscr{M}'$, $\mathscr{M}$ are in same order (Inputs: $\Theta, \mathscr{M}'(F), \mathscr{M}(F)$)

$$\frac{\cdot \vdash \underline{\mathsf{tt}}^{(F')}; F' \equiv_{\mathbb{B}} \underline{\mathsf{tt}}^{(F)}; F}{\Theta \vdash \cdot_{F'} \geq_\equiv \cdot_F}$$

$$\frac{\begin{array}{c}\Theta \vdash \mathscr{M}'(F') \geq_\equiv \mathscr{M}(F) \\ \cdot \vdash \alpha'; F' \leq_\tau \alpha; F \qquad \overset{\mathsf{d}}{\div}\Theta; [\tau] \vdash t' \equiv t : \kappa \qquad \overset{\mathsf{d}}{\div}\Theta \vdash t' = t\ \mathsf{true}\end{array}}{\Theta \vdash \mathscr{M}'(F'), (\mathsf{fold}_{F'}\ \alpha')\ v\, t' =_\tau t' \geq_\equiv \mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\ v\, t =_\tau t}$$

Figure A.25: Declarative submeasuring

$\boxed{\Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Theta; R}$ The "$\overrightarrow{\beta}; G$" part of unrolling $\{\nu : \mu F \mid \mathscr{M}(F)\}$ (ins.: $\Xi, \overrightarrow{\beta}, G, \mathscr{M}(F)$) corresponds to the (from outputs ${}^{\mathsf{d}}\Theta$ and R) type $\exists^{\mathsf{d}}\Theta.\,(R \wedge {}^{\mathsf{d}}\Theta)$

$$\frac{\begin{array}{cc}\overrightarrow{\beta} \circ \mathsf{inj}_1 \doteq \overrightarrow{\beta_1} & \Xi \vdash \wr \overrightarrow{\beta_1}; G_1; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Theta_1; R_1 \\ \overrightarrow{\beta} \circ \mathsf{inj}_2 \doteq \overrightarrow{\beta_2} & \Xi \vdash \wr \overrightarrow{\beta_2}; G_2; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Theta_2; R_2\end{array}}{\Xi \vdash \wr \overrightarrow{\beta}; G_1 \oplus G_2; \mathscr{M}(F) \wr \doteq \cdot; (\exists^{\mathsf{d}}\Theta_1.\,(R_1 \wedge {}^{\mathsf{d}}\Theta_1)) + (\exists^{\mathsf{d}}\Theta_2.\,(R_2 \wedge {}^{\mathsf{d}}\Theta_2))}\ \wr\oplus\wr$$

$$\frac{\begin{array}{c}({}^{\mathsf{d}}\Xi' \text{ may be } \cdot \text{ and } \overrightarrow{\varphi} \text{ may be } \cdot) \\ \overrightarrow{\beta} \rightsquigarrow \overrightarrow{\beta'} \qquad \Xi, {}^{\mathsf{d}}\Xi' \vdash \wr \overrightarrow{\beta'}; \hat{P}; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Theta_0; R_0\end{array}}{\Xi \vdash \wr \overrightarrow{\beta}; \underline{\exists^{\mathsf{d}}\Xi'.\,R' \wedge \overrightarrow{\varphi}} \otimes \hat{P}; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Xi', {}^{\mathsf{d}}\Theta_0, \overrightarrow{\varphi}; R' \times R_0}\ \wr\text{Const}\wr$$

$$\frac{\begin{array}{c}\overrightarrow{a \overset{\mathsf{d}\cdot}{\div} \tau} = \overrightarrow{a} \overset{\mathsf{d}\cdot}{\div} \mathscr{M}(F) \\ \Xi, \overrightarrow{a \overset{\mathsf{d}\cdot}{\div} \tau}, a\,\mathrm{Id} \vdash \wr \overrightarrow{q \Rightarrow t'}; \hat{I}; \mathscr{M}(F) \wr \doteq \Xi'', \overrightarrow{\psi''}; R'' \\ \Xi; \Xi''; zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash \overrightarrow{\psi''} \rightsquigarrow \breve{\Xi}_1; \mathscr{M}_1(F); \overrightarrow{\psi'} \\ \Xi; \Xi''; zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash R'' \rightsquigarrow \breve{\Xi}_2; \mathscr{M}_2(F); R' \\ \breve{\Xi} = \breve{\Xi}_1 \cup \breve{\Xi}_2 \qquad \mathscr{M}'(F) = \mathscr{M}_1(F) \cup \mathscr{M}_2(F) \\ dom(\Xi') \cap dom(\Xi, \overrightarrow{a \overset{\mathsf{d}\cdot}{\div} \tau}, \Xi'', \breve{\Xi}) = \emptyset \qquad \rho = \Xi'/\breve{\Xi} \text{ is a variable renaming}\end{array}}{\Xi \vdash \wr \overrightarrow{(a,q) \Rightarrow t'}; \mathrm{Id} \otimes \hat{I}; \mathscr{M}(F) \wr \doteq \Xi', \Xi'', [\rho]\overrightarrow{\psi'}; \left\{\nu : \mu F \;\middle|\; [\rho]\mathscr{M}'(F)\right\} \times [\rho]R'}\ \wr\mathrm{Id}\wr$$

$$\frac{\overrightarrow{t'} \mathbin{@} \mathscr{M}(F) \doteq \overrightarrow{\varphi}}{\Xi \vdash \wr \overrightarrow{() \Rightarrow t'}; I; \mathscr{M}(F) \wr \doteq \overrightarrow{\varphi}; 1}\ \wr I\wr$$

where

$$\frac{}{\cdot \circ \mathsf{inj}_k \doteq \cdot} \qquad \frac{\overrightarrow{\beta} \circ \mathsf{inj}_k \doteq \overrightarrow{\beta'} \qquad \beta_n \circ \mathsf{inj}_k \doteq \beta_{nk}}{\overrightarrow{\beta}, \beta_n \circ \mathsf{inj}_k \doteq \overrightarrow{\beta'}, \beta_{nk}}$$

$$\frac{}{\cdot \rightsquigarrow \cdot} \qquad \frac{\overrightarrow{\beta} \rightsquigarrow \overrightarrow{\beta'}}{\overrightarrow{\beta}, (\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q) \Rightarrow t' \rightsquigarrow \overrightarrow{\beta'}, q \Rightarrow t' \qquad \overrightarrow{\beta}, (\top, q) \Rightarrow t' \rightsquigarrow \overrightarrow{\beta'}, q \Rightarrow t'}$$

$$\frac{}{\cdot \mathbin{@} \cdot \doteq \cdot} \qquad \frac{\overrightarrow{u} \mathbin{@} \mathscr{M}(F) \doteq \overrightarrow{\varphi}}{(\overrightarrow{u}, t') \mathbin{@} (\mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\, \nu\, \boldsymbol{t} =_\tau t) \doteq \overrightarrow{\varphi}, (t = \langle t' \mid \boldsymbol{t} \rangle)}$$

$$\cdot \overset{\mathsf{d}\cdot}{\div} \cdot_F = \cdot$$

$$(\overrightarrow{a}, a_k) \overset{\mathsf{d}\cdot}{\div} (\mathscr{M}(F), (\mathsf{fold}_F\ \alpha_k)\, \nu\, \boldsymbol{t}_k =_{\tau_k} t_k) = (\overrightarrow{a} \overset{\mathsf{d}\cdot}{\div} \mathscr{M}(F)), a_k \overset{\mathsf{d}\cdot}{\div} \tau_k$$

$\Xi; \Xi'; \overrightarrow{(a, (\mathsf{fold}_F\ \alpha)\, \nu\, _ =_\tau _)} \vdash \mathscr{O} \rightsquigarrow \breve{\Xi}; \mathscr{M}'(F); \mathscr{O}'$ is defined in Figure A.27.

Figure A.26: Unrolling

$\boxed{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \mathscr{O} \rightsquigarrow \check{\Xi};\mathscr{M}'(F);\mathscr{O}'}$

Presupposes $F_k = F$ and $FV(F_k,\alpha_k)=\emptyset$ for all $(a_k,(\!|\alpha_k|\!)_{F_k}\,\nu_=_{\tau_k}\,_) \in \overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)}$; and $\mathscr{O}$ is well-formed under $\Xi,\Xi',a\overset{\mathrm{d}\cdot}{\div}\tau,a\,\mathrm{Id}$
(Inputs: left of $\rightsquigarrow$; outputs: right of $\rightsquigarrow$)

$$\frac{a_k \notin FV(t) \text{ for all } (a_k,_) \in \overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)}}{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash t \rightsquigarrow \cdot;\cdot_F;t}$$

$$\frac{(a_k,(\!|\alpha_k|\!)_F\,\nu_=_{\tau_k}\,_) \in \overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)}}{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash a_k \rightsquigarrow \breve{a}_k^{a_k(\cdot)} \overset{\mathrm{d}\cdot}{\div} \tau_k;(\!|\alpha_k|\!)_F\,\nu\cdot=_{\tau_k}\breve{a}_k^{a_k(\cdot)};\breve{a}_k^{a_k(\cdot)}}$$

$$\frac{\begin{array}{c}(a_k,(\!|\alpha_k|\!)_F\,\nu_=_{\tau_k}\,_) \in \overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \\ \Xi;\Xi';(a_k,(\!|\alpha_k|\!)_F\,\nu_=_{\tau_k}\,_) \vdash \boldsymbol{u} \rightsquigarrow \check{\Xi};\mathscr{M}'(F);\boldsymbol{u}' \qquad \overset{\mathrm{d}\cdot}{\div}(\Xi,\Xi',\check{\Xi});[\tau_k] \vdash \boldsymbol{u}' : \kappa\end{array}}{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash a_k(\boldsymbol{u}) \rightsquigarrow \check{\Xi},\breve{a}_k^{a_k(\boldsymbol{u}')} \overset{\mathrm{d}\cdot}{\div} \kappa;\mathscr{M}',(\!|\alpha_k|\!)_F\,\nu\,\boldsymbol{u}'=_{\tau_k}\breve{a}_k^{a_k(\boldsymbol{u}')};\breve{a}_k^{a_k(\boldsymbol{u}')}}$$

$$\frac{\begin{array}{c}a_k \neq b \text{ for all } (a_k,_) \in \overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \\ \Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \boldsymbol{u} \rightsquigarrow \check{\Xi};\mathscr{M}';\boldsymbol{u}'\end{array}}{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash b(\boldsymbol{u}) \rightsquigarrow \check{\Xi};\mathscr{M}';b(\boldsymbol{u}')}$$

$$\frac{op \in \{\neg_\} \qquad \Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash t \rightsquigarrow \check{\Xi};\mathscr{M}';t'}{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash op\,t \rightsquigarrow \check{\Xi};\mathscr{M}';op\,t'}$$

$$\frac{\begin{array}{c}op \in \{-+-,---,-=-,-\leq-,-\wedge-,-\vee-,(-,-)\} \\ \Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash t_1 \rightsquigarrow \check{\Xi}_1;\mathscr{M}_1;t_1' \\ \Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash t_2 \rightsquigarrow \check{\Xi}_2;\mathscr{M}_2;t_2'\end{array}}{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash t_1\;op\;t_2 \rightsquigarrow \check{\Xi}_1\cup\check{\Xi}_2;\mathscr{M}_1\cup\mathscr{M}_2;t_1'\;op\;t_2'}$$

$$\overline{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \cdot \rightsquigarrow \cdot;\cdot_F;\cdot}$$

$$\frac{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash t \rightsquigarrow \check{\Xi};\mathscr{M};t' \qquad \Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \boldsymbol{t} \rightsquigarrow \check{\Xi}';\mathscr{M}';\boldsymbol{t}'}{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash t,\boldsymbol{t} \rightsquigarrow \check{\Xi}\cup\check{\Xi}';\mathscr{M}\cup\mathscr{M}';t',\boldsymbol{t}'}$$

$$\frac{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \boldsymbol{t} \rightsquigarrow \check{\Xi};\mathscr{M}';\boldsymbol{t}'}{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash .k,\boldsymbol{t} \rightsquigarrow \check{\Xi};\mathscr{M}';.k,\boldsymbol{t}'}$$

$$\frac{\begin{array}{c}\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \mathscr{M}_0 \rightsquigarrow \check{\Xi};\mathscr{M};\mathscr{M}_0' \\ \Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash R \rightsquigarrow \check{\Xi}';\mathscr{M}';R'\end{array}}{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \{\nu:\mu F \mid \mathscr{M}_0\}\times R \rightsquigarrow \check{\Xi}\cup\check{\Xi}';\mathscr{M}\cup\mathscr{M}';\left\{\nu:\mu F \;\middle|\; \mathscr{M}_0'\right\}\times R'}$$

$$\overline{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash 1 \rightsquigarrow \cdot;\cdot_F;1} \qquad \overline{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \cdot_F \rightsquigarrow \cdot;\cdot_F;\cdot_F}$$

$$\frac{\begin{array}{c}\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \mathscr{M}_0 \rightsquigarrow \check{\Xi};\mathscr{M};\mathscr{M}_0' \\ \Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \boldsymbol{t} \rightsquigarrow \check{\Xi}';\mathscr{M}';\boldsymbol{t}'\end{array}}{\Xi;\Xi';\overrightarrow{(a,(\!|\alpha|\!)_F\,\nu_=_\tau\,_)} \vdash \mathscr{M}_0,(\mathsf{fold}_F\;\alpha)\,\nu\,\boldsymbol{t}=_\tau t \rightsquigarrow \check{\Xi}\cup\check{\Xi}';\mathscr{M}\cup\mathscr{M}';\mathscr{M}_0',(\mathsf{fold}_F\;\alpha)\,\nu\,\boldsymbol{t}'=_\tau t}$$

Figure A.27: A judgment (called "liftapps") used for $\wr\mathrm{Id}\wr$

$\boxed{\Theta;\Gamma \vdash h \Rightarrow P}$ Under inputs Θ and Γ, input head h synthesizes (output) type P

$$\frac{(x : R) \in \Gamma}{\Theta;\Gamma \vdash x \Rightarrow R}\ \text{Decl}\Rightarrow\text{Var} \qquad \frac{\overline{\Theta} \vdash P\ \mathsf{type}[\xi_P] \qquad \Theta;\Gamma \vdash v \Leftarrow P}{\Theta;\Gamma \vdash (v : P) \Rightarrow P}\ \text{Decl}\Rightarrow\text{ValAnnot}$$

$\boxed{\Theta;\Gamma \vdash g \Rightarrow \uparrow P}$ Under inputs Θ and Γ, input bound expression g synthesizes (output) type $\uparrow P$

$$\frac{\Theta;\Gamma \vdash h \Rightarrow \downarrow N \qquad \Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P}{\Theta;\Gamma \vdash h(s) \Rightarrow \uparrow P}\ \text{Decl}\Rightarrow\text{App}$$

$$\frac{\overline{\Theta} \vdash P\ \mathsf{type}[\xi_P] \qquad \Theta;\Gamma \vdash e \Leftarrow \uparrow P}{\Theta;\Gamma \vdash (e : \uparrow P) \Rightarrow \uparrow P}\ \text{Decl}\Rightarrow\text{ExpAnnot}$$

Figure A.28: Declarative head and bound expression type synthesis

$\boxed{\Theta;\Gamma \vdash v \Leftarrow P}$ Under inputs Θ and Γ, input value v checks against input type P

$$\frac{(x : R') \in \Gamma \qquad \Theta \vdash R' \leq^{+} R}{\Theta;\Gamma \vdash x \Leftarrow R}\ \text{Decl}\Leftarrow\text{Var} \qquad \frac{}{\Theta;\Gamma \vdash \langle\rangle \Leftarrow 1}\ \text{Decl}\Leftarrow 1$$

$$\frac{\Theta;\Gamma \vdash v_1 \Leftarrow R_1 \qquad \Theta;\Gamma \vdash v_2 \Leftarrow R_2}{\Theta;\Gamma \vdash \langle v_1, v_2\rangle \Leftarrow R_1 \times R_2}\ \text{Decl}\Leftarrow\times \qquad \frac{\Theta;\Gamma \vdash v \Leftarrow P_k}{\Theta;\Gamma \vdash \text{inj}_k\, v \Leftarrow P_1 + P_2}\ \text{Decl}\Leftarrow+_k$$

$$\frac{\dot{\div}\Theta \vdash \sigma : {}^{\mathsf{d}}\Xi \qquad \Theta;\Gamma \vdash v \Leftarrow [\sigma]Q}{\Theta;\Gamma \vdash v \Leftarrow (\exists^{\mathsf{d}}\Xi.\, Q)}\ \text{Decl}\Leftarrow\exists \qquad \frac{\Theta \vdash \overrightarrow{\varphi}\ \mathsf{true} \qquad \Theta;\Gamma \vdash v \Leftarrow R}{\Theta;\Gamma \vdash v \Leftarrow R \wedge \overrightarrow{\varphi}}\ \text{Decl}\Leftarrow\wedge$$

$$\frac{\begin{array}{c}\nexists x.\, v = \overrightarrow{\text{inj}_{k_i}}^{\,i}\left(\overrightarrow{\langle __j, - \rangle}^{\,j}\, x\right) \qquad \mathscr{M}(F) \rightsquigarrow \overrightarrow{\alpha};\overrightarrow{\tau} \\ \dot{\div}\Theta \vdash \wr\, \overrightarrow{\alpha};F;\mathscr{M}(F) \,\wr \doteq {}^{\mathsf{d}}\Theta;R \qquad \Theta;\Gamma \vdash v \Leftarrow \exists^{\mathsf{d}}\Theta.\, (R \wedge {}^{\mathsf{d}}\Theta)\end{array}}{\Theta;\Gamma \vdash \mathsf{into}(v) \Leftarrow \{\nu : \mu F \mid \mathscr{M}(F)\}}\ \text{Decl}\Leftarrow\mu$$

$$\frac{\Theta;\Gamma \vdash e \Leftarrow N}{\Theta;\Gamma \vdash \{e\} \Leftarrow \downarrow N}\ \text{Decl}\Leftarrow\downarrow$$

where

$$\frac{}{\cdot_F \rightsquigarrow \cdot;\cdot} \qquad \frac{\mathscr{M}(F) \rightsquigarrow \overrightarrow{\alpha};\overrightarrow{\tau}}{\mathscr{M}(F), (\mathsf{fold}_F\ \alpha_n)\, \nu\ _ =_{\tau_n} _ \rightsquigarrow \overrightarrow{\alpha}, \alpha_n; \overrightarrow{\tau}, \tau_n}$$

Figure A.29: Declarative value type checking

$\boxed{\Theta;\Gamma \vdash e \Leftarrow N}$ Under inputs Θ and Γ, input expression e checks against input type N

$$\frac{\Theta;\Gamma \vdash v \Leftarrow P}{\Theta;\Gamma \vdash \mathsf{return}\, v \Leftarrow \uparrow P}\ \mathrm{Decl}{\Leftarrow}{\uparrow}$$

$$\frac{\Theta;\Gamma \vdash g \Rightarrow \uparrow(\exists^{\mathsf{d}}\Xi.R \wedge \overrightarrow{\psi}) \qquad \Theta,{}^{\mathsf{d}}\Xi,\overrightarrow{\psi};\Gamma,x:R \vdash e \Leftarrow L}{\Theta;\Gamma \vdash \mathsf{let}\ x = g;\ e \Leftarrow L}\ \mathrm{Decl}{\Leftarrow}\mathsf{let}$$

$$\frac{\Theta;\Gamma \vdash h \Rightarrow P \qquad \Theta;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow L}{\Theta;\Gamma \vdash \mathsf{match}\ h\ \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow L}\ \mathrm{Decl}{\Leftarrow}\mathsf{match}$$

$$\frac{\Theta;\Gamma,x:R \vdash e \Leftarrow L}{\Theta;\Gamma \vdash \lambda x.e \Leftarrow R \to L}\ \mathrm{Decl}{\Leftarrow}\lambda \qquad \frac{\Theta \vdash \mathsf{ff}\ \mathsf{true}}{\Theta;\Gamma \vdash \mathsf{unreachable} \Leftarrow L}\ \mathrm{Decl}{\Leftarrow}\mathrm{Unreachable}$$

$$\frac{\begin{array}{c}\Theta \vdash \forall a \overset{\mathsf{d}\cdot}{\div} \mathbb{N},{}^{\mathsf{d}}\Xi.M \leq^{-} L \\ \Theta,a \div \mathbb{N};\Gamma,x:{\downarrow}\forall a' \overset{\mathsf{d}\cdot}{\div} \mathbb{N},{}^{\mathsf{d}}\Xi.a' < a \supset [a'/a]M \vdash e \Leftarrow \forall^{\mathsf{d}}\Xi.M\end{array}}{\Theta;\Gamma \vdash \mathsf{rec}\, x : (\forall a \overset{\mathsf{d}\cdot}{\div} \mathbb{N},{}^{\mathsf{d}}\Xi.M).\ e \Leftarrow L}\ \mathrm{Decl}{\Leftarrow}\mathsf{rec}$$

$$\frac{\Theta,{}^{\mathsf{d}}\Xi;\Gamma \vdash e \Leftarrow M}{\Theta;\Gamma \vdash e \Leftarrow \forall^{\mathsf{d}}\Xi.M}\ \mathrm{Decl}{\Leftarrow}\forall \qquad \frac{\Theta,\overrightarrow{\varphi};\Gamma \vdash e \Leftarrow L}{\Theta;\Gamma \vdash e \Leftarrow \overrightarrow{\varphi} \supset L}\ \mathrm{Decl}{\Leftarrow}\supset$$

Figure A.30: Declarative expression type checking

$\boxed{\Theta;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}$ Under Θ and Γ, patterns r_i match against type P and branch expressions e_i check against type N (all inputs)

$$\frac{\Theta, {}^{\mathsf{d}}\Xi;\Gamma;[Q] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}{\Theta;\Gamma;[\exists^{\mathsf{d}}\Xi.\, Q] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}\ \text{DeclMatch}\exists$$

$$\frac{\Theta, \overrightarrow{\varphi};\Gamma;[R] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}{\Theta;\Gamma;[R \wedge \overrightarrow{\varphi}] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}\ \text{DeclMatch}\wedge \qquad \frac{\Theta;\Gamma \vdash e \Leftarrow N}{\Theta;\Gamma;[1] \vdash \{\langle\rangle \Rightarrow e\} \Leftarrow N}\ \text{DeclMatch1}$$

$$\frac{\Theta;\Gamma, x_1 : R_1, x_2 : R_2 \vdash e \Leftarrow N}{\Theta;\Gamma;[R_1 \times R_2] \vdash \{\langle x_1, x_2\rangle \Rightarrow e\} \Leftarrow N}\ \text{DeclMatch}\times$$

$$\frac{\Theta, {}^{\mathsf{d}}\Xi_1, \overrightarrow{\psi}_1;\Gamma, x_1 : R_1 \vdash e_1 \Leftarrow N \qquad \Theta, {}^{\mathsf{d}}\Xi_2, \overrightarrow{\psi}_2;\Gamma, x_2 : R_2 \vdash e_2 \Leftarrow N}{\Theta;\Gamma;[(\exists^{\mathsf{d}}\Xi_1.\, R_1 \wedge \overrightarrow{\psi}_1) + (\exists^{\mathsf{d}}\Xi_2.\, R_2 \wedge \overrightarrow{\psi}_2)] \vdash \{\mathtt{inj}_1\, x_1 \Rightarrow e_1 \mid \mathtt{inj}_2\, x_2 \Rightarrow e_2\} \Leftarrow N}\ \text{DeclMatch}+$$

$$\frac{}{\Theta;\Gamma;[0] \vdash \{\} \Leftarrow N}\ \text{DeclMatch0}$$

$$\frac{\begin{array}{c}\mathscr{M}(F) \rightsquigarrow \overrightarrow{\alpha};\overrightarrow{\tau} \\ \overset{\mathsf{d}}{\div}\Theta \vdash \wr\, \overrightarrow{\alpha};F;\mathscr{M}(F)\, \wr \doteq {}^{\mathsf{d}}\Theta;R \qquad \Theta, {}^{\mathsf{d}}\Theta;\Gamma, x : R \vdash e \Leftarrow N\end{array}}{\Theta;\Gamma;[\{\nu : \mu F \mid \mathscr{M}(F)\}] \vdash \{\mathtt{into}(x) \Rightarrow e\} \Leftarrow N}\ \text{DeclMatch}\mu$$

where

$$\frac{}{\cdot_F \rightsquigarrow \cdot;\cdot} \qquad \frac{\mathscr{M}(F) \rightsquigarrow \overrightarrow{\alpha};\overrightarrow{\tau}}{\mathscr{M}(F), (\mathsf{fold}_F\ \alpha_n)\ \nu\ _ =_{\tau_n} _ \rightsquigarrow \overrightarrow{\alpha}, \alpha_n;\overrightarrow{\tau}, \tau_n}$$

Figure A.31: Declarative pattern matching

$\boxed{\Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P}$ Under inputs Θ and Γ, if a head of type $\downarrow N$ (input: N) is applied to the spine s (input), then it will return a result of type $\uparrow P$ (output)

$$\frac{\overset{\mathsf{d}}{\div}\Theta \vdash \sigma : {}^{\mathsf{d}}\Xi \qquad \Theta;\Gamma;[[\sigma]M] \vdash s \Rightarrow \uparrow P}{\Theta;\Gamma;[\forall^{\mathsf{d}}\Xi.M] \vdash s \Rightarrow \uparrow P}\ \text{DeclSpine}\forall$$

$$\frac{\Theta \vdash \overrightarrow{\varphi}\ \mathsf{true} \qquad \Theta;\Gamma;[L] \vdash s \Rightarrow \uparrow P}{\Theta;\Gamma;[\overrightarrow{\varphi} \supset L] \vdash s \Rightarrow \uparrow P}\ \text{DeclSpine}\supset$$

$$\frac{\Theta;\Gamma \vdash v \Leftarrow R \qquad \Theta;\Gamma;[L] \vdash s \Rightarrow \uparrow P}{\Theta;\Gamma;[R \to L] \vdash v, s \Rightarrow \uparrow P}\ \text{DeclSpineApp}$$

$$\frac{}{\Theta;\Gamma;[\uparrow P] \vdash \cdot \Rightarrow \uparrow P}\ \text{DeclSpineNil}$$

Figure A.32: Declarative spine typing

Appendix A.4 Unrefined System and Its Denotational Semantics

Program variables	x, y, z		
Expressions	e	$::=$	$\mathtt{return}\, v \mid \mathtt{let}\, x = g;\, e \mid \mathtt{match}\, h\, \{r_i \Rightarrow e_i\}_{i \in I} \mid \lambda x.\, e \mid \mathtt{rec}\, x.\, e$
Values	v	$::=$	$x \mid \langle\rangle \mid \langle v, v\rangle \mid \mathtt{inj}_1\, v \mid \mathtt{inj}_2\, v \mid \mathtt{into}(v) \mid \{e\}$
Heads	h	$::=$	$x \mid (v : P)$
Bound expressions	g	$::=$	$h(s) \mid (e : \uparrow P)$
Spines	s	$::=$	$\cdot \mid v, s$
Patterns	r	$::=$	$\mathtt{into}(x) \mid \langle\rangle \mid \langle x, y\rangle \mid \mathtt{inj}_1\, x \mid \mathtt{inj}_2\, x$
Unrefined positive types	P, Q, R	$::=$	$1 \mid P \times Q \mid 0 \mid P + Q \mid \downarrow N \mid \mu F$
Unrefined negative types	N, M, L	$::=$	$P \to N \mid \uparrow P$
Types	A, B, C	$::=$	$P \mid N$
Unrefined functors	F, G, H	$::=$	$\hat{P} \mid F \oplus F$
	$\hat{P}$	$::=$	$\hat{I} \mid \underline{P} \otimes \hat{P}$
	$\hat{I}$	$::=$	$I \mid \mathrm{Id} \otimes \hat{I}$
	$\hat{B}$	$::=$	$\underline{P} \mid \mathrm{Id}$
	$\mathscr{F}$	$::=$	$F \mid \hat{B}$

Figure A.33: Unrefined syntax

$\boxed{\vdash G[\mu F] \doteq P}$ Functor G applied to "type" μF corresponds to the (output) type P (inputs: G and F)

$$\frac{\vdash G[\mu F] \doteq P \qquad \vdash H[\mu F] \doteq Q}{\vdash (G \oplus H)[\mu F] \doteq P + Q}\ \text{UnrefUnroll}\oplus$$

$$\frac{\vdash \hat{P}[\mu F] \doteq P}{\vdash (\underline{Q} \otimes \hat{P})[\mu F] \doteq Q \times P}\ \text{UnrefUnrollConst} \qquad \frac{\vdash \hat{I}[\mu F] \doteq P}{\vdash (\mathrm{Id} \otimes \hat{I})[\mu F] \doteq \mu F \times P}\ \text{UnrefUnrollId}$$

$$\frac{}{\vdash I[\mu F] \doteq 1}\ \text{UnrefUnroll}I$$

Figure A.34: Unrefined unrolling

$\boxed{\Gamma \vdash h \Rightarrow P}$ Under input Γ, head h synthesizes (output) type P

$$\frac{(x : P) \in \Gamma}{\Gamma \vdash x \Rightarrow P}\ \text{Unref}{\Rightarrow}\text{Var} \qquad \frac{\Gamma \vdash v \Leftarrow P}{\Gamma \vdash (v : P) \Rightarrow P}\ \text{Unref}{\Rightarrow}\text{ValAnnot}$$

$\boxed{\Gamma \vdash g \Rightarrow \uparrow P}$ Under input Γ, input bound expression g synthesizes type $\uparrow P$ (output)

$$\frac{\Gamma \vdash h \Rightarrow \downarrow N \qquad \Gamma; [N] \vdash s \Rightarrow \uparrow P}{\Gamma \vdash h(s) \Rightarrow \uparrow P}\ \text{Unref}{\Rightarrow}\text{App} \qquad \frac{\Gamma \vdash e \Leftarrow \uparrow P}{\Gamma \vdash (e : \uparrow P) \Rightarrow \uparrow P}\ \text{Unref}{\Rightarrow}\text{ExpAnnot}$$

$\boxed{\Gamma \vdash v \Leftarrow P}$ Under input Γ, input value v checks against input type P

$$\frac{(x : P) \in \Gamma}{\Gamma \vdash x \Leftarrow P}\ \text{Unref}{\Leftarrow}\text{Var} \qquad \frac{}{\Gamma \vdash \langle\rangle \Leftarrow 1}\ \text{Unref}{\Leftarrow}1$$

$$\frac{\Gamma \vdash v_1 \Leftarrow P_1 \qquad \Gamma \vdash v_2 \Leftarrow P_2}{\Gamma \vdash \langle v_1, v_2 \rangle \Leftarrow P_1 \times P_2}\ \text{Unref}{\Leftarrow}\times \qquad \frac{\Gamma \vdash v \Leftarrow P_k}{\Gamma \vdash \text{inj}_k\, v \Leftarrow P_1 + P_2}\ \text{Unref}{\Leftarrow}+_k$$

$$\frac{\vdash F[\mu F] \doteq P \qquad \Gamma \vdash v \Leftarrow P}{\Gamma \vdash \texttt{into}(v) \Leftarrow \mu F}\ \text{Unref}{\Leftarrow}\mu \qquad \frac{\Gamma \vdash e \Leftarrow N}{\Gamma \vdash \{e\} \Leftarrow \downarrow N}\ \text{Unref}{\Leftarrow}\downarrow$$

$\boxed{\Gamma \vdash e \Leftarrow N}$ Under input Γ, input expression e checks against input type N

$$\frac{\Gamma \vdash v \Leftarrow P}{\Gamma \vdash \texttt{return}\, v \Leftarrow \uparrow P}\ \text{Unref}{\Leftarrow}\uparrow \qquad \frac{\Gamma \vdash g \Rightarrow \uparrow P \qquad \Gamma, x : P \vdash e \Leftarrow N}{\Gamma \vdash \texttt{let}\ x = g;\ e \Leftarrow N}\ \text{Unref}{\Leftarrow}\texttt{let}$$

$$\frac{\Gamma \vdash h \Rightarrow P \qquad \Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N}{\Gamma \vdash \texttt{match}\ h\ \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N}\ \text{Unref}{\Leftarrow}\texttt{match}$$

$$\frac{\Gamma, x : P \vdash e \Leftarrow N}{\Gamma \vdash \lambda x.\, e \Leftarrow P \to N}\ \text{Unref}{\Leftarrow}\lambda \qquad \frac{\Gamma, x : \downarrow N \vdash e \Leftarrow N}{\Gamma \vdash \texttt{rec}\, x.\, e \Leftarrow N}\ \text{Unref}{\Leftarrow}\texttt{rec}$$

Figure A.35: Unrefined declarative typing

$\boxed{\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}$ Under Γ, patterns r_i match against type P and branch expressions e_i check against type N (all inputs)

$$\frac{\Gamma \vdash e \Leftarrow N}{\Gamma;[1] \vdash \{\langle\rangle \Rightarrow e\} \Leftarrow N}\ \text{UnrefMatch1} \qquad \frac{\Gamma, x_1 : P_1, x_2 : P_2 \vdash e \Leftarrow N}{\Gamma;[P_1 \times P_2] \vdash \{\langle x_1, x_2\rangle \Rightarrow e\} \Leftarrow N}\ \text{UnrefMatch}\times$$

$$\frac{\Gamma, x_1 : P_1 \vdash e_1 \Leftarrow N \qquad \Gamma, x_2 : P_2 \vdash e_2 \Leftarrow N}{\Gamma;[P_1 + P_2] \vdash \{\mathrm{inj}_1\, x_1 \Rightarrow e_1 \mid \mathrm{inj}_2\, x_2 \Rightarrow e_2\} \Leftarrow N}\ \text{UnrefMatch}+$$

$$\frac{}{\Gamma;[0] \vdash \{\} \Leftarrow N}\ \text{UnrefMatch0} \qquad \frac{\vdash F[\mu F] \mathrel{\mathring{=}} P \qquad \Gamma, x : P \vdash e \Leftarrow N}{\Gamma;[\mu F] \vdash \{\mathtt{into}(x) \Rightarrow e\} \Leftarrow N}\ \text{UnrefMatch}\mu$$

$\boxed{\Gamma;[N] \vdash s \Rightarrow \uparrow P}$ Under input Γ, if input spine s is applied to a head of type $\downarrow N$ (input: N), then it will produce a result of type $\uparrow P$ (output)

$$\frac{\Gamma \vdash v \Leftarrow Q \qquad \Gamma;[N] \vdash s \Rightarrow \uparrow P}{\Gamma;[Q \to N] \vdash v, s \Rightarrow \uparrow P}\ \text{UnrefSpineApp} \qquad \frac{}{\Gamma;[\uparrow P] \vdash \cdot \Rightarrow \uparrow P}\ \text{UnrefSpineNil}$$

Figure A.36: Unrefined matching and spines

Unrefined syntactic substitutions $\quad \sigma ::= \cdot \mid \sigma, v : P/x$

$\boxed{\Gamma_0 \vdash \sigma : \Gamma}$ Under input Γ_0, we know input σ is a syntactic substitution for variables in input Γ

$$\frac{}{\Gamma_0 \vdash \cdot : \cdot}\ \text{UnrefEmpty}\sigma \qquad \frac{\Gamma_0 \vdash \sigma : \Gamma \qquad \Gamma_0 \vdash v \Leftarrow P \qquad x \notin dom(\Gamma)}{\Gamma_0 \vdash \sigma, v : P/x : \Gamma, x : P}\ \text{UnrefVal}\sigma$$

Figure A.37: Unrefined syntactic substitution

$$\text{Semantic substitutions} \quad \delta \ ::= \ \cdot \mid \delta, V/x$$

$\boxed{\vdash \delta : \Gamma}$ We know that input δ is a semantic substitution for variables in input Γ

$$\frac{}{\vdash \cdot : \cdot}\ \text{UnrefEmpty}\delta \qquad \frac{\vdash \delta : \Gamma \qquad V \in [\![P]\!] \qquad x \notin dom(\Gamma)}{\vdash \delta, V/x : \Gamma, x : P}\ \text{UnrefVal}\delta$$

$$[\![\Gamma]\!] = \{\delta \mid \vdash \delta : \Gamma\}$$

Figure A.38: Unrefined semantic substitution

$$
\begin{aligned}
[\![P]\!] &: \mathbf{Cpo} \\
[\![1]\!] &= (\{\bullet\}, \sqsubseteq_{\{\bullet\}}) \\
[\![P \times Q]\!] &= ([\![P]\!] \times [\![Q]\!], \sqsubseteq_{[\![P]\!]\times[\![Q]\!]}) \\
&\quad \text{where } (V_{11}, V_{12}) \sqsubseteq_{D_1 \times D_2} (V_{21}, V_{22}) \text{ iff } V_{11} \sqsubseteq_{D_1} V_{12} \text{ and } V_{21} \sqsubseteq_{D_2} V_{22} \\
[\![0]\!] &= (\emptyset, \sqsubseteq_{\emptyset}) \\
[\![P + Q]\!] &= ([\![P]\!] \uplus [\![Q]\!], \sqsubseteq_{[\![P]\!]\uplus[\![Q]\!]}) \\
&\quad \text{where } (j, V_{1j}) \sqsubseteq_{D_1 \uplus D_2} (j, V_{2j}) \text{ iff } V_{1j} \sqsubseteq_{D_j} V_{2j} \\
[\![\downarrow N]\!] &= ([\![N]\!], \sqsubseteq_{[\![N]\!]}) \\
[\![\mu F]\!] &= (\cup_{k \in \mathbb{N}} [\![F]\!]^k \emptyset, \sqsubseteq_{\mu [\![F]\!]}) \\
&\quad \text{where } V_1 \sqsubseteq_{\mu [\![F]\!]} V_2 \text{ iff there exists } k \in \mathbb{N} \text{ such that } V_1 \sqsubseteq_{[\![F]\!]^{k+1}\emptyset} V_2 \\
&\quad \text{and } \sqsubseteq_{[\![F_1 \oplus F_2]\!]X} = \sqsubseteq_{[\![F_1]\!]X \uplus [\![F_2]\!]X} \\
&\quad \text{and } \sqsubseteq_{[\![\hat{B} \otimes \hat{P}]\!]X} = \sqsubseteq_{[\![\hat{B}]\!]X \times [\![\hat{P}]\!]X} \\
&\quad \text{and } \sqsubseteq_{[\![\mathrm{Id}]\!]X} = \sqsubseteq_X \\
&\quad \text{and } \sqsubseteq_{[\![I]\!]X} = \sqsubseteq_{\{\bullet\}} \\
&\quad \text{and } \sqsubseteq_{[\![\underline{Q}]\!]X} = \sqsubseteq_{[\![Q]\!]}
\end{aligned}
$$

$$
\begin{aligned}
[\![N]\!] &: \mathbf{Cppo} \\
[\![P \to N]\!] &= (\{f : [\![P]\!] \to [\![N]\!] \mid f \text{ is continuous}\}, \sqsubseteq_{[\![P]\!] \Rightarrow [\![N]\!]}, d \mapsto \bot_{[\![N]\!]}) \\
&\quad \text{where } f \sqsubseteq_{D \Rightarrow E} g \text{ iff } f(d) \sqsubseteq_E g(d) \text{ for all } d \in D \\
[\![\uparrow P]\!] &= ([\![P]\!] \uplus \{\bot_\uparrow\}, \left\{ ((1,d),(1,d')) \;\middle|\; d \sqsubseteq_{[\![P]\!]} d' \right\} \cup \left\{ ((2,\bot_\uparrow), d) \;\middle|\; d \in [\![\uparrow P]\!] \right\}, (2, \bot_\uparrow))
\end{aligned}
$$

$$
\begin{aligned}
[\![\mathscr{F}]\!] &: \mathbf{Cpo} \to \mathbf{Cpo} \\
[\![F \oplus G]\!] &= X \mapsto [\![F]\!]X \uplus [\![G]\!]X \\
[\![I]\!] &= X \mapsto \{\bullet\} \\
[\![\hat{B} \otimes \hat{P}]\!] &= X \mapsto [\![\hat{B}]\!]X \times [\![\hat{P}]\!]X \\
[\![\underline{P}]\!] &= X \mapsto [\![P]\!] \\
[\![\mathrm{Id}]\!] &= X \mapsto X
\end{aligned}
$$

$$
\begin{aligned}
\mathit{fmap}\ [\![F_1 \oplus F_2]\!]\ f &= d \mapsto \begin{cases} (1, (\mathit{fmap}\ [\![F_1]\!]\ f)\ d') & \text{if } d = (1, d') \\ (2, (\mathit{fmap}\ [\![F_2]\!]\ f)\ d') & \text{if } d = (2, d') \end{cases} \\
\mathit{fmap}\ [\![I]\!]\ f &= \mathit{id}_{\{\bullet\}} \\
\mathit{fmap}\ [\![\hat{B} \otimes \hat{P}]\!]\ f &= (d_1, d_2) \mapsto \left((\mathit{fmap}\ [\![\hat{B}]\!]\ f)\ d_1, (\mathit{fmap}\ [\![\hat{P}]\!]\ f)\ d_2 \right) \\
\mathit{fmap}\ [\![\underline{P}]\!]\ f &= \mathit{id}_{[\![P]\!]} \\
\mathit{fmap}\ [\![\mathrm{Id}]\!]\ f &= f
\end{aligned}
$$

Figure A.39: Denotational Semantics of Unrefined Types and Functors

$$[\![\Gamma \vdash h \Rightarrow P]\!] : [\![\Gamma]\!] \to [\![P]\!]$$
$$[\![x]\!]_\delta = \delta(x)$$
$$[\![(v : P)]\!]_\delta = [\![v]\!]_\delta$$

$$[\![\Gamma \vdash g \Rightarrow \uparrow P]\!] : [\![\Gamma]\!] \to [\![\uparrow P]\!]$$
$$[\![h(s)]\!]_\delta = [\![s]\!]_\delta \ [\![h]\!]_\delta$$
$$[\![(e : \uparrow P)]\!]_\delta = [\![e]\!]_\delta$$

Figure A.40: Denotational semantics of unrefined heads h and bound expressions g

$$[\![\Gamma \vdash v \Leftarrow P]\!] : [\![\Gamma]\!] \to [\![P]\!]$$
$$[\![x]\!]_\delta = \delta(x)$$
$$[\![\langle\rangle]\!]_\delta = \bullet$$
$$[\![\langle v_1, v_2\rangle]\!]_\delta = ([\![v_1]\!]_\delta, [\![v_2]\!]_\delta)$$
$$[\![\mathtt{inj}_k\, v]\!]_\delta = (k, [\![v]\!]_\delta)$$
$$[\![\mathtt{into}(v)]\!]_\delta = [\![v]\!]_\delta$$
$$[\![\{e\}]\!]_\delta = [\![e]\!]_\delta$$

$$[\![\Gamma \vdash e \Leftarrow N]\!] : [\![\Gamma]\!] \to [\![N]\!]$$
$$[\![\mathtt{return}\, v]\!]_\delta = (1, [\![v]\!]_\delta)$$
$$[\![\Gamma \vdash \mathtt{let}\ x = g;\ e \Leftarrow N]\!]_\delta = \begin{cases} [\![e]\!]_{(\delta, V/x)} & \text{if } [\![g]\!]_\delta = (1, V) \\ \bot_{[\![N]\!]} & \text{if } [\![g]\!]_\delta = (2, \bot_\uparrow) \end{cases}$$
$$[\![\lambda x.e]\!]_\delta = V \mapsto [\![e]\!]_{(\delta, V/x)}$$
$$[\![\Gamma \vdash \mathtt{rec}\, x.\ e \Leftarrow N]\!]_\delta = \bigsqcup_{k \in \mathbb{N}} \left(V \mapsto [\![\Gamma, x : \downarrow N \vdash e \Leftarrow N]\!]_{\delta, V/x}\right)^k \bot_{[\![N]\!]}$$
$$[\![\mathtt{match}\ h\ \{r_i \Rightarrow e_i\}_{i \in I}]\!]_\delta = [\![\{r_i \Rightarrow e_i\}_{i \in I}]\!]_\delta\, [\![h]\!]_\delta$$

Figure A.41: Denotational semantics of unrefined values v and expressions e

$$[\![\Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N]\!] : [\![\Gamma]\!] \to [\![P]\!] \to [\![N]\!]$$
$$[\![\{\langle\rangle \Rightarrow e\}]\!]_\delta = V \mapsto [\![e]\!]_\delta$$
$$[\![\{\langle x_1, x_2\rangle \Rightarrow e\}]\!]_\delta = (V_1, V_2) \mapsto [\![e]\!]_{(\delta, V_1/x_1, V_2/x_2)}$$
$$[\![\{\mathtt{inj}_1\, x_1 \Rightarrow e_1 \mid \mathtt{inj}_2\, x_2 \Rightarrow e_2\}]\!]_\delta = V \mapsto \begin{cases} [\![e_1]\!]_{\delta, V_1/x_1} & \text{if } V = (1, V_1) \\ [\![e_2]\!]_{\delta, V_2/x_2} & \text{if } V = (2, V_2) \end{cases}$$
$$[\![\{\}]\!]_\delta = \text{empty function}$$
$$[\![\{\mathtt{into}(x) \Rightarrow e\}]\!]_\delta = V \mapsto [\![e]\!]_{\delta, V/x}$$

$$[\![\Gamma; [N] \vdash s \Rightarrow M]\!] : [\![\Gamma]\!] \to [\![N]\!] \to [\![M]\!]$$
$$[\![v, s]\!]_\delta = f \mapsto [\![s]\!]_\delta\, (f([\![v]\!]_\delta))$$
$$[\![\cdot]\!]_\delta = V \mapsto V$$

Figure A.42: Denotational semantics of unrefined match expressions and spines

Appendix A.5 Index erasure

$$
\begin{aligned}
|1| &= 1 \\
|R \times R'| &= |R| \times |R'| \\
|0| &= 0 \\
|P + P'| &= |P| + |P'| \\
|{\downarrow}N| &= {\downarrow}|N| \\
|\{\nu : \mu F \mid \mathscr{M}(F)\}| &= \mu|F| \\
|\exists a : \kappa. P| &= |P| \\
|Q \wedge \varphi| &= |Q| \\
|R \to L| &= |R| \to |L| \\
|{\uparrow}P| &= {\uparrow}|P| \\
|\forall a : \kappa. N| &= |N| \\
|\varphi \supset M| &= |M|
\end{aligned}
\qquad
\begin{aligned}
|F \oplus G| &= |F| \oplus |G| \\
|I| &= I \\
|\hat{B} \otimes \hat{P}| &= |\hat{B}| \otimes |\hat{P}| \\
|\underline{P}| &= \underline{|P|} \\
|\mathrm{Id}| &= \mathrm{Id}
\end{aligned}
$$

Figure A.43: Index erasure of refined types and functors to unrefined types and functors

$$
\begin{aligned}
|x| &= x \\
|(v : P)| &= (|v| : |P|) \\
|h(s)| &= |h|(|s|) \\
|(e : N)| &= (|e| : |N|) \\
|\cdot| &= \cdot \\
|v, s| &= |v|, |s| \\
|\langle\rangle| &= \langle\rangle \\
|\langle v_1, v_2\rangle| &= \langle |v_1|, |v_2| \rangle \\
|\mathsf{inj}_1\, v| &= \mathsf{inj}_1\, |v| \\
|\mathsf{inj}_2\, v| &= \mathsf{inj}_2\, |v| \\
|\mathsf{into}(v)| &= \mathsf{into}(|v|) \\
|\{e\}| &= \{|e|\} \\
|\mathsf{return}\, v| &= \mathsf{return}\, |v| \\
|\mathsf{let}\, x = g;\, e| &= \mathsf{let}\, x = |g|;\, |e| \\
|\mathsf{match}\, h\, \{r_i \Rightarrow e_i\}_{i \in I}| &= \mathsf{match}\, |h|\, |\{r_i \Rightarrow e_i\}_{i \in I}| \\
|\lambda x.e| &= \lambda x.|e| \\
|\mathsf{rec}\, x : (\forall a : \mathbb{N}.N).\, e| &= \mathsf{rec}\, x.\, |e| \\
|\Theta; \Gamma \vdash \mathsf{unreachable} \Leftarrow L| &= \mathsf{diverge}_{|L|} \\
|\{r_i \Rightarrow e_i\}_{i \in I}| &= \{r_i \Rightarrow |e_i|\}_{i \in I}
\end{aligned}
$$

Figure A.44: Index erasure of refined program terms

$$
\begin{aligned}
|\cdot| &= \cdot & \qquad |\cdot| &= \cdot \\
|\sigma, t/a| &= |\sigma| & \qquad |\delta, d/a| &= |\delta| \\
|\sigma, v : R/x| &= |\sigma|, |v| : |R|/x & \qquad |\delta, V/x| &= |\delta|, V/x
\end{aligned}
$$

Figure A.45: Index erasure of substitution

Appendix A.6 Denotational Semantics (Refined System)

$$\text{Semantic substitutions} \quad \delta \ ::= \ \cdot \mid \delta, d/a \mid \delta, V/x$$

$\boxed{\vdash \delta : \Theta; \Gamma}$ We know that δ is a semantic substitution for variables in Θ and Γ (all inputs)

$$\frac{}{\vdash \cdot : \cdot; \cdot}\ \text{Empty}\delta \qquad \frac{\vdash \delta : \Theta; \Gamma \qquad d \in [\![\tau]\!] \qquad a \notin dom(\Theta)}{\vdash \delta, d/a : \Theta, a : \tau[, a\,\mathrm{Id}]; \Gamma}\ \text{Ix}\delta\text{Id}$$

$$\frac{\vdash \delta : \Theta; \Gamma \qquad [\![\varphi]\!]_{\lfloor \delta \rfloor} = \{\bullet\}}{\vdash \delta : \Theta, \varphi; \Gamma}\ \text{Prop}\delta$$

$$\frac{\vdash \delta : \Theta; \Gamma \qquad V \in [\![R]\!]_{\lfloor \delta \rfloor} \qquad x \notin dom(\Gamma)}{\vdash \delta, V/x : \Theta; \Gamma, x : R}\ \text{Val}\delta$$

$$[\![\Theta; \Gamma]\!] = \{\delta \mid \vdash \delta : \Theta; \Gamma\}$$
$$[\![\Theta]\!] = [\![\Theta; \cdot]\!]$$
$$[\![\Gamma]\!] = [\![\cdot; \Gamma]\!]$$

$\boxed{\lfloor \delta \rfloor}$ Filter out program variable entries

$$\lfloor \cdot \rfloor = \cdot$$
$$\lfloor \delta, d/a \rfloor = \lfloor \delta \rfloor, d/a$$
$$\lfloor \delta, V/x \rfloor = \lfloor \delta \rfloor$$

Figure A.46: Semantic substitutions and their operations and judgments

$$
\begin{aligned}
[\![\Xi; [\tau] \vdash \boldsymbol{t} : \kappa]\!] &: [\![\Xi]\!] \to [\![\tau]\!] \to [\![\kappa]\!] \\
[\![\cdot]\!]_\delta &= d \mapsto d \\
[\![t, \boldsymbol{u}]\!]_\delta &= f \mapsto [\![\boldsymbol{u}]\!]_\delta\,(f([\![t]\!]_\delta)) \\
[\![.k, \boldsymbol{u}]\!]_\delta &= (d_1, d_2) \mapsto [\![\boldsymbol{u}]\!]_\delta\,(d_k)
\end{aligned}
$$

$$
\begin{aligned}
[\![\Xi \vdash t : \tau\ [_]]\!] &: [\![\Xi]\!] \to [\![\tau]\!] \\
[\![a]\!]_\delta &= \delta(a) \\
[\![n]\!]_\delta &= n \\
[\![t_1 + t_2]\!]_\delta &= [\![t_1]\!]_\delta + [\![t_2]\!]_\delta \\
[\![t_1 - t_2]\!]_\delta &= [\![t_1]\!]_\delta - [\![t_2]\!]_\delta \\
[\![(t_1, t_2)]\!]_\delta &= ([\![t_1]\!]_\delta, [\![t_2]\!]_\delta) \\
[\![\lambda a.t]\!]_\delta &= d \mapsto [\![t]\!]_{\delta, d/a} \\
[\![a(\boldsymbol{t})]\!]_\delta &= [\![\boldsymbol{t}]\!]_\delta\,\delta(a) \\
[\![t = t']\!]_\delta &= \begin{cases} \{\bullet\} & \text{if } [\![t]\!]_\delta = [\![t']\!]_\delta \\ \emptyset & \text{else} \end{cases} \\
[\![\varphi_1 \wedge \varphi_2]\!]_\delta &= [\![\varphi_1]\!]_\delta \cap [\![\varphi_2]\!]_\delta \\
[\![\varphi_1 \vee \varphi_2]\!]_\delta &= [\![\varphi_1]\!]_\delta \cup [\![\varphi_2]\!]_\delta \\
[\![\neg\varphi]\!]_\delta &= \{\bullet\} \setminus [\![\varphi]\!]_\delta \\
[\![\mathsf{tt}]\!]_\delta &= \{\bullet\} \\
[\![\mathsf{ff}]\!]_\delta &= \emptyset
\end{aligned}
$$

Figure A.47: Denotational semantics of (well-formed) indexes t and index spines $\boldsymbol{t}$

$\boxed{\delta\restriction_{\Theta}}$ Semantic substitution restriction

$$\begin{aligned}
\cdot\restriction_{\Theta} &= \cdot \\
(\delta, d/a)\restriction_{\Theta} &= \begin{cases} \delta\restriction_{\Theta}, d/a & \text{if } a \in dom(\Theta) \\ \delta\restriction_{\Theta} & \text{else} \end{cases} \\
(\delta, V/x)\restriction_{\Theta} &= \delta\restriction_{\Theta}
\end{aligned}$$

$\boxed{\delta_1\restriction_{\xi} = \delta_2\restriction_{\xi}}$ δ_1 and δ_2 agree at ξ, that is, for all $\mathfrak{D}{\to}a \in \xi$, if $\delta_1\restriction_{\mathfrak{D}} = \delta_2\restriction_{\mathfrak{D}}$ then $\delta_1(a) = \delta_2(a)$

$$\frac{}{\delta_1\restriction_{\cdot} = \delta_2\restriction_{\cdot}} \qquad \frac{\delta_1\restriction_{\xi} = \delta_2\restriction_{\xi} \qquad \text{if } \delta_1\restriction_{\mathfrak{D}} = \delta_2\restriction_{\mathfrak{D}} \text{ then } \delta_1(a) = \delta_2(a)}{\delta_1\restriction_{\xi,\mathfrak{D}\to a} = \delta_2\restriction_{\xi,\mathfrak{D}\to a}}$$

$\boxed{\left[\!\left[\check{\Xi}\right]\!\right]^{\mathsf{fix}}_{\delta}}$ Semantics for liftapps output contexts

Define $\left[\!\left[\check{\Xi}\right]\!\right]^{\mathsf{fix}}_{\delta}$ to be the fixed point of applying $\left[\!\left[\check{\Xi}\right]\!\right]_{\delta,_}$ initially to $\left[\!\left[\check{\Xi}\right]\!\right]_{\delta}$ where

$$\begin{aligned}
[\![\cdot]\!]_{\delta} &= \cdot \\
\left[\!\left[\check{\Xi}, \check{a}^{a(\boldsymbol{u})}\right]\!\right]_{\delta} &= \left[\!\left[\check{\Xi}\right]\!\right]_{\delta}, \left([\![\boldsymbol{u}]\!]_{\delta}\, \delta(a)\right)/\check{a}^{a(\boldsymbol{u})}
\end{aligned}$$

Figure A.48: Operations and judgments for semantic substitutions

Given $\Xi \vdash F$ functor$[_]$ and $\vdash \delta : \Xi$, define

$$\mu\,[\![F]\!]_\delta = \bigcup_{k=0}^{\infty} [\![F]\!]_\delta^k\,\emptyset$$

Given $\Xi \vdash F$ functor$[_]$ and $\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$ and $\vdash \delta : \Xi$, define

$$fold_{[\![F]\!]_\delta}\,[\![\alpha]\!]_\delta = [\![\alpha]\!]_\delta \circ \Big([\![F]\!]_\delta\,(fold_{[\![F]\!]_\delta}\,[\![\alpha]\!]_\delta)\Big)$$

$$\begin{aligned}
\mathrm{Ob}(\mathbf{rCpo}) &= \{(D,R) \mid D \in \mathbf{Cpo} \text{ and } R \subseteq D\} \\
\mathrm{Hom}_{\mathbf{rCpo}}((D_1,R_1),(D_2,R_2)) &= \left\{ f \in \mathrm{Hom}_{\mathbf{Cpo}}(D_1,D_2) \;\middle|\; f(R_1) \subseteq R_2 \right\} \\
&\quad (\mathbf{rCppo} \text{ defined similarly})
\end{aligned}$$

$$\begin{aligned}
[\![\Xi \vdash P\ \mathsf{type}[_]]\!] &: [\![\Xi]\!] \to \mathbf{rCpo} \\
[\![1]\!]_\delta &= \{\bullet\} \\
[\![R_1 \times R_2]\!]_\delta &= [\![R_1]\!]_\delta \times [\![R_2]\!]_\delta \\
[\![0]\!]_\delta &= \emptyset \\
[\![P_1 + P_2]\!]_\delta &= [\![P_1]\!]_\delta \uplus [\![P_2]\!]_\delta \\
[\![\{v : \mu F \mid \mathscr{M}(F)\}]\!]_\delta &= \{V \in \mu\,[\![|F|]\!] \mid V \in \mu\,[\![F]\!]_\delta \text{ and } [\![\mathscr{M}(F)]\!]_\delta V = \{\bullet\}\} \\
&\quad \text{where } [\![\mathscr{M}(F)]\!]_\delta V \text{ is true } (\{\bullet\}) \text{ if} \\
&\quad\quad [\![t]\!]_\delta\,((fold_{[\![F]\!]_\delta}\,[\![\alpha]\!]_\delta)\,V) = [\![t]\!]_\delta \\
&\quad\quad\quad \text{for all } (\mathsf{fold}_F\ \alpha)\,v\,t =_\tau t \in \mathscr{M}(F) \\
&\quad \text{and is false } (\emptyset) \text{ otherwise} \\
\left[\!\left[\exists^{\mathsf{d}}\Xi.Q\right]\!\right]_\delta &= \left\{ V \in [\![|Q|]\!] \;\middle|\; \exists \delta' \in \left[\!\left[{}^{\mathsf{d}}\Xi\right]\!\right].\, V \in [\![Q]\!]_{\delta,\delta'} \right\} \\
[\![R \wedge \overrightarrow{\varphi}]\!]_\delta &= \left\{ V \in [\![|R|]\!] \;\middle|\; V \in [\![R]\!]_\delta \text{ and } [\![\varphi]\!]_\delta = \{\bullet\} \text{ for all } \varphi \in \overrightarrow{\varphi} \right\} \\
[\![\downarrow N]\!]_\delta &= [\![N]\!]_\delta
\end{aligned}$$

$$\begin{aligned}
[\![\Xi \vdash N\ \mathsf{type}[_]]\!] &: [\![\Xi]\!] \to \mathbf{rCppo} \\
\left[\!\left[\forall^{\mathsf{d}}\Xi.M\right]\!\right]_\delta &= \left\{ f \in [\![|M|]\!] \;\middle|\; \forall \delta' \in \left[\!\left[{}^{\mathsf{d}}\Xi\right]\!\right].\, f \in [\![M]\!]_{\delta,\delta'} \right\} \\
[\![R \to L]\!]_\delta &= \{f \in [\![|R \to L|]\!] \mid \forall V \in [\![R]\!]_\delta.\, f(V) \in [\![L]\!]_\delta\} \\
[\![\overrightarrow{\varphi} \supset L]\!]_\delta &= \left\{ f \in [\![|L|]\!] \;\middle|\; \text{if } [\![\varphi]\!]_\delta = \{\bullet\} \text{ for all } \varphi \in \overrightarrow{\varphi} \text{ then } f \in [\![L]\!]_\delta \right\} \\
[\![\uparrow P]\!]_\delta &= \{(1,V) \mid V \in [\![P]\!]_\delta\}
\end{aligned}$$

Figure A.49: Denotational semantics of (well-formed) refined types (specifying the second component, i.e. the refined set; the first component is denotation of erasure)

(This figure is part of a mutually recursive definition that includes Fig. A.49.)

$$
\begin{aligned}
[\![\Xi \vdash \mathscr{F}\ \mathsf{functor}[_]]\!] &: [\![\Xi]\!] \to \mathbf{rCpo} \to \mathbf{rCpo} \\
[\![\mathscr{F}]\!]_\delta\,(D,R) &= ([\![|\mathscr{F}|]\!]\,D, [\![\mathscr{F}]\!]_\delta\,R) \\
[\![F_1 \oplus F_2]\!]_\delta\,R &= [\![F_1]\!]_\delta\,R \uplus [\![F_2]\!]_\delta\,R \\
[\![I]\!]_\delta\,R &= \{\bullet\} \\
[\![\hat{B} \otimes \hat{P}]\!]_\delta\,R &= [\![\hat{B}]\!]_\delta\,R \times [\![\hat{P}]\!]_\delta\,R \\
[\![\underline{P}]\!]_\delta\,R &= [\![P]\!]_\delta \\
[\![\mathrm{Id}]\!]_\delta\,R &= R \\
\mathit{fmap}\,[\![\mathscr{F}]\!]_\delta\,f &= \mathit{fmap}\,[\![|\mathscr{F}|]\!]\,f
\end{aligned}
$$

$$[\![\Xi \vdash \alpha : F(\tau) \Rightarrow \tau]\!] : \prod_{\delta \in [\![\Xi]\!]} [\![F]\!]_\delta\,[\![\tau]\!] \to [\![\tau]\!]$$

$$[\![\Xi \vdash \alpha : (F_1 \oplus F_2)(\tau) \Rightarrow \tau]\!]_\delta\,V = \begin{cases} [\![\Xi \vdash \alpha_1 : F_1(\tau) \Rightarrow \tau]\!]_\delta\,V' & \text{if } V = (1,V') \\ [\![\Xi \vdash \alpha_2 : F_2(\tau) \Rightarrow \tau]\!]_\delta\,V' & \text{if } V = (2,V') \end{cases}$$
$$\text{where } \alpha \circ \mathsf{inj}_1 \mathrel{\mathring{=}} \alpha_1 \text{ and } \alpha \circ \mathsf{inj}_2 \mathrel{\mathring{=}} \alpha_2$$

$$[\![\Xi \vdash (\top, q) \Rightarrow t : (\underline{P} \otimes \hat{P})(\tau) \Rightarrow \tau]\!]_\delta\,(_,V) = [\![\Xi \vdash q \Rightarrow t : \hat{P}(\tau) \Rightarrow \tau]\!]_\delta\,V$$

$$\left[\!\!\left[\Xi \vdash (\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q) \Rightarrow t : (\underline{\exists\, {}^{\mathsf{d}}\Xi'.\, Q} \otimes \hat{P})(\tau) \Rightarrow \tau\right]\!\!\right]_\delta (V_1, V_2) =$$
$$\left[\!\!\left[\Xi, {}^{\mathsf{d}}\Xi' \vdash (\top, q) \Rightarrow t : (\underline{Q} \otimes \hat{P})(\tau) \Rightarrow \tau\right]\!\!\right]_{(\delta,\delta')} (V_1, V_2)$$
$$\text{where } \delta' \in \left[\!\!\left[{}^{\mathsf{d}}\Xi'\right]\!\!\right] \text{ satisfies } V_1 \in [\![Q]\!]_{\delta,\delta'}$$

$$[\![\Xi \vdash (a, q) \Rightarrow t : (\mathrm{Id} \otimes \hat{I})(\tau) \Rightarrow \tau]\!]_\delta\,(f,V) = [\![\Xi, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id} \vdash q \Rightarrow t : \hat{I}(\tau) \Rightarrow \tau]\!]_{(\delta, f/a)}\,V$$

$$[\![\Xi \vdash () \Rightarrow t : I(\tau) \Rightarrow \tau]\!]_\delta\,\bullet = [\![\overset{\mathsf{d}}{\div}\,\Xi \vdash t : \tau]\!]_{\delta \restriction \overset{\mathsf{d}}{\div}\,\Xi}$$

Figure A.50: Denotational semantics of (well-formed) refined functors and algebras

$$[\![\Theta;\Gamma \vdash h \Rightarrow P]\!] = (\delta \in [\![\Theta;\Gamma]\!]) \mapsto [\![|\Gamma| \vdash |h| \Rightarrow |P|]\!]_{|\delta|}$$
$$[\![\Theta;\Gamma \vdash g \Rightarrow \uparrow P]\!] = (\delta \in [\![\Theta;\Gamma]\!]) \mapsto [\![|\Gamma| \vdash |g| \Rightarrow |\uparrow P|]\!]_{|\delta|}$$
$$[\![\Theta;\Gamma \vdash v \Leftarrow P]\!] = (\delta \in [\![\Theta;\Gamma]\!]) \mapsto [\![|\Gamma| \vdash |v| \Leftarrow |P|]\!]_{|\delta|}$$
$$[\![\Theta;\Gamma \vdash e \Leftarrow N]\!] = (\delta \in [\![\Theta;\Gamma]\!]) \mapsto [\![|\Gamma| \vdash |e| \Leftarrow |N|]\!]_{|\delta|}$$
$$[\![\Theta;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N]\!] = (\delta \in [\![\Theta;\Gamma]\!]) \mapsto [\![|\Gamma|;[|P|] \vdash |\{r_i \Rightarrow e_i\}_{i\in I}| \Leftarrow |N|]\!]_{|\delta|}$$
$$[\![\Theta;\Gamma;[N] \vdash s \Rightarrow M]\!] = (\delta \in [\![\Theta;\Gamma]\!]) \mapsto [\![|\Gamma|;[|N|] \vdash |s| \Rightarrow |M|]\!]_{|\delta|}$$

Figure A.51: Denotational semantics of refined program terms

Appendix A.7 Algorithmic System

$\boxed{\hat{\Theta}\ \mathsf{algctx}}$ Algorithmic context $\hat{\Theta}$ (input) is well-formed

$$\frac{\|\hat{\Theta}\|\ \mathsf{ctx} \qquad \overset{\mathsf{d}\cdot}{\div}\|\hat{\Theta}\| \vdash t : \kappa \text{ for all } \blacktriangleright\hat{a} : \kappa{=}t \in \hat{\Theta} \qquad \overset{\mathsf{d}\cdot}{\div}\|\hat{\Theta}\|, \blacktriangleright\hat{\Theta} \vdash t : \kappa \text{ for all } \hat{a} : \kappa{=}t \in \hat{\Theta}}{\hat{\Theta}\ \mathsf{algctx}}$$

$$\|\Theta\| = \Theta$$
$$\|\hat{\Theta}, [\blacktriangleright]\hat{a} \overset{\mathsf{d}\cdot}{\div} \kappa[{=}t]\| = \|\hat{\Theta}\|$$

$$[\xi]\Theta = \Theta$$
$$[\xi](\hat{\Theta}, \hat{a} \overset{\mathsf{d}\cdot}{\div} \kappa) = \begin{cases} [\xi]\hat{\Theta}, \blacktriangleright\hat{a} \overset{\mathsf{d}\cdot}{\div} \kappa & \text{if } \hat{a} \in \mathfrak{D} \text{ for some } \blacktriangleright\mathfrak{D} \in \xi \\ [\xi]\hat{\Theta}, \hat{a} \overset{\mathsf{d}\cdot}{\div} \kappa & \text{else} \end{cases}$$
$$[\xi](\hat{\Theta}, \hat{a} : \kappa{=}t) = \begin{cases} [\xi]\hat{\Theta}, \blacktriangleright\hat{a} : \kappa{=}t & \text{if } \hat{a} \in \mathfrak{D} \text{ for some } \blacktriangleright\mathfrak{D} \in \xi \\ [\xi]\hat{\Theta}, \hat{a} : \kappa{=}t & \text{else} \end{cases}$$

$$\blacktriangleright\Theta = \cdot$$
$$\blacktriangleright(\hat{\Theta}, \blacktriangleright\hat{a} \overset{\mathsf{d}\cdot}{\div} \kappa) = \blacktriangleright\hat{\Theta}, \blacktriangleright\hat{a} \overset{\mathsf{d}\cdot}{\div} \kappa$$
$$\blacktriangleright(\hat{\Theta}, \blacktriangleright\hat{a} : \kappa{=}t) = \blacktriangleright\hat{\Theta}, \blacktriangleright\hat{a} : \kappa{=}t$$
$$\blacktriangleright(\hat{\Theta}, \hat{a} \overset{\mathsf{d}\cdot}{\div} \kappa[{=}t]) = \blacktriangleright\hat{\Theta}$$

Figure A.52: Algorithmic context well-formedness and related definitions, operations, judgments

$$
\begin{aligned}
sol(\Theta) &= \cdot \\
sol(\hat{\Theta}, [\blacktriangleright]\hat{a} \stackrel{\mathsf{d}\cdot}{\div} \kappa) &= sol(\hat{\Theta}) \\
sol(\hat{\Theta}, \hat{a} : \kappa{=}t) &= sol(\hat{\Theta}), \hat{a} : \kappa{=}t \\
sol(\hat{\Theta}, \blacktriangleright\hat{a} : \kappa{=}t) &= sol(\hat{\Theta}), \blacktriangleright\hat{a} : \kappa{=}t
\end{aligned}
$$

$$
\begin{aligned}
unsol(\Theta) &= \cdot \\
unsol(\hat{\Theta}, \hat{a} \stackrel{\mathsf{d}\cdot}{\div} \kappa) &= unsol(\hat{\Theta}), \hat{a} \stackrel{\mathsf{d}\cdot}{\div} \kappa \\
unsol(\hat{\Theta}, \blacktriangleright\hat{a} \stackrel{\mathsf{d}\cdot}{\div} \kappa) &= unsol(\hat{\Theta}), \blacktriangleright\hat{a} \stackrel{\mathsf{d}\cdot}{\div} \kappa \\
unsol(\hat{\Theta}, [\blacktriangleright]\hat{a} : \kappa{=}t) &= unsol(\hat{\Theta})
\end{aligned}
$$

$\hat{\Theta}$ present iff $\blacktriangleright$ does not occur in $\hat{\Theta}$

$$
\begin{aligned}
[\hat{\Theta}]\Theta &= \Theta \\
[\hat{\Theta}](\hat{\Theta}', \hat{a} \stackrel{\mathsf{d}\cdot}{\div} \kappa) &= [\hat{\Theta}]\hat{\Theta}', \hat{a} \stackrel{\mathsf{d}\cdot}{\div} \kappa \\
[\hat{\Theta}](\hat{\Theta}', \blacktriangleright\hat{a} \stackrel{\mathsf{d}\cdot}{\div} \kappa) &= [\hat{\Theta}]\hat{\Theta}', \blacktriangleright\hat{a} \stackrel{\mathsf{d}\cdot}{\div} \kappa \\
[\hat{\Theta}](\hat{\Theta}', \hat{a} : \kappa{=}t) &= [\hat{\Theta}]\hat{\Theta}', \hat{a} : \kappa{=}[\hat{\Theta}]t \\
[\hat{\Theta}](\hat{\Theta}', \blacktriangleright\hat{a} : \kappa{=}t) &= [\hat{\Theta}]\hat{\Theta}', \blacktriangleright\hat{a} : \kappa{=}t
\end{aligned}
$$

Figure A.53: Operations and judgments for algorithmic contexts

$\boxed{\hat{\Xi} \rhd t : \tau\ [\xi_t]}$ Under $\hat{\Xi}$ (input), index t (input) has sort τ (input) and value-determined dependencies ξ_t (output) Note that $\hat{\Xi} \rhd t : \tau$ abbreviates $\hat{\Xi} \rhd t : \tau\ [_]$

$$\frac{(a : \tau) \in \hat{\Xi}}{\hat{\Xi} \rhd a : \tau\ [\cdot]}\ \text{AlgIxVar} \qquad \frac{([\blacktriangleright]\hat{a} \stackrel{\mathsf{d}}{\div} \kappa[=t]) \in \hat{\Xi}}{\hat{\Xi} \rhd \hat{a} : \kappa\ [\cdot]}\ \text{AlgIxEVar[Solved][}\blacktriangleright\text{]}$$

$$\frac{\hat{\Xi} \rhd t : \tau\ [\xi'] \qquad \xi \subsetneq \xi'}{\hat{\Xi} \rhd t : \tau\ [\xi]}\ \text{AlgIxSub} \qquad \frac{t \in \mathscr{K}_\kappa}{\hat{\Xi} \rhd t : \kappa\ [\cdot]}\ \text{AlgIxConst}$$

$$\frac{\kappa \in \{\mathbb{N}, \mathbb{Z}\} \qquad \hat{\Xi} \rhd t_1 : \kappa\ [\xi_1] \qquad \hat{\Xi} \rhd t_2 : \kappa\ [\xi_2]}{\hat{\Xi} \rhd t_1 + t_2 : \kappa\ [\cdot]}\ \text{AlgIxPlus}$$

$$\frac{\kappa \in \{\mathbb{N}, \mathbb{Z}\} \qquad \hat{\Xi} \rhd t_1 : \kappa\ [\xi_1] \qquad \hat{\Xi} \rhd t_2 : \kappa\ [\xi_2]}{\hat{\Xi} \rhd t_1 - t_2 : \kappa\ [\cdot]}\ \text{AlgIxMinus}$$

$$\frac{\hat{\Xi} \rhd t_1 : \tau_1\ [\xi_1] \qquad \hat{\Xi} \rhd t_2 : \tau_2\ [\xi_2]}{\hat{\Xi} \rhd (t_1, t_2) : \tau_1 \times \tau_2\ [\cdot]}\ \text{AlgIxProd}$$

$$\frac{a \div \kappa, \hat{\Xi} \rhd t : \tau\ [\xi]}{\hat{\Xi} \rhd \lambda a.t : \kappa \Rightarrow \tau\ [\cdot]}\ \text{AlgIx}\lambda \qquad \frac{(a : \tau) \in \hat{\Xi} \qquad \hat{\Xi}; [\tau] \rhd \boldsymbol{t} : \kappa}{\hat{\Xi} \rhd a(\boldsymbol{t}) : \kappa\ [\cdot]}\ \text{AlgIxApp}$$

$$\frac{([\blacktriangleright]\lfloor\hat{a}\rfloor \stackrel{\mathsf{d}}{\div} \kappa[=u]) \in \hat{\Xi} \qquad (t \stackrel{\mathsf{d}}{\div} \kappa[=u']) \notin \hat{\Xi} \qquad \stackrel{\mathsf{d}}{\div}\hat{\Xi} \rhd t : \kappa\ [\xi_t]}{\hat{\Xi} \rhd \lfloor\hat{a}\rfloor = t : \mathbb{B}\ [FV(t) \to \lfloor\hat{a}\rfloor]}\ \text{AlgIx=L}$$

$$\frac{([\blacktriangleright]\lfloor\hat{a}\rfloor \stackrel{\mathsf{d}}{\div} \kappa[=u]) \in \hat{\Xi} \qquad (t \stackrel{\mathsf{d}}{\div} \kappa[=u']) \notin \hat{\Xi} \qquad \stackrel{\mathsf{d}}{\div}\hat{\Xi} \rhd t : \kappa\ [\xi_t]}{\hat{\Xi} \rhd t = \lfloor\hat{a}\rfloor : \mathbb{B}\ [FV(t) \to \lfloor\hat{a}\rfloor]}\ \text{AlgIx=R}$$

$$\frac{([\blacktriangleright]\lfloor\hat{a}\rfloor_1 \stackrel{\mathsf{d}}{\div} \kappa[=u_1]) \in \hat{\Xi} \qquad ([\blacktriangleright]\lfloor\hat{a}\rfloor_2 \stackrel{\mathsf{d}}{\div} \kappa[=u_2]) \in \hat{\Xi}}{\hat{\Xi} \rhd \lfloor\hat{a}\rfloor_1 = \lfloor\hat{a}\rfloor_2 : \mathbb{B}\ [\lfloor\hat{a}\rfloor_1 \to \lfloor\hat{a}\rfloor_2, \lfloor\hat{a}\rfloor_2 \to \lfloor\hat{a}\rfloor_1]}\ \text{AlgIx=LR}$$

$$\frac{\hat{\Xi} \rhd u_1 = t_1 : \mathbb{B}\ [\xi_1] \qquad \hat{\Xi} \rhd u_2 = t_2 : \mathbb{B}\ [\xi_2]}{\hat{\Xi} \rhd (u_1, u_2) = (t_1, t_2) : \mathbb{B}\ [\xi_1 \cup \xi_2]}\ \text{AlgIx=}\times$$

$$\frac{\text{no other rule applies} \qquad \hat{\Xi} \rhd t_1 : \kappa\ [\xi_1] \qquad \hat{\Xi} \rhd t_2 : \kappa\ [\xi_2]}{\hat{\Xi} \rhd t_1 = t_2 : \mathbb{B}\ [\cdot]}\ \text{AlgIx=}$$

$$\frac{\hat{\Xi} \rhd \varphi_1 : \mathbb{B}\ [\xi_1] \qquad \hat{\Xi} \rhd \varphi_2 : \mathbb{B}\ [\xi_2]}{\hat{\Xi} \rhd \varphi_1 \wedge \varphi_2 : \mathbb{B}\ [\xi_1 \cup \xi_2]}\ \text{AlgIx}\wedge$$

$$\frac{\hat{\Xi} \rhd \varphi_1 : \mathbb{B}\ [\xi_1] \qquad \hat{\Xi} \rhd \varphi_2 : \mathbb{B}\ [\xi_2]}{\hat{\Xi} \rhd \varphi_1 \vee \varphi_2 : \mathbb{B}\ [\cdot]}\ \text{AlgIx}\vee \qquad \frac{\hat{\Xi} \rhd \varphi : \mathbb{B}\ [\xi]}{\hat{\Xi} \rhd \neg\varphi : \mathbb{B}\ [\cdot]}\ \text{AlgIx}\neg$$

$$\frac{\kappa \in \{\mathbb{N}, \mathbb{Z}\} \qquad \hat{\Xi} \rhd t_1 : \kappa\ [\xi_1] \qquad \hat{\Xi} \rhd t_2 : \kappa\ [\xi_2]}{\hat{\Xi} \rhd t_1 \leq t_2 : \mathbb{B}\ [\cdot]}\ \text{AlgIx}\leq$$

Figure A.54: Algorithmic index sorting

$\boxed{\hat{\Xi};[\tau] \rhd \boldsymbol{t} : \kappa}$ Under $\hat{\Xi}$ (input), fully applying an index of sort τ (input) to $\boldsymbol{t}$ (input) yields an index of sort κ (output)

$$\frac{}{\hat{\Xi};[\kappa] \rhd \cdot : \kappa}\ \text{AlgIxSpineNil} \qquad \frac{\hat{\Xi} \rhd t_0 : \kappa_0 \qquad \hat{\Xi};[\tau] \rhd \boldsymbol{t} : \kappa}{\hat{\Xi};[\kappa_0 \Rightarrow \tau] \rhd t_0, \boldsymbol{t} : \kappa}\ \text{AlgIxSpineEntry}$$

$$\frac{\hat{\Xi};[\tau_k] \rhd \boldsymbol{t} : \kappa}{\hat{\Xi};[\tau_1 \times \tau_2] \rhd .k, \boldsymbol{t} : \kappa}\ \text{AlgIxSpineProj}_k$$

Figure A.55: Algorithmic index spine sorting

$\boxed{\hat{\Xi} \rhd \mathscr{M}(F)\ \mathsf{msmts}[\xi_{\mathscr{M}(F)}]}$ Under $\hat{\Xi}$ (input), measurement list $\mathscr{M}(F)$ (input) is well-formed, with value-determined dependencies $\xi_{\mathscr{M}(F)}$ (output)

$$\frac{\cdot \rhd F\ \mathsf{functor}[_]}{\hat{\Xi} \rhd \cdot_F\ \mathsf{msmts}[\cdot]}$$

$$\frac{\hat{\Xi} \rhd \mathscr{M}(F)\ \mathsf{msmts}[\xi] \qquad \cdot \rhd \alpha : F(\tau) \Rightarrow \tau \qquad {}^{\mathsf{d}\div}\hat{\Xi}; [\tau] \rhd \boldsymbol{t} : \kappa \qquad ([\blacktriangleright]^{\lceil\hat{b}\rceil} {}^{\mathsf{d}\div}\kappa) \in \hat{\Xi}}{\hat{\Xi} \rhd \mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\, \nu\, \boldsymbol{t} =_\tau {}^{\lceil\hat{b}\rceil} \mathsf{msmts}[\xi \cup FV(\boldsymbol{t}) \to {}^{\lceil\hat{b}\rceil}]}$$

$$\frac{\begin{array}{c}\hat{\Xi} \rhd \mathscr{M}(F)\ \mathsf{msmts}[\xi] \\ \cdot \rhd \alpha : F(\tau) \Rightarrow \tau \qquad {}^{\mathsf{d}\div}\hat{\Xi}; [\tau] \rhd \boldsymbol{t} : \kappa \qquad ([\blacktriangleright] t\, {}^{\mathsf{d}\div}\kappa) \notin \hat{\Xi} \qquad {}^{\mathsf{d}\div}\hat{\Xi} \rhd t : \kappa\end{array}}{\hat{\Xi} \rhd \mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\, \nu\, \boldsymbol{t} =_\tau t\ \mathsf{msmts}[\xi]}$$

$\boxed{\hat{\Xi} \rhd A\ \mathsf{type}[\xi_A]}$ Under $\hat{\Xi}$ (input), type A (input) is well-formed, with value-determined dependencies ξ_A (output)

$$\frac{{}^{\mathsf{d}}\Xi, \hat{\Xi} \rhd Q\ \mathsf{type}[\xi_Q] \qquad \xi_Q - {}^{\mathsf{d}\div}\hat{\Xi} \vdash {}^{\mathsf{d}}\Xi\ \mathsf{det}}{\hat{\Xi} \rhd \exists {}^{\mathsf{d}}\Xi.\, Q\ \mathsf{type}[\xi_Q - {}^{\mathsf{d}}\Xi]}\ \mathrm{AlgTp}\exists$$

$$\frac{\hat{\Xi} \rhd R\ \mathsf{type}[\xi_R] \qquad \hat{\Xi} \rhd \overrightarrow{\varphi} : \mathbb{B}\ [\xi_{\overrightarrow{\varphi}}]}{\hat{\Xi} \rhd R \wedge \overrightarrow{\varphi}\ \mathsf{type}[\xi_R \cup \xi_{\overrightarrow{\varphi}}]}\ \mathrm{AlgTp}\wedge \qquad \frac{\hat{\Xi} \rhd \mathscr{M}(F)\ \mathsf{msmts}[\xi]}{\hat{\Xi} \rhd \{\nu : \mu F \mid \mathscr{M}(F)\}\ \mathsf{type}[\xi]}\ \mathrm{AlgTp}\mu$$

$$\frac{\hat{\Xi} \rhd P_1\ \mathsf{type}[\xi_1] \qquad \hat{\Xi} \rhd P_2\ \mathsf{type}[\xi_2]}{\hat{\Xi} \rhd P_1 + P_2\ \mathsf{type}[\cdot]}\ \mathrm{AlgTp}+$$

$$\frac{\hat{\Xi} \rhd R_1\ \mathsf{type}[\xi_1] \qquad \hat{\Xi} \rhd R_2\ \mathsf{type}[\xi_2]}{\hat{\Xi} \rhd R_1 \times R_2\ \mathsf{type}[\xi_1 \cup \xi_2]}\ \mathrm{AlgTp}\times \qquad \frac{}{\hat{\Xi} \rhd 0\ \mathsf{type}[\cdot]}\ \mathrm{AlgTp0}$$

$$\frac{}{\hat{\Xi} \rhd 1\ \mathsf{type}[\cdot]}\ \mathrm{AlgTp1} \qquad \frac{\hat{\Xi} \rhd N\ \mathsf{type}[\xi_N]}{\hat{\Xi} \rhd \downarrow N\ \mathsf{type}[\cdot]}\ \mathrm{AlgTp}\downarrow \qquad \frac{\hat{\Xi} \rhd P\ \mathsf{type}[\xi_P]}{\hat{\Xi} \rhd \uparrow P\ \mathsf{type}[\cdot]}\ \mathrm{AlgTp}\uparrow$$

$$\frac{\hat{\Xi} \rhd R\ \mathsf{type}[\xi_R] \qquad \hat{\Xi} \rhd L\ \mathsf{type}[\xi_L]}{\hat{\Xi} \rhd R \to L\ \mathsf{type}[\xi_R \cup \xi_L]}\ \mathrm{AlgTp}\to$$

$$\frac{\hat{\Xi} \rhd L\ \mathsf{type}[\xi_L] \qquad \hat{\Xi} \rhd \overrightarrow{\varphi} : \mathbb{B}\ [\xi_{\overrightarrow{\varphi}}]}{\hat{\Xi} \rhd \overrightarrow{\varphi} \supset L\ \mathsf{type}[\xi_L \cup \xi_{\overrightarrow{\varphi}}]}\ \mathrm{AlgTp}\supset$$

$$\frac{{}^{\mathsf{d}}\Xi, \hat{\Xi} \rhd M\ \mathsf{type}[\xi_M] \qquad \xi_M - {}^{\mathsf{d}\div}\hat{\Xi} \vdash {}^{\mathsf{d}}\Xi\ \mathsf{det}}{\hat{\Xi} \rhd \forall {}^{\mathsf{d}}\Xi.\, M\ \mathsf{type}[\xi_M - {}^{\mathsf{d}}\Xi]}\ \mathrm{AlgTp}\forall$$

Figure A.56: Algorithmic type well-formedness

$\boxed{\hat{\Xi} \rhd \mathscr{F}\ \mathsf{functor}[\xi]}$ Under $\hat{\Xi}$ (input), functor $\mathscr{F}$ (input) is well-formed, with value-determined dependencies ξ (output)

$$\frac{\hat{\Xi} \rhd P\ \mathsf{type}[\xi]}{\hat{\Xi} \rhd \underline{P}\ \mathsf{functor}[\xi]}\ \text{AlgFuncConst} \qquad \frac{}{\hat{\Xi} \rhd \mathrm{Id}\ \mathsf{functor}[\cdot]}\ \text{AlgFuncId}$$

$$\frac{}{\hat{\Xi} \rhd I\ \mathsf{functor}[\cdot]}\ \text{AlgFunc}I \qquad \frac{\hat{\Xi} \rhd \hat{B}\ \mathsf{functor}[\xi_1] \qquad \hat{\Xi} \rhd \hat{P}\ \mathsf{functor}[\xi_2]}{\hat{\Xi} \rhd \hat{B} \otimes \hat{P}\ \mathsf{functor}[\xi_1 \cup \xi_2]}\ \text{AlgFunc}\otimes$$

$$\frac{\hat{\Xi} \rhd F_1\ \mathsf{functor}[\xi_1] \qquad \hat{\Xi} \rhd F_2\ \mathsf{functor}[\xi_2]}{\hat{\Xi} \rhd F_1 \oplus F_2\ \mathsf{functor}[\cdot]}\ \text{AlgFunc}\oplus$$

$\boxed{\hat{\Xi} \rhd \alpha : F(\tau) \Rightarrow \tau}$ Under $\hat{\Xi}$, we know α is a well-formed algebra of kind $F(\tau) \Rightarrow \tau$ (inputs: $\hat{\Xi}, \alpha, F, \tau$)

$$\frac{\begin{array}{ll}\alpha \circ \mathsf{inj}_1 \mathrel{\hat{=}} \alpha_1 & \hat{\Xi} \rhd \alpha_1 : F_1(\tau) \Rightarrow \tau \\ \alpha \circ \mathsf{inj}_2 \mathrel{\hat{=}} \alpha_2 & \hat{\Xi} \rhd \alpha_2 : F_2(\tau) \Rightarrow \tau\end{array}}{\hat{\Xi} \rhd \alpha : (F_1 \oplus F_2)(\tau) \Rightarrow \tau}\ \text{AlgAlg}\oplus$$

$$\frac{\hat{\Xi} \rhd Q\ \mathsf{type}[\xi_Q] \qquad \hat{\Xi} \rhd q \Rightarrow t : \hat{P}(\tau) \Rightarrow \tau}{\hat{\Xi} \rhd (\top, q) \Rightarrow t : (\underline{Q} \otimes \hat{P})(\tau) \Rightarrow \tau}\ \text{AlgAlgConst}$$

$$\frac{{}^{\mathsf{d}}\Xi', \hat{\Xi} \rhd (\top, q) \Rightarrow t : (\underline{Q} \otimes \hat{P})(\tau) \Rightarrow \tau}{\hat{\Xi} \rhd (\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q) \Rightarrow t : (\underline{\exists^{\mathsf{d}}\Xi'.\, Q} \otimes \hat{P})(\tau) \Rightarrow \tau}\ \text{AlgAlg}\exists$$

$$\frac{a \mathrel{\dot{\div}}^{\mathsf{d}} \tau, a\,\mathrm{Id}, \hat{\Xi} \rhd q \Rightarrow t : \hat{I}(\tau) \Rightarrow \tau}{\hat{\Xi} \rhd (a, q) \Rightarrow t : (\mathrm{Id} \otimes \hat{I})(\tau) \Rightarrow \tau}\ \text{AlgAlgId} \qquad \frac{\mathrel{\dot{\div}}^{\mathsf{d}} \hat{\Xi} \rhd t : \tau}{\hat{\Xi} \rhd () \Rightarrow t : I(\tau) \Rightarrow \tau}\ \text{AlgAlg}I$$

Figure A.57: Algorithmic well-formedness of functors and algebras

$\boxed{\begin{array}{l}\hat{\Xi} \vdash {}^{(\forall)}W \ \mathsf{wf}[\xi] \\ \hat{\Xi} \vdash \chi \ \mathsf{Wf}[\xi]\end{array}}$ Under $\hat{\Xi}$ constraint(s) ${}^{(\forall)}W$ (χ) is (are) well-formed with det. dependencies ξ
Note $\hat{\Xi} \vdash {}^{(\forall)}W \ \mathsf{wf}$ e.g. abbreviates $\hat{\Xi} \vdash {}^{(\forall)}W \ \mathsf{wf}[_]$ (_ means "don't care")
(Inputs: $\hat{\Xi}, {}^{(\forall)}W, \chi$; output: ξ)

$$\frac{\hat{\Xi} \rhd \varphi : \mathbb{B} \ [\xi]}{\hat{\Xi} \vdash \varphi \ \mathsf{wf}[\xi]} \qquad \frac{\overset{\mathsf{d}}{\div}\hat{\Xi} \rhd u : \tau \qquad \overset{\mathsf{d}}{\div}\hat{\Xi} \rhd t : \tau}{\hat{\Xi} \vdash u \equiv_\tau t \ \mathsf{wf}[\cdot]} \qquad \frac{\overset{\mathsf{d}}{\div}\hat{\Xi}; [\tau] \rhd \boldsymbol{u} : \kappa \qquad \overset{\mathsf{d}}{\div}\hat{\Xi}; [\tau] \rhd \boldsymbol{t} : \kappa}{\hat{\Xi} \vdash \boldsymbol{u} \equiv_{[\tau]} \boldsymbol{t} \ \mathsf{wf}[\cdot]}$$

$$\frac{\hat{\Xi} \vdash {}^{(\forall)}W_1 \ \mathsf{wf}[\xi_1] \qquad \hat{\Xi} \vdash {}^{(\forall)}W_2 \ \mathsf{wf}[\xi_2]}{\hat{\Xi} \vdash {}^{(\forall)}W_1 \wedge {}^{(\forall)}W_2 \ \mathsf{wf}[\xi_1 \cup \xi_2]} \qquad \frac{\hat{\Xi} \rhd \varphi : \mathbb{B} \qquad \hat{\Xi} \vdash {}^{(\supset)}W \ \mathsf{wf}[\xi]}{\hat{\Xi} \vdash \varphi \supset {}^{(\supset)}W \ \mathsf{wf}[\cdot]}$$

$$\frac{a \overset{\mathsf{d}}{\div} \tau, \hat{\Xi} \vdash {}^{(\forall)}W \ \mathsf{wf}[\xi]}{\hat{\Xi} \vdash \forall a \overset{\mathsf{d}}{\div} \tau. \, {}^{(\forall)}W \ \mathsf{wf}[\cdot]} \qquad \frac{\hat{\Xi} \rhd A \ \mathsf{type}[_] \qquad \hat{\Xi} \rhd B \ \mathsf{type}[_]}{\hat{\Xi} \vdash \underline{A <:^{\pm} B} \ \mathsf{wf}[\cdot]}$$

$$\frac{\begin{array}{l}\text{there exist } \boldsymbol{t}, \tau, t, \kappa, \xi \text{ such that } t \rightsquigarrow \xi \text{ and } \hat{\Xi}; [\tau] \rhd \boldsymbol{t} : \kappa \\ \text{and for all } W_k \in \overrightarrow{W} \ (\neq \cdot) \text{ we have } \overset{\mathsf{d}}{\div}\hat{\Xi} \vdash W_k \ \mathsf{wf}[_] \\ \quad \text{and there exist } u_k, \boldsymbol{u}_k \text{ such that } W_k = (\boldsymbol{u}_k \equiv_{[\tau]} \boldsymbol{t} \wedge u_k \equiv_\kappa t) \text{ and } (u_k, \boldsymbol{u}_k) \, \mathsf{ground}\end{array}}{\hat{\Xi} \vdash \bigvee \overrightarrow{W} \ \mathsf{wf}[\xi]}$$

$$\frac{}{\hat{\Xi} \vdash \cdot \ \mathsf{Wf}[\cdot]} \qquad \frac{\hat{\Xi} \vdash W \ \mathsf{wf}[\xi_W] \qquad \hat{\Xi} \vdash \chi \ \mathsf{Wf}[\xi_\chi]}{\hat{\Xi} \vdash W, \chi \ \mathsf{Wf}[\xi_W \cup \xi_\chi]}$$

$$\frac{e \, \mathsf{ground} \qquad \hat{\Xi} \rhd N \ \mathsf{type}[_] \qquad \hat{\Xi} \vdash \chi \ \mathsf{Wf}[\xi]}{\hat{\Xi} \vdash (e \Leftarrow N), \chi \ \mathsf{Wf}[\xi]}$$

where

$$\frac{t \text{ is not an evar}}{t \rightsquigarrow \cdot} \qquad \frac{t \text{ is an evar}}{t \rightsquigarrow \blacktriangleright\{t\}}$$

We define an operation $\lfloor - \rfloor$ that removes program constraints:

$$\begin{aligned} \lfloor \cdot \rfloor &= \mathsf{tt} \\ \lfloor (e \Leftarrow N), \chi \rfloor &= \lfloor \chi \rfloor \\ \lfloor W, \chi \rfloor &= W \wedge \lfloor \chi \rfloor \end{aligned}$$

Figure A.58: Algorithmic constraint well-formedness

$\boxed{\Theta \models {}^{(\forall)}W}$ Under Θ (input), constraint ${}^{(\forall)}W$ (input) algorithmically holds

$$\frac{\Theta \vdash \varphi\ \mathsf{true}}{\Theta \models \varphi}\ \models W\mathrm{Prp} \qquad \frac{\overset{\mathsf{d}}{\div}\Theta \vdash u \equiv t : \tau}{\Theta \models u \equiv_\tau t}\ \models W\mathrm{IxEq} \qquad \frac{\overset{\mathsf{d}}{\div}\Theta; [\tau] \vdash \boldsymbol{t} \equiv \boldsymbol{t}' : \kappa}{\Theta \models \boldsymbol{t} \equiv_{[\tau]} \boldsymbol{t}'}\ \models W\mathrm{IxSpineEq}$$

$$\frac{\Theta \models {}^{(\forall)}W_1 \qquad \Theta \models {}^{(\forall)}W_2}{\Theta \models {}^{(\forall)}W_1 \wedge {}^{(\forall)}W_2}\ \models W\wedge \qquad \frac{\Theta \models W_k \text{ for some } W_k \in \overrightarrow{W}}{\Theta \models \bigvee \overrightarrow{W}}\ \models W\vee$$

$$\frac{\Theta, \varphi \models {}^{(\supset)}W}{\Theta \models \varphi \supset {}^{(\supset)}W}\ \models W\supset \qquad \frac{\Theta, a \overset{\mathsf{d}}{\div} \tau \models {}^{(\forall)}W}{\Theta \models \forall a \overset{\mathsf{d}}{\div} \tau.\, {}^{(\forall)}W}\ \models W\forall \qquad \frac{\Theta \vdash A <:^{\pm} B}{\Theta \models \underline{A <:^{\pm} B}}\ \models W<:^{\pm}$$

Figure A.59: Subtyping constraint checking

$\boxed{\Theta; \Gamma \lhd \chi}$ Under (ground) Θ and Γ (inputs), constraints χ (input) algorithmically hold

$$\frac{}{\Theta; \Gamma \lhd \cdot}\ \lhd\mathrm{Empty} \qquad \frac{\Theta \models W \qquad \Theta; \Gamma \lhd \chi}{\Theta; \Gamma \lhd W, \chi}\ \lhd W$$

$$\frac{\Theta; \Gamma \rhd e \Leftarrow N \qquad \Theta; \Gamma \lhd \chi}{\Theta; \Gamma \lhd (e \Leftarrow N), \chi}\ \lhd\mathrm{NegChk}$$

Figure A.60: Constraint verification

$\boxed{\hat{\Theta} \vdash W\ \mathsf{Inst}\blacktriangleright \dashv \hat{\Theta}'}$ Input W instantiates input $\hat{\Theta}$ to output $\hat{\Theta}'$

$$\frac{\text{no other rule applies}}{\hat{\Theta} \vdash W\ \mathsf{Inst}\blacktriangleright \dashv \hat{\Theta}} \qquad \frac{\hat{\Theta} \vdash W\ \mathsf{Inst}\blacktriangleright \dashv \hat{\Theta}'' \qquad \hat{\Theta}'' \vdash W'\ \mathsf{Inst}\blacktriangleright \dashv \hat{\Theta}'}{\hat{\Theta} \vdash W \wedge W'\ \mathsf{Inst}\blacktriangleright \dashv \hat{\Theta}'}$$

$$\frac{\overset{\mathsf{d}}{\div}\|\hat{\Theta}_1\| \vdash t : \kappa}{\hat{\Theta}_1, \blacktriangleright\hat{a} \overset{\mathsf{d}}{\div} \kappa, \hat{\Theta}_2 \vdash t \equiv_\kappa \hat{a}\ \mathsf{Inst}\blacktriangleright \dashv \hat{\Theta}_1, \blacktriangleright\hat{a} : \kappa{=}t, \hat{\Theta}_2}$$

$\boxed{\hat{\Theta} \vdash {}^{(\forall)}W\ \mathsf{Inst} \dashv \hat{\Theta}'}$ Input ${}^{(\forall)}W$ instantiates input $\hat{\Theta}$ to output $\hat{\Theta}'$

$$\frac{\text{no other rule applies}}{\hat{\Theta} \vdash {}^{(\forall)}W\ \mathsf{Inst} \dashv \hat{\Theta}} \qquad \frac{\hat{\Theta} \vdash {}^{(\forall)}W_1\ \mathsf{Inst} \dashv \hat{\Theta}'' \qquad \hat{\Theta}'' \vdash {}^{(\forall)}W_2\ \mathsf{Inst} \dashv \hat{\Theta}'}{\hat{\Theta} \vdash {}^{(\forall)}W_1 \wedge {}^{(\forall)}W_2\ \mathsf{Inst} \dashv \hat{\Theta}'}$$

$$\frac{\begin{array}{c}\overset{\mathsf{d}}{\div}\|\hat{\Theta}\|, \blacktriangleright\hat{\Theta} \vdash t : \kappa \\ \hline \hat{\Theta}_1, \hat{a} \overset{\mathsf{d}}{\div} \kappa, \hat{\Theta}_2 \vdash \hat{a} = t\ \mathsf{Inst} \dashv \hat{\Theta}_1, \hat{a} : \kappa{=}t, \hat{\Theta}_2 \\ \underbrace{\hat{\Theta}_1, \hat{a} \overset{\mathsf{d}}{\div} \kappa, \hat{\Theta}_2}_{\hat{\Theta}} \vdash t = \hat{a}\ \mathsf{Inst} \dashv \hat{\Theta}_1, \hat{a} : \kappa{=}t, \hat{\Theta}_2\end{array}}{}$$

$$\frac{\hat{\Theta} \vdash u_1 = t_1\ \mathsf{Inst} \dashv \hat{\Theta}'' \qquad \hat{\Theta}'' \vdash u_2 = t_2\ \mathsf{Inst} \dashv \hat{\Theta}'}{\hat{\Theta} \vdash (u_1, u_2) = (t_1, t_2)\ \mathsf{Inst} \dashv \hat{\Theta}'} \qquad \frac{\hat{\Theta} \vdash \varphi_1\ \mathsf{Inst} \dashv \hat{\Theta}'' \qquad \hat{\Theta}'' \vdash \varphi_2\ \mathsf{Inst} \dashv \hat{\Theta}'}{\hat{\Theta} \vdash \varphi_1 \wedge \varphi_2\ \mathsf{Inst} \dashv \hat{\Theta}'}$$

$\boxed{\hat{\Theta} \vdash W\ \mathsf{fixInst} \dashv \hat{\Theta}'}$ The fixed point of input W instantiations starting at input $\hat{\Theta}$ is output $\hat{\Theta}'$

$$\frac{\hat{\Theta} \vdash W\ \mathsf{Inst} \dashv \hat{\Theta}}{\hat{\Theta} \vdash W\ \mathsf{fixInst} \dashv \hat{\Theta}} \qquad \frac{\hat{\Theta} \vdash W\ \mathsf{Inst} \dashv \hat{\Theta}'' \qquad \hat{\Theta}'' \neq \hat{\Theta} \qquad \hat{\Theta}'' \vdash [\hat{\Theta}'']^2 W\ \mathsf{fixInst} \dashv \hat{\Theta}'}{\hat{\Theta} \vdash W\ \mathsf{fixInst} \dashv \hat{\Theta}'}$$

$\boxed{\begin{array}{l}\hat{\Theta}; \cdot \vdash W\ \mathsf{fixInstChk} \dashv \Omega \\ \hat{\Theta}; \Gamma \vdash \chi\ \mathsf{fixInstChk} \dashv \Omega\end{array}}$ Under input(s) $\hat{\Theta}$ (and Γ) the input constraints W (χ) algorithmically hold at output solutions Ω

$$\frac{\begin{array}{c}\hat{\Theta} \vdash W\ \mathsf{fixInst} \dashv \hat{\Theta}' \qquad \mathsf{select}\ \overrightarrow{W_o}\ \mathsf{from}\ [\hat{\Theta}']^2 W \\ \hat{\Theta}' \vdash \wedge\overrightarrow{W_o}\ \mathsf{Inst}\blacktriangleright \dashv \Omega \qquad \|\hat{\Theta}\| \models [\Omega]^2 W\end{array}}{\hat{\Theta}; \cdot \vdash W\ \mathsf{fixInstChk} \dashv \Omega}$$

$$\frac{\begin{array}{c}\hat{\Theta} \vdash \lfloor\chi\rfloor\ \mathsf{fixInst} \dashv \hat{\Theta}' \qquad \mathsf{select}\ \overrightarrow{W_o}\ \mathsf{from}\ [\hat{\Theta}']^2 \lfloor\chi\rfloor \\ \hat{\Theta}' \vdash \wedge\overrightarrow{W_o}\ \mathsf{Inst}\blacktriangleright \dashv \Omega \qquad \|\hat{\Theta}\|; \Gamma \lhd [\Omega]^2 \chi\end{array}}{\hat{\Theta}; \Gamma \vdash \chi\ \mathsf{fixInstChk} \dashv \Omega}$$

where

$$\frac{\mathsf{select}\ \overrightarrow{W}\ \mathsf{from}\ {}^{(\forall)}W \qquad \mathsf{select}\ \overrightarrow{W'}\ \mathsf{from}\ {}^{(\forall)}W'}{\mathsf{select}\ \overrightarrow{W}, \overrightarrow{W'}\ \mathsf{from}\ {}^{(\forall)}W \wedge {}^{(\forall)}W'} \qquad \frac{(_ \wedge t \equiv_\kappa \hat{a}) \in \overrightarrow{W}}{\mathsf{select}\ t \equiv_\kappa \hat{a}\ \mathsf{from} \bigwedge \overrightarrow{W}}$$

$$\frac{{}^{(\forall)}W \neq - \wedge -\ \text{and}\ {}^{(\forall)}W \neq \bigwedge \overrightarrow{W}\ \text{with}\ _ \wedge _ \equiv__ \hat{a} \in \overrightarrow{W}}{\mathsf{select} \cdot \mathsf{from}\ {}^{(\forall)}W}$$

Figure A.61: Constraint instantiation

$\hat{\Theta} \vdash R <:^{+} Q \mathbin{/} {}^{(\forall)}W$
$\hat{\Theta} \vdash M <:^{-} L \mathbin{/} {}^{(\forall)}W$
$\Theta \vdash A <:^{\pm} B$

Under $\hat{\Theta}$, type R (M) is algorithmically a subtype of Q (L) (all inputs) if output constraint ${}^{(\forall)}W$ holds algorithmically for some solutions;
Under Θ type A is algorithmically a subtype of B (inputs: Θ, A, B)

$$\frac{}{\hat{\Theta} \vdash 0 <:^{+} 0 \mathbin{/} \mathsf{tt}} <:^{+}0 \qquad \frac{}{\hat{\Theta} \vdash 1 <:^{+} 1 \mathbin{/} \mathsf{tt}} <:^{+}1$$

$$\frac{\hat{\Theta} \vdash R_1 <:^{+} R_1' \mathbin{/} W_1 \qquad \hat{\Theta} \vdash R_2 <:^{+} R_2' \mathbin{/} W_2}{\hat{\Theta} \vdash R_1 \times R_2 <:^{+} R_1' \times R_2' \mathbin{/} W_1 \wedge W_2} <:^{+}\times$$

$$\frac{P_1 = \exists^{\mathsf{d}} \Xi_1 . R_1 \wedge \overrightarrow{\varphi_1} \qquad P_2 = \exists^{\mathsf{d}} \Xi_2 . R_2 \wedge \overrightarrow{\varphi_2}}{\hat{\Theta} \vdash P_1 + P_2 <:^{+} P_1' + P_2' \mathbin{/} (\forall^{\mathsf{d}} \Xi_1 . \overrightarrow{\varphi_1} \supset \underline{R_1 <:^{+} P_1'}) \wedge (\forall^{\mathsf{d}} \Xi_2 . \overrightarrow{\varphi_2} \supset \underline{R_2 <:^{+} P_2'})} <:^{+}+$$

$$\frac{\hat{\Theta} \vdash R <:^{+} R' \mathbin{/} W}{\hat{\Theta} \vdash R <:^{+} R' \wedge \overrightarrow{\varphi} \mathbin{/} W \wedge \overrightarrow{\varphi}} <:^{+}\wedge\mathrm{R}$$

$$\frac{\hat{\Theta} \vdash \mathscr{M}'(F') \geq \mathscr{M}(F) \mathbin{/} W}{\hat{\Theta} \vdash \left\{ \nu : \mu F' \;\middle|\; \mathscr{M}'(F') \right\} <:^{+} \{ \nu : \mu F \mid \mathscr{M}(F) \} \mathbin{/} W} <:^{+}\mu$$

$$\frac{}{\hat{\Theta} \vdash \downarrow N <:^{+} \downarrow (\forall^{\mathsf{d}} \Xi . \overrightarrow{\varphi} \supset L) \mathbin{/} \forall^{\mathsf{d}} \Xi . \overrightarrow{\varphi} \supset \underline{N <:^{-} L}} <:^{+}\downarrow$$

$$\frac{{}^{\mathsf{d}}\Xi \text{ may be } \cdot \qquad \Theta, \widehat{{}^{\mathsf{d}}\Xi} \vdash R <:^{+} [\widehat{{}^{\mathsf{d}}\Xi}/{}^{\mathsf{d}}\Xi] Q \mathbin{/} W \qquad \overline{\Theta}, \widehat{{}^{\mathsf{d}}\Xi} \vdash W \ \mathsf{wf}[\xi_W] \qquad \Theta, [\xi_W]\widehat{{}^{\mathsf{d}}\Xi}; \cdot \vdash W \ \mathsf{fixInstChk} \dashv \Omega}{\Theta \vdash R <:^{+} \exists^{\mathsf{d}} \Xi . Q} <:^{+}[\exists]$$

$$\frac{}{\hat{\Theta} \vdash \uparrow (\exists^{\mathsf{d}} \Xi . R \wedge \overrightarrow{\varphi}) <:^{-} \uparrow P \mathbin{/} \forall^{\mathsf{d}} \Xi . \overrightarrow{\varphi} \supset \underline{R <:^{+} P}} <:^{-}\uparrow$$

$$\frac{\hat{\Theta} \vdash L' <:^{-} L \mathbin{/} W}{\hat{\Theta} \vdash \overrightarrow{\varphi} \supset L' <:^{-} L \mathbin{/} W \wedge \overrightarrow{\varphi}} <:^{-}\supset\mathrm{L}$$

$$\frac{\hat{\Theta} \vdash R' <:^{+} R \mathbin{/} W_1 \qquad \hat{\Theta} \vdash L <:^{-} L' \mathbin{/} W_2}{\hat{\Theta} \vdash R \to L <:^{-} R' \to L' \mathbin{/} W_1 \wedge W_2} <:^{-}\to$$

$$\frac{{}^{\mathsf{d}}\Xi \text{ may be } \cdot \qquad \Theta, \widehat{{}^{\mathsf{d}}\Xi} \vdash [\widehat{{}^{\mathsf{d}}\Xi}/{}^{\mathsf{d}}\Xi] N <:^{-} L \mathbin{/} W \qquad \overline{\Theta}, \widehat{{}^{\mathsf{d}}\Xi} \vdash W \ \mathsf{wf}[\xi_W] \qquad \Theta, [\xi_W]\widehat{{}^{\mathsf{d}}\Xi}; \cdot \vdash W \ \mathsf{fixInstChk} \dashv \Omega}{\Theta \vdash \forall^{\mathsf{d}} \Xi . M <:^{-} L} <:^{-}[\forall]$$

Figure A.62: Algorithmic subtyping

$\boxed{\hat{\Theta} \vdash \mathscr{M}'(F') \geq \mathscr{M}(F) \mathbin{/} W}$ Under input $\hat{\Theta}$, measurements $\mathscr{M}'(F')$ cover $\mathscr{M}(F)$ (inputs) if output W holds for some solutions

$$\frac{\cdot \rhd \underline{\mathsf{tt}}^{(F')}; F' <:_{\mathbb{B}} \underline{\mathsf{tt}}^{(F)}; F}{\hat{\Theta} \vdash \mathscr{M}'(F') \geq \cdot_F \mathbin{/} \mathsf{tt}}$$

$$\frac{\hat{\Theta} \vdash \mathscr{M}'(F') \geq \mathscr{M}(F) \mathbin{/} W \qquad \text{let } \overrightarrow{W} = \left\{ \boldsymbol{t}' \equiv_{[\tau]} \boldsymbol{t} \wedge t' \equiv_{\kappa} t \;\middle|\; \begin{array}{c} \cdot \rhd \alpha'; F' <:_{\tau} \alpha; F \text{ and } \hat{\Theta}; [\tau] \rhd \boldsymbol{t}' : \kappa \\ \text{for some } (\mathsf{fold}_{F'}\ \alpha')\, \nu\, \boldsymbol{t}' =_{\tau} t' \in \mathscr{M}'(F') \end{array} \right\}}{\hat{\Theta} \vdash \mathscr{M}'(F') \geq \mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\, \nu\, \boldsymbol{t} =_{\tau} t \mathbin{/} W \wedge \left(\bigvee \overrightarrow{W} \right)}$$

Figure A.63: Algorithmic measurement covering

$\boxed{\Xi \rhd \alpha; F <:_\tau \beta; G}$ Under Ξ, algebra $\alpha : F(\tau) \Rightarrow \tau$ is algorithmically a submeasure of $\beta : G(\tau) \Rightarrow \tau$ (Inputs: $\Xi, \alpha, F, \tau, \beta, G$)

$$\frac{\begin{array}{ccc}\alpha \circ \mathsf{inj}_1 \mathrel{\dot{=}} \alpha_1 & \beta \circ \mathsf{inj}_1 \mathrel{\dot{=}} \beta_1 & \Xi \rhd \alpha_1; F_1 <:_\tau \beta_1; G_1 \\ \alpha \circ \mathsf{inj}_2 \mathrel{\dot{=}} \alpha_2 & \beta \circ \mathsf{inj}_2 \mathrel{\dot{=}} \beta_2 & \Xi \rhd \alpha_2; F_2 <:_\tau \beta_2; G_2\end{array}}{\Xi \rhd \alpha; F_1 \oplus F_2 <:_\tau \beta; G_1 \oplus G_2}\ \text{Meas}\rhd<:/\oplus$$

$$\frac{\begin{array}{c}{}^{\mathsf{d}}\Xi' \neq \cdot \\ \Xi, {}^{\mathsf{d}}\Xi, \overrightarrow{\varphi}, \widehat{{}^{\mathsf{d}}\Xi'} \vdash R <:^{+} [\widehat{{}^{\mathsf{d}}\Xi'}/{}^{\mathsf{d}}\Xi']Q' \mathbin{/} {}^{(\forall)}W \\ \Xi, {}^{\mathsf{d}}\Xi, \widehat{{}^{\mathsf{d}}\Xi'} \vdash {}^{(\forall)}W\ \mathsf{wf}[\xi] \\ \Xi, {}^{\mathsf{d}}\Xi, \overrightarrow{\varphi}, [\xi]\widehat{{}^{\mathsf{d}}\Xi'}; \cdot \vdash {}^{(\forall)}W\ \mathsf{fixInstChk} \dashv \Omega \\ \Xi, {}^{\mathsf{d}}\Xi \rhd q \Rightarrow t; \hat{P} <:_\tau q' \Rightarrow [\Omega]^2[\widehat{{}^{\mathsf{d}}\Xi'}/{}^{\mathsf{d}}\Xi']t'; \hat{P}'\end{array}}{\Xi \rhd (\mathtt{pk}({}^{\mathsf{d}}\Xi, \top), q) \Rightarrow t; \underline{\exists {}^{\mathsf{d}}\Xi. R \wedge \overrightarrow{\varphi}} \otimes \hat{P} <:_\tau (\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q') \Rightarrow t'; \underline{\exists {}^{\mathsf{d}}\Xi'. Q'} \otimes \hat{P}'}\ \text{Meas}\rhd<:/\exists\text{R}$$

$$\frac{\begin{array}{c}({}^{\mathsf{d}}\Xi \text{ may be } \cdot \text{ and } \overrightarrow{\varphi} \text{ may be } \cdot) \\ \Xi, {}^{\mathsf{d}}\Xi, \overrightarrow{\varphi} \vdash R <:^{+} Q' \\ \Xi, {}^{\mathsf{d}}\Xi \rhd q \Rightarrow t; \hat{P} <:_\tau q' \Rightarrow t'; \hat{P}'\end{array}}{\Xi \rhd (\mathtt{pk}({}^{\mathsf{d}}\Xi, \top), q) \Rightarrow t; \underline{\exists {}^{\mathsf{d}}\Xi. R \wedge \overrightarrow{\varphi}} \otimes \hat{P} <:_\tau (\top, q') \Rightarrow t'; \underline{Q'} \otimes \hat{P}'}\ \text{Meas}\rhd<:/\text{Const}$$

$$\frac{\Xi, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id} \rhd q \Rightarrow t; \hat{I} <:_\tau q' \Rightarrow t'; \hat{I}}{\Xi \rhd (a, q) \Rightarrow t; \mathrm{Id} \otimes \hat{I} <:_\tau (a, q') \Rightarrow t'; \mathrm{Id} \otimes \hat{I}}\ \text{Meas}\rhd<:/\mathrm{Id}$$

$$\frac{\overset{\mathsf{d}}{\div}\Xi \vdash u \equiv t : \tau}{\Xi \rhd () \Rightarrow u; I <:_\tau () \Rightarrow t; I}\ \text{Meas}\rhd<:/I$$

Figure A.64: Algorithmic submeasuring

$\boxed{\hat{\Xi} \rhd \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta; R}$ Unrolling yields type $\exists^{\mathsf{d}}\Theta.\,(R \wedge {}^{\mathsf{d}}\Theta)$
(inputs: left of $\doteq$; outputs: right of $\doteq$)

$$\frac{\begin{array}{cc}\overrightarrow{\beta} \circ \mathsf{inj}_1 \doteq \overrightarrow{\beta_1} & \hat{\Xi} \rhd \wr \overrightarrow{\beta_1}; G_1; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta_1; R_1 \\ \overrightarrow{\beta} \circ \mathsf{inj}_2 \doteq \overrightarrow{\beta_2} & \hat{\Xi} \rhd \wr \overrightarrow{\beta_2}; G_2; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta_2; R_2\end{array}}{\hat{\Xi} \rhd \wr \overrightarrow{\beta}; G_1 \oplus G_2; \mathscr{M}(F) \int \doteq \cdot; (\exists^{\mathsf{d}}\Theta_1.\,(R_1 \wedge {}^{\mathsf{d}}\Theta_1)) + (\exists^{\mathsf{d}}\Theta_2.\,(R_2 \wedge {}^{\mathsf{d}}\Theta_2))}\ \mathrm{Alg}\wr\oplus\int$$

($^{\mathsf{d}}\Xi'$ may be $\cdot$ and $\overrightarrow{\varphi}$ may be $\cdot$)

$$\frac{\overrightarrow{\beta} \rightsquigarrow \overrightarrow{\beta'} \qquad {}^{\mathsf{d}}\Xi', \hat{\Xi} \vdash \wr \overrightarrow{\beta'}; \hat{P}; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta; R}{\hat{\Xi} \rhd \wr \overrightarrow{\beta}; \underline{\exists^{\mathsf{d}}\Xi'.\,R' \wedge \overrightarrow{\varphi}} \otimes \hat{P}; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Xi', {}^{\mathsf{d}}\Theta, \overrightarrow{\varphi}; R' \times R}\ \mathrm{Alg}\wr\mathrm{Const}\int$$

$$\frac{\begin{array}{c}\overrightarrow{a \stackrel{\mathsf{d}}{\div} \tau} = \overrightarrow{a} \stackrel{\mathsf{d}}{\div} \mathscr{M}(F) \\ \overrightarrow{a \stackrel{\mathsf{d}}{\div} \tau, a\,\mathrm{Id}}, \hat{\Xi} \rhd \wr \overrightarrow{q \Rightarrow t'}; \hat{I}; \mathscr{M}(F) \int \doteq \Xi'', \overrightarrow{\psi''}; R'' \\ \hat{\Xi}; \Xi''; zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash \overrightarrow{\psi''} \rightsquigarrow \breve{\Xi}_1; \mathscr{M}_1(F); \overrightarrow{\psi'} \\ \hat{\Xi}; \Xi''; zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash R'' \rightsquigarrow \breve{\Xi}_2; \mathscr{M}_2(F); R' \\ \breve{\Xi}' = \breve{\Xi}_1 \cup \breve{\Xi}_2 \qquad \mathscr{M}'(F) = \mathscr{M}_1(F) \cup \mathscr{M}_2(F)\end{array} \quad dom(\Xi') \cap dom(\hat{\Xi}, \overrightarrow{a \stackrel{\mathsf{d}}{\div} \tau}, \Xi'', \breve{\Xi}') = \emptyset \qquad \rho \text{ is the variable renaming } \Xi'/\breve{\Xi}'}{\hat{\Xi} \rhd \wr \overrightarrow{(a,q) \Rightarrow t'}; \mathrm{Id} \otimes \hat{I}; \mathscr{M}(F) \int \doteq \Xi', \Xi'', [\rho]\overrightarrow{\psi'}; \{\nu : \mu F \mid [\rho]\mathscr{M}'(F)\} \times [\rho]R'}\ \mathrm{Alg}\wr\mathrm{Id}\int$$

$$\frac{\overrightarrow{t'} @ \mathscr{M}(F) \doteq \overrightarrow{\varphi}}{\hat{\Xi} \rhd \wr \overrightarrow{() \Rightarrow t'}; I; \mathscr{M}(F) \int \doteq \overrightarrow{\varphi}; 1}\ \mathrm{Alg}\wr I\int$$

Figure A.65: Algorithmic unrolling

$\boxed{\Theta;\Gamma \rhd h \Rightarrow P}$ Under input Θ and Γ, input head h synthesizes (output) type P

$$\frac{(x : R) \in \Gamma}{\Theta;\Gamma \rhd x \Rightarrow R}\ \text{Alg}\Rightarrow\text{Var} \qquad \frac{\overline{\Theta} \vdash P\ \mathsf{type}[\xi] \qquad \Theta;\Gamma \rhd v \Leftarrow P}{\Theta;\Gamma \rhd (v : P) \Rightarrow P}\ \text{Alg}\Rightarrow\text{ValAnnot}$$

$\boxed{\Theta;\Gamma \rhd g \Rightarrow \uparrow P}$ Under inputs Θ and Γ, input bound expression g synthesizes (output) $\uparrow P$ (possibly non-deterministically)

$$\frac{\Theta;\Gamma \rhd h \Rightarrow \downarrow N \qquad \Theta;\Gamma;[N] \rhd s \Rightarrow \uparrow P}{\Theta;\Gamma \rhd h(s) \Rightarrow \uparrow P}\ \text{Alg}\Rightarrow\text{App}$$

$$\frac{\overline{\Theta} \vdash P\ \mathsf{type}[\xi] \qquad \Theta;\Gamma \rhd e \Leftarrow \uparrow P}{\Theta;\Gamma \rhd (e : \uparrow P) \Rightarrow \uparrow P}\ \text{Alg}\Rightarrow\text{ExpAnnot}$$

Figure A.66: Algorithmic head and bound expression synthesis

$\boxed{\begin{array}{l}\Theta;\Gamma \rhd v \Leftarrow P \\ \hat{\Theta};\Gamma \vdash v \Leftarrow P \mid \chi \dashv \Delta'\end{array}}$ Under inputs $\hat{\Theta}$ and Γ, input value v checks against input type P, with (outputs) computation constraints χ and new unsolved Δ' (output)

$$\frac{(x : R') \in \Gamma \qquad \hat{\Theta} \vdash R' <:^{+} R \mid W}{\hat{\Theta};\Gamma \vdash x \Leftarrow R \mid W \dashv \cdot}\ \mathrm{Alg}{\Leftarrow}\mathrm{Var} \qquad \frac{}{\hat{\Theta};\Gamma \vdash \langle\rangle \Leftarrow 1 \mid \cdot \dashv \cdot}\ \mathrm{Alg}{\Leftarrow}1$$

$$\frac{\hat{\Theta};\Gamma \vdash v_1 \Leftarrow R_1 \mid \chi_1 \dashv \Delta_1 \qquad \hat{\Theta};\Gamma \vdash v_2 \Leftarrow R_2 \mid \chi_2 \dashv \Delta_2}{\hat{\Theta};\Gamma \vdash \langle v_1, v_2\rangle \Leftarrow (R_1 \times R_2) \mid \chi_1, \chi_2 \dashv \Delta_1, \Delta_2}\ \mathrm{Alg}{\Leftarrow}\times$$

$$\frac{\hat{\Theta};\Gamma \vdash v \Leftarrow P_k \mid \chi \dashv \Delta}{\hat{\Theta};\Gamma \vdash \mathrm{inj}_k\, v \Leftarrow (P_1 + P_2) \mid \chi \dashv \Delta}\ \mathrm{Alg}{\Leftarrow}{+_k}$$

$$\frac{\hat{\Theta}, \widehat{{}^{\mathsf{d}}\Xi};\Gamma \vdash v \Leftarrow [\widehat{{}^{\mathsf{d}}\Xi}/{}^{\mathsf{d}}\Xi]Q \mid \chi \dashv \Delta}{\hat{\Theta};\Gamma \vdash v \Leftarrow (\exists^{\mathsf{d}}\Xi.\, Q) \mid \chi \dashv \widehat{{}^{\mathsf{d}}\Xi}, \Delta}\ \mathrm{Alg}{\Leftarrow}\exists$$

$$\frac{\nexists x.\, v = \overrightarrow{\mathrm{inj}_{k_i}}^{i}\left(\overrightarrow{\langle __j, - \rangle}^{j} x\right) \qquad \mathscr{M}(F) \rightsquigarrow \overrightarrow{\alpha}; \overrightarrow{\tau} \qquad \overset{\mathsf{d}}{\div}\hat{\Theta} \rhd \wr \overrightarrow{\alpha}; F; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta; R \qquad \hat{\Theta};\Gamma \vdash v \Leftarrow \exists^{\mathsf{d}}\Theta.\, R \wedge {}^{\mathsf{d}}\Theta \mid \chi \dashv \Delta}{\hat{\Theta};\Gamma \vdash \mathrm{into}(v) \Leftarrow \{\nu : \mu F \mid \mathscr{M}(F)\} \mid \chi \dashv \Delta}\ \mathrm{Alg}{\Leftarrow}\mu$$

$$\frac{}{\hat{\Theta};\Gamma \vdash \{e\} \Leftarrow {\downarrow}N \mid (e \Leftarrow N) \dashv \cdot}\ \mathrm{Alg}{\Leftarrow}{\downarrow} \qquad \frac{\hat{\Theta};\Gamma \vdash v \Leftarrow R \mid \chi \dashv \Delta}{\hat{\Theta};\Gamma \vdash v \Leftarrow (R \wedge \overrightarrow{\varphi}) \mid (\overrightarrow{\varphi}, \chi) \dashv \Delta}\ \mathrm{Alg}{\Leftarrow}\wedge$$

$$\frac{\Theta;\Gamma \vdash v \Leftarrow P \mid \chi \dashv \Delta \qquad \overline{\Theta}, \Delta \vdash \chi\ \mathsf{Wf}[\xi] \qquad \Theta, [\xi]\Delta;\Gamma \vdash \chi\ \mathsf{fixInstChk} \dashv \Omega}{\Theta;\Gamma \rhd v \Leftarrow P}\ \mathrm{Alg}{\Leftarrow}\mathrm{Val}$$

Figure A.67: Algorithmic value checking

$\boxed{\Theta;\Gamma \rhd e \Leftarrow N}$ Under inputs Θ and Γ, input expression e checks against input N

$$\frac{\Theta;\Gamma \rhd v \Leftarrow P}{\Theta;\Gamma \rhd \mathsf{return}\, v \Leftarrow \uparrow P}\ \mathrm{Alg}{\Leftarrow}{\uparrow}$$

$$\frac{\Theta, {}^{\mathsf{d}}\Xi, \overrightarrow{\psi};\Gamma, x : R \rhd e \Leftarrow L \text{ for some } (\exists^{\mathsf{d}}\Xi.R \wedge \overrightarrow{\psi}) \in \{P \mid \Theta;\Gamma \rhd g \Rightarrow \uparrow P\}}{\Theta;\Gamma \rhd \mathsf{let}\ x = g;\ e \Leftarrow L}\ \mathrm{Alg}{\Leftarrow}\mathsf{let}$$

$$\frac{\Theta;\Gamma \rhd h \Rightarrow P \qquad \Theta;\Gamma;[P] \rhd \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow L}{\Theta;\Gamma \rhd \mathsf{match}\ h\ \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow L}\ \mathrm{Alg}{\Leftarrow}\mathsf{match}$$

$$\frac{\Theta;\Gamma, x : R \rhd e \Leftarrow L}{\Theta;\Gamma \rhd \lambda x.e \Leftarrow R \to L}\ \mathrm{Alg}{\Leftarrow}\lambda \qquad \frac{\Theta \vdash \mathsf{ff}\ \mathsf{true}}{\Theta;\Gamma \rhd \mathsf{unreachable} \Leftarrow L}\ \mathrm{Alg}{\Leftarrow}\mathrm{Unreachable}$$

$$\frac{\begin{array}{c}\Theta \vdash \forall a \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.M <:^{-} L \\ \Theta, a \div \mathbb{N};\Gamma, x : \downarrow \forall a' \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.a' < a \supset [a'/a]M \rhd e \Leftarrow \forall^{\mathsf{d}}\Xi.M\end{array}}{\Theta;\Gamma \rhd \mathsf{rec}\, x : (\forall a \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.M).\ e \Leftarrow L}\ \mathrm{Alg}{\Leftarrow}\mathsf{rec}$$

$$\frac{\Theta, {}^{\mathsf{d}}\Xi;\Gamma \rhd e \Leftarrow N}{\Theta;\Gamma \rhd e \Leftarrow \forall^{\mathsf{d}}\Xi.N}\ \mathrm{Alg}{\Leftarrow}\forall \qquad \frac{\Theta, \overrightarrow{\varphi};\Gamma \rhd e \Leftarrow L}{\Theta;\Gamma \rhd e \Leftarrow \overrightarrow{\varphi} \supset L}\ \mathrm{Alg}{\Leftarrow}\supset$$

Figure A.68: Algorithmic expression checking

$\boxed{\Theta;\Gamma;[P] \rhd \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}$ Under Θ and Γ, patterns r_i match against type P and branch expressions e_i check against type N (all inputs)

$$\frac{\Theta, {}^{\mathsf{d}}\Xi;\Gamma;[Q] \rhd \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}{\Theta;\Gamma;[\exists^{\mathsf{d}}\Xi.\, Q] \rhd \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}\ \text{AlgMatch}\exists$$

$$\frac{\Theta, \overrightarrow{\varphi};\Gamma;[R] \rhd \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}{\Theta;\Gamma;[R \wedge \overrightarrow{\varphi}] \rhd \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}\ \text{AlgMatch}\wedge \qquad \frac{\Theta;\Gamma \rhd e \Leftarrow N}{\Theta;\Gamma;[1] \rhd \{\langle\rangle \Rightarrow e\} \Leftarrow N}\ \text{AlgMatch1}$$

$$\frac{\Theta;\Gamma, x_1 : R_1, x_2 : R_2 \rhd e \Leftarrow N}{\Theta;\Gamma;[R_1 \times R_2] \rhd \{\langle x_1, x_2\rangle \Rightarrow e\} \Leftarrow N}\ \text{AlgMatch}\times$$

$$\frac{\Theta, {}^{\mathsf{d}}\Xi_1, \overrightarrow{\psi}_1;\Gamma, x_1 : R_1 \rhd e_1 \Leftarrow N \qquad \Theta, {}^{\mathsf{d}}\Xi_2, \overrightarrow{\psi}_2;\Gamma, x_2 : R_2 \rhd e_2 \Leftarrow N}{\Theta;\Gamma;[(\exists^{\mathsf{d}}\Xi_1.\, R_1 \wedge \overrightarrow{\psi}_1) + (\exists^{\mathsf{d}}\Xi_2.\, R_2 \wedge \overrightarrow{\psi}_2)] \rhd \{\mathtt{inj}_1\, x_1 \Rightarrow e_1 \mid \mathtt{inj}_2\, x_2 \Rightarrow e_2\} \Leftarrow N}\ \text{AlgMatch}+$$

$$\frac{}{\Theta;\Gamma;[0] \rhd \{\} \Leftarrow N}\ \text{AlgMatch0}$$

$$\frac{\mathscr{M}(F) \rightsquigarrow \overrightarrow{\alpha};\overrightarrow{\tau} \qquad {}^{\mathsf{d}\div}\Theta \rhd \wr\, \overrightarrow{\alpha};F;\mathscr{M}(F)\, \int \doteq {}^{\mathsf{d}}\Theta;R \qquad \Theta, {}^{\mathsf{d}}\Theta;\Gamma, x : R \rhd e \Leftarrow N}{\Theta;\Gamma;[\{\nu : \mu F \mid \mathscr{M}(F)\}] \rhd \{\mathtt{into}(x) \Rightarrow e\} \Leftarrow N}\ \text{AlgMatch}\mu$$

Figure A.69: Algorithmic pattern matching

$\Theta;\Gamma;[N] \rhd s \Rightarrow \uparrow P$
$\hat{\Theta};\Gamma;[M] \vdash s \Rightarrow \uparrow P \mathbin{/} \chi \dashv \Delta$

Inputs Θ,Γ,N,s nondeterministically synthesize $\uparrow P$ (output);
Under $\hat{\Theta}$ and Γ,
applying s to a head of type $\downarrow M$ nondeterministically infers $\uparrow P$,
completable if χ holds for some $\hat{\Theta},\Delta$ solutions
(inputs: $\hat{\Theta},\Gamma,M,s$; outputs: $\uparrow P,\chi,\Delta$)

$$\frac{{}^{\mathsf{d}}\Xi \text{ may be } \cdot \qquad \Theta,\widehat{{}^{\mathsf{d}}\Xi};\Gamma;[[\widehat{{}^{\mathsf{d}}\Xi}/{}^{\mathsf{d}}\Xi]M] \vdash s \Rightarrow \uparrow P \mathbin{/} \chi \dashv \Delta \qquad \overline{\Theta},\widehat{{}^{\mathsf{d}}\Xi},\Delta \vdash \chi\ \mathsf{Wf}[\xi] \qquad \Theta,[\xi](\widehat{{}^{\mathsf{d}}\Xi},\Delta);\Gamma \vdash \chi\ \mathsf{fixInstChk} \dashv \Omega}{\Theta;\Gamma;[\forall {}^{\mathsf{d}}\Xi.M] \rhd s \Rightarrow \uparrow [\Omega][\Omega]P}\ \text{AlgSpine}[\forall]$$

$$\frac{\hat{\Theta};\Gamma;[L] \vdash s \Rightarrow \uparrow P \mathbin{/} \chi \dashv \Delta}{\hat{\Theta};\Gamma;[\overrightarrow{\varphi} \supset L] \vdash s \Rightarrow \uparrow P \mathbin{/} \overrightarrow{\varphi},\chi \dashv \Delta}\ \text{AlgSpine}\supset$$

$$\frac{\hat{\Theta};\Gamma \vdash v \Leftarrow R \mathbin{/} \chi \dashv \Delta \qquad \hat{\Theta};\Gamma;[L] \vdash s \Rightarrow \uparrow P \mathbin{/} \chi' \dashv \Delta'}{\hat{\Theta};\Gamma;[R \to L] \vdash v,s \Rightarrow \uparrow P \mathbin{/} \chi,\chi' \dashv \Delta,\Delta'}\ \text{AlgSpineApp}$$

$$\frac{}{\hat{\Theta};\Gamma;[\uparrow P] \vdash \cdot \Rightarrow \uparrow P \mathbin{/} \mathsf{tt} \dashv \cdot}\ \text{AlgSpineNil}$$

Figure A.70: Algorithmic spine typing

$\boxed{\hat{\Theta} \longrightarrow \hat{\Theta}'}$ Algorithmic context $\hat{\Theta}$ extends to $\hat{\Theta}'$ (inputs)

$$\frac{\begin{array}{c}\|\hat{\Theta}\| = \|\hat{\Theta}'\| \\ \hat{\Theta} \text{ and } \hat{\Theta}' \text{ agree on the sorts and } \blacktriangleright\text{-status of evars} \\ \text{if } [\blacktriangleright]\hat{a} : \kappa{=}t \in \hat{\Theta} \text{ then } [\blacktriangleright]\hat{a} : \kappa{=}t \in \hat{\Theta}'\end{array}}{\hat{\Theta} \longrightarrow \hat{\Theta}'}$$

Figure A.71: Algorithmic context extension

$\boxed{\hat{\Theta} \xrightarrow{\mathsf{SMT}} \hat{\Theta}'}$ Algorithmic context $\hat{\Theta}$ relaxedly extends to $\hat{\Theta}'$ (inputs)

$$\frac{\begin{array}{c}\|\hat{\Theta}\| = \|\hat{\Theta}'\| \\ \hat{\Theta} \text{ and } \hat{\Theta}' \text{ agree on the sorts and } \blacktriangleright\text{-status of evars} \\ \text{if } [\blacktriangleright]\hat{a} : \kappa{=}t \in \hat{\Theta} \text{ then } [\blacktriangleright]\hat{a} : \kappa{=}t \in \hat{\Theta}' \text{ or } \exists([\blacktriangleright]\hat{a} : \kappa{=}t' \in \hat{\Theta}').\, \|\hat{\Theta}\| \vdash [\hat{\Theta}]t = [\hat{\Theta}']t' \ \mathsf{true}\end{array}}{\hat{\Theta} \xrightarrow{\mathsf{SMT}} \hat{\Theta}'}$$

Figure A.72: Relaxed algorithmic context extension

Appendix A.8 Intermediate Systems for Algorithmic Completeness

$\boxed{\Theta \mathrel{\widetilde{\vdash}} \mathscr{M}'(F') \geq \mathscr{M}(F) \mathbin{/} W}$ Under Θ, measurements $\mathscr{M}'(F')$ cover $\mathscr{M}(F)$ if W holds (inputs: $\Theta, \mathscr{M}'(F'), \mathscr{M}(F)$; output: W)

$$\frac{\cdot \mathrel{\widetilde{\vdash}} \underline{\mathsf{tt}}^{(F')}; F' <:_{\mathbb{B}} \underline{\mathsf{tt}}^{(F)}; F}{\Theta \mathrel{\widetilde{\vdash}} \mathscr{M}'(F') \geq \cdot_F \mathbin{/} \mathsf{tt}}$$

$$\frac{\Theta \mathrel{\widetilde{\vdash}} \mathscr{M}'(F') \geq \mathscr{M}(F) \mathbin{/} W \qquad \text{let } \overrightarrow{W} = \left\{ \boldsymbol{t}' \equiv_{[\tau]} \boldsymbol{t} \wedge t' \equiv_{\kappa} t \;\middle|\; \begin{array}{l} \cdot \mathrel{\widetilde{\vdash}} \alpha'; F' <:_{\tau} \alpha; F \text{ and } \overset{\mathsf{d}}{\div}\Theta; [\tau] \vdash \boldsymbol{t}' : \kappa \\ \text{for some } (\mathsf{fold}_{F'}\ \alpha')\, \nu\, \boldsymbol{t}' =_{\tau} t' \in \mathscr{M}'(F') \end{array} \right\}}{\Theta \mathrel{\widetilde{\vdash}} \mathscr{M}'(F') \geq \mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\, \nu\, \boldsymbol{t} =_{\tau} t \mathbin{/} W \wedge \left(\bigvee \overrightarrow{W} \right)}$$

Figure A.73: Semideclarative measurement covering

$\boxed{\begin{array}{l}\Theta \mathrel{\widetilde{\vdash}} R <:^{+} P \mathbin{/} {}^{(\forall)}W \\ \Theta \mathrel{\widetilde{\vdash}} N <:^{-} L \mathbin{/} {}^{(\forall)}W\end{array}}$ Under input Θ, type input R (N) is a subtype of input P (L), if output constraint ${}^{(\forall)}W$ holds

$$\frac{}{\Theta \mathrel{\widetilde{\vdash}} 1 <:^{+} 1 \mathbin{/} \mathsf{tt}}\,\widetilde{<:}^{+}1 \qquad \frac{}{\Theta \mathrel{\widetilde{\vdash}} 0 <:^{+} 0 \mathbin{/} \mathsf{tt}}\,\widetilde{<:}^{+}0$$

$$\frac{\Theta \mathrel{\widetilde{\vdash}} R_1 <:^{+} R_1' \mathbin{/} W_1 \qquad \Theta \mathrel{\widetilde{\vdash}} R_2 <:^{+} R_2' \mathbin{/} W_2}{\Theta \mathrel{\widetilde{\vdash}} R_1 \times R_2 <:^{+} R_1' \times R_2' \mathbin{/} W_1 \wedge W_2}\,\widetilde{<:}^{+}\times$$

$$\frac{P_1 = \exists^{\mathsf{d}} \Xi_1.\, R_1 \wedge \overrightarrow{\varphi_1} \qquad P_2 = \exists^{\mathsf{d}} \Xi_2.\, R_2 \wedge \overrightarrow{\varphi_2}}{\Theta \mathrel{\widetilde{\vdash}} P_1 + P_2 <:^{+} P_1' + P_2' \mathbin{/} (\forall^{\mathsf{d}} \Xi_1.\, \overrightarrow{\varphi_1} \supset \underline{R_1 <:^{+} P_1'}) \wedge (\forall^{\mathsf{d}} \Xi_2.\, \overrightarrow{\varphi_2} \supset \underline{R_2 <:^{+} P_2'})}\,\widetilde{<:}^{+}+$$

$$\frac{\Theta \mathrel{\widetilde{\vdash}} R <:^{+} R' \mathbin{/} W}{\Theta \mathrel{\widetilde{\vdash}} R <:^{+} R' \wedge \overrightarrow{\varphi} \mathbin{/} W \wedge \overrightarrow{\varphi}}\,\widetilde{<:}^{+}\wedge\mathrm{R} \qquad \frac{\overset{\mathsf{d}}{\div}\Theta \vdash \sigma : {}^{\mathsf{d}}\Xi \qquad \Theta \mathrel{\widetilde{\vdash}} R <:^{+} [\sigma]Q \mathbin{/} W}{\Theta \mathrel{\widetilde{\vdash}} R <:^{+} \exists^{\mathsf{d}} \Xi.\, Q \mathbin{/} W}\,\widetilde{<:}^{+}\exists\mathrm{R}$$

$$\frac{\Theta \mathrel{\widetilde{\vdash}} \mathscr{M}'(F') \geq \mathscr{M}(F) \mathbin{/} W}{\Theta \mathrel{\widetilde{\vdash}} \left\{\nu : \mu F' \;\middle|\; \mathscr{M}'(F')\right\} <:^{+} \{\nu : \mu F \mid \mathscr{M}(F)\} \mathbin{/} W}\,\widetilde{<:}^{+}\mu$$

$$\frac{}{\Theta \mathrel{\widetilde{\vdash}} {\downarrow}N <:^{+} {\downarrow}(\forall^{\mathsf{d}} \Xi.\, \overrightarrow{\varphi} \supset L) \mathbin{/} \forall^{\mathsf{d}} \Xi.\, \overrightarrow{\varphi} \supset \underline{N <:^{-} L}}\,\widetilde{<:}^{+}{\downarrow}$$

$$\frac{}{\Theta \mathrel{\widetilde{\vdash}} {\uparrow}(\exists^{\mathsf{d}} \Xi.\, R \wedge \overrightarrow{\varphi}) <:^{-} {\uparrow}P \mathbin{/} \forall^{\mathsf{d}} \Xi.\, \overrightarrow{\varphi} \supset \underline{R <:^{+} P}}\,\widetilde{<:}^{-}{\uparrow}$$

$$\frac{\Theta \mathrel{\widetilde{\vdash}} L' <:^{-} L \mathbin{/} W}{\Theta \mathrel{\widetilde{\vdash}} \overrightarrow{\varphi} \supset L' <:^{-} L \mathbin{/} W \wedge \overrightarrow{\varphi}}\,\widetilde{<:}^{-}\supset\mathrm{L} \qquad \frac{\overset{\mathsf{d}}{\div}\Theta \vdash \sigma : {}^{\mathsf{d}}\Xi \qquad \Theta \mathrel{\widetilde{\vdash}} [\sigma]M <:^{-} L \mathbin{/} W}{\Theta \mathrel{\widetilde{\vdash}} \forall^{\mathsf{d}} \Xi.\, M <:^{-} L \mathbin{/} W}\,\widetilde{<:}^{-}\forall\mathrm{L}$$

$$\frac{\Theta \mathrel{\widetilde{\vdash}} R' <:^{+} R \mathbin{/} W_1 \qquad \Theta \mathrel{\widetilde{\vdash}} L <:^{-} L' \mathbin{/} W_2}{\Theta \mathrel{\widetilde{\vdash}} R \to L <:^{-} R' \to L' \mathbin{/} W_1 \wedge W_2}\,\widetilde{<:}^{-}\to$$

Figure A.74: Semideclarative subtyping

$\boxed{\Xi \mathrel{\widetilde{\vdash}} \alpha; F <:_\tau \beta; G}$ Under Ξ, algebra $\alpha : F(\tau) \Rightarrow \tau$ is semideclaratively a submeasure of $\beta : G(\tau) \Rightarrow \tau$ (All inputs)

$$\frac{\begin{array}{lll}\alpha \circ \mathsf{inj}_1 \mathrel{\hat{=}} \alpha_1 & \beta \circ \mathsf{inj}_1 \mathrel{\hat{=}} \beta_1 & \Xi \mathrel{\widetilde{\vdash}} \alpha_1; F_1 <:_\tau \beta_1; G_1 \\ \alpha \circ \mathsf{inj}_2 \mathrel{\hat{=}} \alpha_2 & \beta \circ \mathsf{inj}_2 \mathrel{\hat{=}} \beta_2 & \Xi \mathrel{\widetilde{\vdash}} \alpha_2; F_2 <:_\tau \beta_2; G_2\end{array}}{\Xi \mathrel{\widetilde{\vdash}} \alpha; F_1 \oplus F_2 <:_\tau \beta; G_1 \oplus G_2} \mathrel{\widetilde{}}{<:_\tau}\oplus$$

$$\frac{\Xi, {}^{\mathsf{d}}\Xi' \mathrel{\widetilde{\vdash}} (\top, q) \Rightarrow t; \underline{Q} \otimes \hat{P} <:_\tau (o', q') \Rightarrow t'; \underline{P} \otimes \hat{P}'}{\Xi \mathrel{\widetilde{\vdash}} (\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q) \Rightarrow t; \underline{\exists^{\mathsf{d}}\Xi'.\, Q} \otimes \hat{P} <:_\tau (o', q') \Rightarrow t'; \underline{P} \otimes \hat{P}'} \mathrel{\widetilde{}}{<:_\tau}\exists\mathrm{L}$$

$$\frac{\overset{\mathsf{d}}{\div}\Xi \vdash \sigma : {}^{\mathsf{d}}\Xi' \qquad \Xi \mathrel{\widetilde{\vdash}} (\top, q) \Rightarrow t; \underline{Q} \otimes \hat{P} <:_\tau (\top, q') \Rightarrow [\sigma]t'; \underline{[\sigma]Q'} \otimes \hat{P}'}{\Xi \mathrel{\widetilde{\vdash}} (\top, q) \Rightarrow t; \underline{Q} \otimes \hat{P} <:_\tau (\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q') \Rightarrow t'; \underline{\exists^{\mathsf{d}}\Xi'.\, Q'} \otimes \hat{P}'} \mathrel{\widetilde{}}{<:_\tau}\exists\mathrm{R}$$

($\overrightarrow{\varphi}$ may be $\cdot$)

$$\frac{\Xi, \overrightarrow{\varphi} \mathrel{\widetilde{\vdash}} R <:^{+} Q' \mathbin{/} {}^{(\forall)}W \qquad \Xi, \overrightarrow{\varphi} \mathrel{\widetilde{\models}} {}^{(\forall)}W \qquad \Xi \mathrel{\widetilde{\vdash}} q \Rightarrow t; \hat{P} <:_\tau q' \Rightarrow t'; \hat{P}'}{\Xi \mathrel{\widetilde{\vdash}} (\top, q) \Rightarrow t; \underline{R \wedge \overrightarrow{\varphi}} \otimes \hat{P} <:_\tau (\top, q') \Rightarrow t'; \underline{Q'} \otimes \hat{P}'} \mathrel{\widetilde{}}{<:_\tau}\mathrm{Const}$$

$$\frac{\Xi, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id} \mathrel{\widetilde{\vdash}} q \Rightarrow t; \hat{I} <:_\tau q' \Rightarrow t'; \hat{I}}{\Xi \mathrel{\widetilde{\vdash}} (a, q) \Rightarrow t; \mathrm{Id} \otimes \hat{I} <:_\tau (a, q') \Rightarrow t'; \mathrm{Id} \otimes \hat{I}} \mathrel{\widetilde{}}{<:_\tau}\mathrm{Id} \qquad \frac{\overset{\mathsf{d}}{\div}\Xi \vdash u \equiv t : \tau}{\Xi \mathrel{\widetilde{\vdash}} () \Rightarrow u; I <:_\tau () \Rightarrow t; I} \mathrel{\widetilde{}}{<:_\tau}I$$

Figure A.75: Semideclarative submeasuring

$\boxed{\Theta;\Gamma \mathrel{\widetilde{\vdash}} h \Rightarrow P}$ Under inputs Θ and Γ, input head h semideclaratively infers (output) type P

$$\frac{(x : R) \in \Gamma}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} x \Rightarrow R}\ {}^{\sim}\!\Rightarrow\text{Var}$$

$$\frac{\overline{\Theta} \vdash P\ \mathsf{type}[\xi] \qquad \Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow P \mathbin{/} \chi \qquad \Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} (v : P) \Rightarrow P}\ {}^{\sim}\!\Rightarrow\text{ValAnnot}$$

$\boxed{\Theta;\Gamma \mathrel{\widetilde{\vdash}} g \Rightarrow \uparrow P}$ Under input Θ and Γ, input bound expression g semideclaratively synthesizes type $\uparrow P$ (output)

$$\frac{\Theta;\Gamma \mathrel{\widetilde{\vdash}} h \Rightarrow \downarrow N \qquad \Theta;\Gamma;[N] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \mathbin{/} \chi \qquad \Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} h(s) \Rightarrow \uparrow P}\ {}^{\sim}\!\Rightarrow\text{App}$$

$$\frac{\overline{\Theta} \vdash P\ \mathsf{type}[\xi] \qquad \Theta;\Gamma \mathrel{\widetilde{\vdash}} e \Leftarrow \uparrow P}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} (e : \uparrow P) \Rightarrow \uparrow P}\ {}^{\sim}\!\Rightarrow\text{ExpAnnot}$$

Figure A.76: Semideclarative head and bound expression type synthesis

$\boxed{\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow P \mathbin{/} \chi}$ Under inputs Θ and Γ, input value v semideclaratively checks against type P, with output constraints χ

$$\frac{(x : R') \in \Gamma \qquad \Theta \mathrel{\widetilde{\vdash}} R' <:^{+} R \mathbin{/} W}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} x \Leftarrow R \mathbin{/} W}\,\widetilde{}\Leftarrow\text{Var} \qquad \frac{}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} \langle\rangle \Leftarrow 1 \mathbin{/} \cdot}\,\widetilde{}\Leftarrow 1$$

$$\frac{\Theta;\Gamma \mathrel{\widetilde{\vdash}} v_1 \Leftarrow R_1 \mathbin{/} \chi_1 \qquad \Theta;\Gamma \mathrel{\widetilde{\vdash}} v_2 \Leftarrow R_2 \mathbin{/} \chi_2}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} \langle v_1, v_2\rangle \Leftarrow R_1 \times R_2 \mathbin{/} \chi_1, \chi_2}\,\widetilde{}\Leftarrow\times$$

$$\frac{\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow P_k \mathbin{/} \chi}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} \mathrm{inj}_k\, v \Leftarrow P_1 + P_2 \mathbin{/} \chi}\,\widetilde{}\Leftarrow +_k$$

$$\frac{\nexists x.\, v = \overrightarrow{\mathrm{inj}_{k_i}}^{\,i}\left(\overrightarrow{\langle __j, - \rangle}^{\,j} x\right) \qquad \mathscr{M}(F) \rightsquigarrow \overrightarrow{\alpha}; \overrightarrow{\tau} \qquad \overset{\mathsf{d}}{\div}\Theta \vdash \wr \overrightarrow{\alpha}; F; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta; R \qquad \Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow \exists^{\mathsf{d}}\Theta.\, R \wedge {}^{\mathsf{d}}\Theta \mathbin{/} \chi}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} \mathrm{into}(v) \Leftarrow \{\nu : \mu F \mid \mathscr{M}(F)\} \mathbin{/} \chi}\,\widetilde{}\Leftarrow\mu$$

$$\frac{}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} \{e\} \Leftarrow \downarrow N \mathbin{/} (e \Leftarrow N)}\,\widetilde{}\Leftarrow\downarrow \qquad \frac{\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow R \mathbin{/} \chi}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow R \wedge \overrightarrow{\varphi} \mathbin{/} \overrightarrow{\varphi}, \chi}\,\widetilde{}\Leftarrow\wedge$$

$$\frac{\overset{\mathsf{d}}{\div}\Theta \vdash \sigma : {}^{\mathsf{d}}\Xi \qquad \Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow [\sigma]Q \mathbin{/} \chi}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow (\exists^{\mathsf{d}}\Xi.\, Q) \mathbin{/} \chi}\,\widetilde{}\Leftarrow\exists$$

Figure A.77: Semideclarative value type checking

$\boxed{\Theta;\Gamma \mathrel{\widetilde{\vdash}} e \Leftarrow N}$ Under inputs Θ and Γ, input expression e semideclaratively checks against input type N

$$\frac{\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow P \mid \chi \qquad \Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} \mathsf{return}\, v \Leftarrow \uparrow P}\,\widetilde{\Leftarrow}\uparrow$$

$$\frac{\Theta;\Gamma \mathrel{\widetilde{\vdash}} g \Rightarrow \uparrow(\exists^{\mathsf{d}}\Xi.R \wedge \overrightarrow{\psi}) \qquad \Theta,{}^{\mathsf{d}}\Xi,\overrightarrow{\psi};\Gamma,x:R \mathrel{\widetilde{\vdash}} e \Leftarrow L}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} \mathsf{let}\, x = g;\, e \Leftarrow L}\,\widetilde{\Leftarrow}\mathsf{let}$$

$$\frac{\Theta;\Gamma \mathrel{\widetilde{\vdash}} h \Rightarrow P \qquad \Theta;\Gamma;[P] \mathrel{\widetilde{\vdash}} \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow L}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} \mathsf{match}\, h\, \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow L}\,\widetilde{\Leftarrow}\mathsf{match}$$

$$\frac{\Theta;\Gamma,x:R \mathrel{\widetilde{\vdash}} e \Leftarrow L}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} \lambda x.e \Leftarrow R \to L}\,\widetilde{\Leftarrow}\lambda \qquad \frac{\Theta \vdash \mathsf{ff}\ \mathsf{true}}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} \mathsf{unreachable} \Leftarrow L}\,\widetilde{\Leftarrow}\mathsf{Unreachable}$$

$$\frac{\Theta \mathrel{\widetilde{\vdash}} \forall a \overset{\mathsf{d}}{\div} \mathbb{N},{}^{\mathsf{d}}\Xi.M <:^{-} L \mid W \qquad \Theta \mathrel{\widetilde{\models}} W}{\Theta,a \div \mathbb{N};\Gamma,x:{\downarrow}\forall a' \overset{\mathsf{d}}{\div} \mathbb{N},{}^{\mathsf{d}}\Xi.a' < a \supset [a'/a]M \mathrel{\widetilde{\vdash}} e \Leftarrow \forall^{\mathsf{d}}\Xi.M}\,\widetilde{\Leftarrow}\mathsf{rec}$$
$$\Theta;\Gamma \mathrel{\widetilde{\vdash}} \mathsf{rec}\, x : (\forall a \overset{\mathsf{d}}{\div} \mathbb{N},{}^{\mathsf{d}}\Xi.M).\, e \Leftarrow L$$

$$\frac{\Theta,{}^{\mathsf{d}}\Xi;\Gamma \mathrel{\widetilde{\vdash}} e \Leftarrow M}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} e \Leftarrow \forall^{\mathsf{d}}\Xi.M}\,\widetilde{\Leftarrow}\forall \qquad \frac{\Theta,\overrightarrow{\varphi};\Gamma \mathrel{\widetilde{\vdash}} e \Leftarrow L}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} e \Leftarrow \overrightarrow{\varphi} \supset L}\,\widetilde{\Leftarrow}\supset$$

Figure A.78: Semideclarative expression type checking

$\boxed{\Theta;\Gamma;[P] \mathrel{\widetilde{\vdash}} \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}$ Under Θ and Γ, patterns r_i match semideclaratively against type P and branch expressions e_i check against type N (all inputs)

$$\frac{\Theta, {}^{\mathsf{d}}\Xi;\Gamma;[Q] \mathrel{\widetilde{\vdash}} \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}{\Theta;\Gamma;[\exists^{\mathsf{d}}\Xi.\,Q] \mathrel{\widetilde{\vdash}} \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}\ \widetilde{}\text{Match}\exists$$

$$\frac{\Theta, \overrightarrow{\varphi};\Gamma;[R] \mathrel{\widetilde{\vdash}} \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}{\Theta;\Gamma;[R \wedge \overrightarrow{\varphi}] \mathrel{\widetilde{\vdash}} \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}\ \widetilde{}\text{Match}\wedge \qquad \frac{\Theta;\Gamma \mathrel{\widetilde{\vdash}} e \Leftarrow N}{\Theta;\Gamma;[1] \mathrel{\widetilde{\vdash}} \{\langle\rangle \Rightarrow e\} \Leftarrow N}\ \widetilde{}\text{Match1}$$

$$\frac{\Theta;\Gamma,x_1 : R_1,x_2 : R_2 \mathrel{\widetilde{\vdash}} e \Leftarrow N}{\Theta;\Gamma;[R_1 \times R_2] \mathrel{\widetilde{\vdash}} \{\langle x_1,x_2\rangle \Rightarrow e\} \Leftarrow N}\ \widetilde{}\text{Match}\times$$

$$\frac{\Theta, {}^{\mathsf{d}}\Xi_1, \overrightarrow{\psi}_1;\Gamma,x_1 : R_1 \mathrel{\widetilde{\vdash}} e_1 \Leftarrow N \qquad \Theta, {}^{\mathsf{d}}\Xi_2, \overrightarrow{\psi}_2;\Gamma,x_2 : R_2 \mathrel{\widetilde{\vdash}} e_2 \Leftarrow N}{\Theta;\Gamma;[(\exists^{\mathsf{d}}\Xi_1.\,R_1 \wedge \overrightarrow{\psi}_1) + (\exists^{\mathsf{d}}\Xi_2.\,R_2 \wedge \overrightarrow{\psi}_2)] \mathrel{\widetilde{\vdash}} \{\mathtt{inj}_1\, x_1 \Rightarrow e_1 \mid \mathtt{inj}_2\, x_2 \Rightarrow e_2\} \Leftarrow N}\ \widetilde{}\text{Match}+$$

$$\frac{}{\Theta;\Gamma;[0] \mathrel{\widetilde{\vdash}} \{\} \Leftarrow N}\ \widetilde{}\text{Match0}$$

$$\frac{\mathscr{M}(F) \rightsquigarrow \overrightarrow{\alpha};\overrightarrow{\tau} \qquad \overset{\mathsf{d}}{\div}\Theta \vdash \wr \overrightarrow{\alpha};F;\mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta;R \qquad \Theta, {}^{\mathsf{d}}\Theta;\Gamma,x : R \mathrel{\widetilde{\vdash}} e \Leftarrow N}{\Theta;\Gamma;[\{\nu : \mu F \mid \mathscr{M}(F)\}] \mathrel{\widetilde{\vdash}} \{\mathtt{into}(x) \Rightarrow e\} \Leftarrow N}\ \widetilde{}\text{Match}\mu$$

$\boxed{\Theta;\Gamma;[N] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \mathbin{/} \chi}$ Under inputs Θ and Γ, if spine s (input) is applied to a head of type $\downarrow N$ (input: N), it semideclaratively returns a result of type $\uparrow P$ (output), if output constraints χ hold semideclaratively

$$\frac{\overset{\mathsf{d}}{\div}\Theta \vdash \sigma : {}^{\mathsf{d}}\Xi \qquad \Theta;\Gamma;[[\sigma]M] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \mathbin{/} \chi}{\Theta;\Gamma;[\forall^{\mathsf{d}}\Xi.\,M] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \mathbin{/} \chi}\ \widetilde{}\text{Spine}\forall$$

$$\frac{\Theta;\Gamma;[L] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \mathbin{/} \chi}{\Theta;\Gamma;[\overrightarrow{\varphi} \supset L] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \mathbin{/} \overrightarrow{\varphi},\chi}\ \widetilde{}\text{Spine}\supset$$

$$\frac{\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow R \mathbin{/} \chi \qquad \Theta;\Gamma;[L] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \mathbin{/} \chi'}{\Theta;\Gamma;[R \to L] \mathrel{\widetilde{\vdash}} v,s \Rightarrow \uparrow P \mathbin{/} \chi,\chi'}\ \widetilde{}\text{SpineApp}$$

$$\frac{}{\Theta;\Gamma;[\uparrow P] \mathrel{\widetilde{\vdash}} \cdot \Rightarrow \uparrow P \mathbin{/} \mathsf{tt}}\ \widetilde{}\text{SpineNil}$$

Figure A.79: Semideclarative pattern matching and spine typing

$\boxed{\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W}$ Under input Θ, input constraint ${}^{(\forall)}W$ semideclaratively holds

$$\frac{\Theta \vdash \varphi\ \mathsf{true}}{\Theta \mathrel{\widetilde{\models}} \varphi}\ \widetilde{\models}\mathrm{Prp} \qquad \frac{\overset{\mathsf{d}}{\cdot}\Theta \vdash u \equiv t : \tau}{\Theta \mathrel{\widetilde{\models}} u \equiv_\tau t}\ \widetilde{\models}\mathrm{IxEq} \qquad \frac{\overset{\mathsf{d}}{\cdot}\Theta; [\tau] \vdash \boldsymbol{t} \equiv \boldsymbol{t}' : \kappa}{\Theta \mathrel{\widetilde{\models}} \boldsymbol{t} \equiv_{[\tau]} \boldsymbol{t}'}\ \widetilde{\models}\boldsymbol{t}\mathrm{Eq}$$

$$\frac{\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W_1 \qquad \Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W_2}{\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W_1 \wedge {}^{(\forall)}W_2}\ \widetilde{\models}\wedge \qquad \frac{\Theta \mathrel{\widetilde{\models}} W_k \text{ for some } W_k \in \overrightarrow{W}}{\Theta \mathrel{\widetilde{\models}} \bigvee \overrightarrow{W}}\ \widetilde{\models}\vee$$

$$\frac{\Theta, \varphi \mathrel{\widetilde{\models}} {}^{(\supset)}W}{\Theta \mathrel{\widetilde{\models}} \varphi \supset {}^{(\supset)}W}\ \widetilde{\models}\supset \qquad \frac{\Theta, a \overset{\mathsf{d}}{\div} \tau \mathrel{\widetilde{\models}} {}^{(\forall)}W}{\Theta \mathrel{\widetilde{\models}} \forall a \overset{\mathsf{d}}{\div} \tau.\, {}^{(\forall)}W}\ \widetilde{\models}\forall$$

$$\frac{\Theta \mathrel{\widetilde{\vdash}} A <:^{\pm} B \mathbin{/} {}^{(\forall)}W \qquad \Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W}{\Theta \mathrel{\widetilde{\models}} \underline{A <:^{\pm} B}}\ \widetilde{\models}<:^{\pm}$$

$\boxed{\Theta; \Gamma \mathrel{\widetilde{\lhd}} \chi}$ Under inputs Θ and Γ, input constraints χ semideclaratively hold

$$\frac{}{\Theta; \Gamma \mathrel{\widetilde{\lhd}} \cdot}\ \widetilde{\lhd}\mathrm{Empty} \qquad \frac{\Theta \mathrel{\widetilde{\models}} W \qquad \Theta; \Gamma \mathrel{\widetilde{\lhd}} \chi}{\Theta; \Gamma \mathrel{\widetilde{\lhd}} W, \chi}\ \widetilde{\lhd}W$$

$$\frac{\Theta; \Gamma \mathrel{\widetilde{\vdash}} e \Leftarrow N \qquad \Theta; \Gamma \mathrel{\widetilde{\lhd}} \chi}{\Theta; \Gamma \mathrel{\widetilde{\lhd}} (e \Leftarrow N), \chi}\ \widetilde{\lhd}\mathrm{NegChk}$$

Figure A.80: Semideclarative constraint verification

$\boxed{\Theta \vdash A \equiv^{\pm} B}$ Under input Θ, input types A and B are equivalent

$$\frac{}{\Theta \vdash 1 \equiv^{+} 1}\ \mathrm{Tp}{\equiv}^{+}1 \qquad \frac{}{\Theta \vdash 0 \equiv^{+} 0}\ \mathrm{Tp}{\equiv}^{+}0$$

$$\frac{\Theta \vdash R_1 \equiv^{+} R_1' \qquad \Theta \vdash R_2 \equiv^{+} R_2'}{\Theta \vdash R_1 \times R_2 \equiv^{+} R_1' \times R_2'}\ \mathrm{Tp}{\equiv}^{+}\times \qquad \frac{\Theta \vdash P_1 \equiv^{+} P_1' \qquad \Theta \vdash P_2 \equiv^{+} P_2'}{\Theta \vdash P_1 + P_2 \equiv^{+} P_1' + P_2'}\ \mathrm{Tp}{\equiv}^{+}+$$

$$\frac{\Theta \vdash R \equiv^{+} R' \qquad \Theta \vdash \overrightarrow{\varphi} \equiv \overrightarrow{\varphi'} : \mathbb{B}}{\Theta \vdash R \wedge \overrightarrow{\varphi} \equiv^{+} R' \wedge \overrightarrow{\varphi'}}\ \mathrm{Tp}{\equiv}^{+}\wedge \qquad \frac{\Theta, {}^{\mathsf{d}}\Xi \vdash Q \equiv^{+} Q'}{\Theta \vdash \exists^{\mathsf{d}}\Xi.\, Q \equiv^{+} \exists^{\mathsf{d}}\Xi.\, Q'}\ \mathrm{Tp}{\equiv}^{+}\exists$$

$$\frac{\Theta \vdash \mathscr{M}_1(F) \equiv \mathscr{M}_2(F)}{\Theta \vdash \{\nu : \mu F \mid \mathscr{M}_1(F)\} \equiv^{+} \{\nu : \mu F \mid \mathscr{M}_2(F)\}}\ \mathrm{Tp}{\equiv}^{+}\mu$$

$$\frac{\Theta \vdash N \equiv^{-} N'}{\Theta \vdash \downarrow N \equiv^{+} \downarrow N'}\ \mathrm{Tp}{\equiv}^{+}\downarrow \qquad \frac{\Theta \vdash P \equiv^{+} P'}{\Theta \vdash \uparrow P \equiv^{-} \uparrow P'}\ \mathrm{Tp}{\equiv}^{-}\uparrow$$

$$\frac{\Theta \vdash L \equiv^{-} L' \qquad \Theta \vdash \overrightarrow{\varphi} \equiv \overrightarrow{\varphi'} : \mathbb{B}}{\Theta \vdash \overrightarrow{\varphi} \supset L \equiv^{-} \overrightarrow{\varphi'} \supset L'}\ \mathrm{Tp}{\equiv}^{-}\supset \qquad \frac{\Theta, {}^{\mathsf{d}}\Xi \vdash M \equiv^{-} M'}{\Theta \vdash \forall^{\mathsf{d}}\Xi.\, M \equiv^{-} \forall^{\mathsf{d}}\Xi.\, M'}\ \mathrm{Tp}{\equiv}^{-}\forall$$

$$\frac{\Theta \vdash R \equiv^{+} R' \qquad \Theta \vdash L \equiv^{-} L'}{\Theta \vdash R \to L \equiv^{-} R' \to L'}\ \mathrm{Tp}{\equiv}^{-}\to$$

$\boxed{\Theta \vdash \mathscr{M}'(F) \equiv \mathscr{M}(F)}$ Under input Θ, input measurement lists $\mathscr{M}'(F)$ and $\mathscr{M}(F)$ are equivalent

$$\frac{}{\Theta \vdash \cdot_F \equiv \cdot_F} \qquad \frac{\Theta \vdash \mathscr{M}'(F) \equiv \mathscr{M}(F) \qquad {}^{\mathsf{d}}\div\Theta; [\tau] \vdash \boldsymbol{t} \equiv \boldsymbol{u} : \kappa \qquad {}^{\mathsf{d}}\div\Theta \vdash t = u\ \mathsf{true}}{\Theta \vdash \mathscr{M}'(F), (\mathsf{fold}_F\ \alpha)\, \nu\, \boldsymbol{t} =_\tau t \equiv \mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\, \nu\, \boldsymbol{u} =_\tau u}$$

Figure A.81: Declarative equivalence of types (and measurements)

$\boxed{\Xi \vdash \alpha; F \equiv_\tau \beta; G}$ Under Ξ, measures $\alpha : F(\tau) \Rightarrow \tau$ and $\beta : G(\tau) \Rightarrow \tau$ are equivalent

$$\frac{\begin{array}{lll}\alpha \circ \mathsf{inj}_1 \mathrel{\mathring{=}} \alpha_1 & \beta \circ \mathsf{inj}_1 \mathrel{\mathring{=}} \beta_1 & \Xi \vdash \alpha_1; F_1 \equiv_\tau \beta_1; G_1 \\ \alpha \circ \mathsf{inj}_2 \mathrel{\mathring{=}} \alpha_2 & \beta \circ \mathsf{inj}_2 \mathrel{\mathring{=}} \beta_2 & \Xi \vdash \alpha_2; F_2 \equiv_\tau \beta_2; G_2\end{array}}{\Xi \vdash \alpha; F_1 \oplus F_2 \equiv_\tau \beta; G_1 \oplus G_2}\ \text{Meas}{\equiv}{\oplus}$$

$$\frac{\Xi \vdash P \equiv^+ P' \qquad \Xi \vdash q \Rightarrow t; \hat{P} \equiv_\tau q' \Rightarrow t'; \hat{P}'}{\Xi \vdash (\top, q) \Rightarrow t; \underline{P} \otimes \hat{P} \equiv_\tau (\top, q') \Rightarrow t'; \underline{P'} \otimes \hat{P}'}\ \text{Meas}{\equiv}\text{Const}$$

$$\frac{\Xi, a \mathrel{\dot{\div}} \tau, a\,\mathrm{Id} \vdash q \Rightarrow t; \hat{I} \equiv_\tau q' \Rightarrow t'; \hat{I}'}{\Xi \vdash (a, q) \Rightarrow t; \mathrm{Id} \otimes \hat{I} \equiv_\tau (a, q') \Rightarrow t'; \mathrm{Id} \otimes \hat{I}'}\ \text{Meas}{\equiv}\text{Id}$$

$$\frac{\mathrel{\dot{\div}}\Xi \vdash u \equiv t : \tau}{\Xi \vdash () \Rightarrow u; I \equiv_\tau () \Rightarrow t; I}\ \text{Meas}{\equiv}I$$

Figure A.82: Declarative measure equivalence

$\boxed{\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W \leftrightarrow {}^{(\forall)}W'}$ Under Θ (input), constraints ${}^{(\forall)}W$ and ${}^{(\forall)}W'$ (inputs) are equivalent

$$\frac{\Theta \vdash \varphi \equiv \psi : \mathbb{B}}{\Theta \mathrel{\widetilde{\models}} \varphi \leftrightarrow \psi}\ \mathrel{\widetilde{\models}}{\leftrightarrow}\text{Prp} \qquad \frac{\mathrel{\dot{\div}}\Theta \vdash u \equiv t : \tau \qquad \mathrel{\dot{\div}}\Theta \vdash u' \equiv t' : \tau}{\Theta \mathrel{\widetilde{\models}} u \equiv_\tau u' \leftrightarrow t \equiv_\tau t'}\ \mathrel{\widetilde{\models}}{\leftrightarrow}\text{IxEq}$$

$$\frac{\mathrel{\dot{\div}}\Theta; [\tau] \vdash \boldsymbol{t}_1 \equiv \boldsymbol{t}_1' : \kappa \qquad \mathrel{\dot{\div}}\Theta; [\tau] \vdash \boldsymbol{t}_2 \equiv \boldsymbol{t}_2' : \kappa}{\Theta \mathrel{\widetilde{\models}} \boldsymbol{t}_1 \equiv_{[\tau]} \boldsymbol{t}_2 \leftrightarrow \boldsymbol{t}_1' \equiv_{[\tau]} \boldsymbol{t}_2'}\ \mathrel{\widetilde{\models}}{\leftrightarrow}\boldsymbol{t}\text{Eq}$$

$$\frac{\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W_1 \leftrightarrow {}^{(\forall)}W_1' \qquad \Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W_2 \leftrightarrow {}^{(\forall)}W_2'}{\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W_1 \wedge {}^{(\forall)}W_2 \leftrightarrow {}^{(\forall)}W_1' \wedge {}^{(\forall)}W_2'}\ \mathrel{\widetilde{\models}}{\leftrightarrow}\wedge$$

$$\frac{\Theta \mathrel{\widetilde{\models}} W_k \leftrightarrow W_k' \text{ for all } (W_k, W_k') \in zip(\overrightarrow{W})(\overrightarrow{W'})}{\Theta \mathrel{\widetilde{\models}} \bigvee \overrightarrow{W} \leftrightarrow \bigvee \overrightarrow{W'}}\ \mathrel{\widetilde{\models}}{\leftrightarrow}\vee$$

$$\frac{\Theta \vdash \varphi \equiv \varphi' : \mathbb{B} \qquad \Theta, \varphi \mathrel{\widetilde{\models}} {}^{(\supset)}W \leftrightarrow {}^{(\supset)}W'}{\Theta \mathrel{\widetilde{\models}} \varphi \supset {}^{(\supset)}W \leftrightarrow \varphi' \supset {}^{(\supset)}W'}\ \mathrel{\widetilde{\models}}{\leftrightarrow}\supset$$

$$\frac{\Theta, a \mathrel{\dot{\div}} \tau \mathrel{\widetilde{\models}} {}^{(\forall)}W \leftrightarrow {}^{(\forall)}W'}{\Theta \mathrel{\widetilde{\models}} \forall a \mathrel{\dot{\div}} \tau.\, {}^{(\forall)}W \leftrightarrow \forall a \mathrel{\dot{\div}} \tau.\, {}^{(\forall)}W'}\ \mathrel{\widetilde{\models}}{\leftrightarrow}\forall \qquad \frac{\Theta \vdash A \equiv^\pm A' \qquad \Theta \vdash B \equiv^\pm B'}{\Theta \mathrel{\widetilde{\models}} \underline{A <:^\pm B} \leftrightarrow \underline{A' <:^\pm B'}}\ \mathrel{\widetilde{\models}}{\leftrightarrow}{<:}^\pm$$

Figure A.83: Constraint equivalence

$\boxed{\Theta \mathrel{\widetilde{\lhd}} \chi_1 \leftrightarrow \chi_2}$ Under Θ (input), constraint lists χ_1 and χ_2 (inputs) are equivalent

$$\frac{}{\Theta \mathrel{\widetilde{\lhd}} \cdot \leftrightarrow \cdot}\ \widetilde{\lhd}{\leftrightarrow}\text{Empty} \qquad \frac{\Theta \mathrel{\widetilde{\models}} W_1 \leftrightarrow W_2 \qquad \Theta \mathrel{\widetilde{\lhd}} \chi_1 \leftrightarrow \chi_2}{\Theta \mathrel{\widetilde{\lhd}} W_1, \chi_1 \leftrightarrow W_2, \chi_2}\ \widetilde{\lhd}{\leftrightarrow}W$$

$$\frac{\Theta \vdash N_1 \equiv^{-} N_2 \qquad \Theta \mathrel{\widetilde{\lhd}} \chi_1 \leftrightarrow \chi_2}{\Theta \mathrel{\widetilde{\lhd}} (e \Leftarrow N_1), \chi_1 \leftrightarrow (e \Leftarrow N_2), \chi_2}\ \widetilde{\lhd}{\leftrightarrow}{\Leftarrow}-$$

Figure A.84: Equivalence of constraint lists

$\boxed{\begin{array}{l}\hat{\Theta};\cdot \mathrel{\widetilde{\vdash}} W\ \mathsf{fixInstChk} \dashv \Omega \\ \hat{\Theta};\Gamma \mathrel{\widetilde{\vdash}} \chi\ \mathsf{fixInstChk} \dashv \Omega\end{array}}$ Under input(s) $\hat{\Theta}$ (and Γ) the input constraints W (χ) semideclaratively hold at output solutions Ω

As in Figure A.61 but replacing $\|\hat{\Theta}\| \models [\Omega]^2 W$ by $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} [\Omega]^2 W$
and replacing $\|\hat{\Theta}\|;\Gamma \lhd [\Omega]^2 \chi$ by $\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\lhd}} [\Omega]^2 \chi$.

Figure A.85: Semideclarative fixInst

Appendix A.9 Miscellaneous Definitions

$\xi - a$, $\xi - \mathfrak{D}$, $\xi - \Xi$	Fig. A.5
$- \wedge \Theta$ and $- \wedge \overrightarrow{\varphi}$	Fig. A.14
$\exists \Theta. -$	Fig. A.14
$\Theta \supset -$ and $\overrightarrow{\varphi} \supset -$	Fig. A.14
$\forall \Theta. -$	Fig. A.14
$\overset{\mathsf{d}}{\dot{-}} -$	Fig. A.4
$\langle - \mid - \rangle$	Fig. A.6
$[\sigma]-, [\sigma]^{\mathsf{h}} -$	Fig. A.6, Fig. A.7
$\mathscr{K}_\kappa$	Fig. A.10
$hgt(-)$	Def. C.1 (Height and Structure)
$[\![\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma]\!]$	Def. C.2 (Denotation of Syntactic Substitution)
$^{\mathsf{d}}[\sigma] -$	Def. C.4 (Substitution on ξ)
$\lfloor \delta \rfloor$	Fig. A.46
$\lfloor \sigma \rfloor$	Def. A.4 (Remove Program Entries)
$\lfloor \chi \rfloor$	Fig. A.58
$\delta \restriction_\Theta$	Fig. A.48
$\sigma \restriction_\Theta$	Similar to definition of $\delta \restriction_\Theta$ in Fig. A.48
$\delta_1 \restriction_\xi = \delta_2 \restriction_\xi$	Fig. A.48
$[\![\breve{\Xi}]\!]_\delta^{\mathsf{fix}}$ and $[\![\breve{\Xi}]\!]_\delta$	Fig. A.48
$\overline{\Theta}$	Def. A.2 (Remove Propositions)
$\Theta - \mathrm{Id}$	Def. A.3 (Remove Id Variables)
$id_{\Theta;\Gamma}$ and $(\Theta;\Gamma)/(\Theta;\Gamma)$	Def. C.5 (Id. Substitution)
$\langle \sigma \rangle \Xi$, $\langle \sigma \rangle \mathfrak{D}$	Def. C.3 (FV Image of a Substitution)
$\Theta \vdash \Gamma \leq^+ \Gamma'$	Def. C.7
Predomain (cpo)	Def. D.1
Domain (cppo)	Def. D.2
Continuous function	Def. D.3
Cpo	Def. D.4
Cppo	Def. D.5
fold and $fold^n$	Def. E.1 (Fold)
$\Delta \vdash \xi' \angle \xi$	Fig. G.1

Figure A.86: Miscellaneous definitions

Appendix B

On *cl* and The Equivalence of *cl* and det

Definition B.1 (ξ Closure Operator)**.** Define

$$cl^0(\xi)(\mathfrak{D}) = \mathfrak{D}$$
$$cl^{n+1}(\xi)(\mathfrak{D}) = cl^n(\xi)(\mathfrak{D}) \cup \{b \mid \mathfrak{A} \subseteq cl^n(\xi)(\mathfrak{D}) \text{ for some } \mathfrak{A}\to b \in \xi\}$$

Define

$$cl(\xi)(\mathfrak{D}) = \bigcup_{k\in\mathbb{N}} cl^k(\xi)(\mathfrak{D})$$

Lemma B.1. *If $n \in \mathbb{N}$ and $b \notin cl^n(\xi)(\emptyset)$ and $b \in \mathfrak{B}$ then $cl^n(\xi \cup \mathfrak{B}\to c)(\emptyset) = cl^n(\xi)(\emptyset)$.*

Proof. By induction on n.

- **Case** $n = 0$: $cl^n(\xi \cup \mathfrak{B}\to c)(\emptyset) = cl^0(\xi \cup \mathfrak{B}\to c)(\emptyset) = \emptyset = cl^0(\xi)(\emptyset) = cl^n(\xi)(\emptyset)$.
- **Case** $n = k+1$:

$b \notin cl^n(\xi)(\emptyset)$	Given
$cl^k(\xi)(\emptyset) \subseteq cl^{k+1}(\xi)(\emptyset)$	By def. of cl^{k+1}, $\cup$ property
$= cl^n(\xi)(\emptyset)$	Current case
$b \notin cl^k(\xi)(\emptyset)$	Otherwise $b \in cl^n(\xi)(\emptyset)$

$$
\begin{aligned}
cl^n(\xi \cup \mathfrak{B}{\to}c)(\emptyset) &= cl^{k+1}(\xi \cup \mathfrak{B}{\to}c)(\emptyset) \\
&= cl^k(\xi \cup \mathfrak{B}{\to}c)(\emptyset) \\
&\quad \cup \left\{ d \;\middle|\; \mathfrak{A} \subseteq cl^k(\xi \cup \mathfrak{B}{\to}c)(\emptyset) \text{ for some } \mathfrak{A}{\to}d \in (\xi \cup \mathfrak{B}{\to}c) \right\} && \text{By def.} \\
&= cl^k(\xi)(\emptyset) \\
&\quad \cup \left\{ d \;\middle|\; \mathfrak{A} \subseteq cl^k(\xi)(\emptyset) \text{ for some } \mathfrak{A}{\to}d \in (\xi \cup \mathfrak{B}{\to}c) \right\} && \text{By i.h.} \\
&= cl^k(\xi)(\emptyset) \\
&\quad \cup \left\{ d \;\middle|\; \mathfrak{A} \subseteq cl^k(\xi)(\emptyset) \text{ for some } \mathfrak{A}{\to}d \in \xi \right\} && \text{As } \mathfrak{B} \not\subseteq cl^k(\xi)(\emptyset) \\
&= cl^{k+1}(\xi)(\emptyset) && \text{By def.} \\
&= cl^n(\xi)(\emptyset)
\end{aligned}
$$

□

Lemma B.2. *If* $\mathfrak{B} \subseteq cl^n(\xi \cup \mathfrak{B}{\to}c)(\emptyset)$ *then* $\mathfrak{B} \subseteq cl^n(\xi)(\emptyset)$.

Proof. By contradiction. Suppose there exists $b \in \mathfrak{B}$ such that $b \notin cl^n(\xi)(\emptyset)$. By Lemma B.1, $cl^n(\xi)(\emptyset) = cl^n(\xi \cup \mathfrak{B}{\to}c)(\emptyset)$. Therefore, $b \notin cl^n(\xi \cup \mathfrak{B}{\to}c)(\emptyset)$, contradicting $b \in \mathfrak{B} \subseteq cl^n(\xi \cup \mathfrak{B}{\to}c)(\emptyset)$. □

Lemma B.3. *If* $b \in cl(\xi)(\emptyset)$ *then* $cl(\xi \cup \mathfrak{B}{\to}c)(\emptyset) \subseteq cl(\xi \cup (\mathfrak{B} \cup \{b\}){\to}c)(\emptyset)$.

Proof. It suffices to show (by induction on n) that,

for all $n \in \mathbb{N}$, we have $cl^n(\xi \cup \mathfrak{B}{\to}c)(\emptyset) \subseteq cl(\xi \cup (\mathfrak{B} \cup \{b\}){\to}c)(\emptyset)$.

The $n = 0$ case is easy.

For the $n = k+1$ case, suppose $\mathfrak{D}{\to}a \in (\xi \cup \mathfrak{B}{\to}c)$ and $\mathfrak{D} \subseteq cl^k(\xi \cup \mathfrak{B}{\to}c)(\emptyset)$

and show $a \in cl(\xi \cup (\mathfrak{B} \cup \{b\}){\to}c)(\emptyset)$.

- **Case** $\mathfrak{D}{\to}a = \mathfrak{B}{\to}c$:

$$
\begin{array}{rll}
\mathfrak{B} \subseteq cl^k(\xi \cup \mathfrak{B}{\to}c)(\emptyset) & & \text{As } \mathfrak{D} = \mathfrak{B} \\
\mathfrak{B} \subseteq cl^k(\xi)(\emptyset) & & \text{By Lemma B.2} \\
\subseteq cl(\xi)(\emptyset) & & \\
\{b\} \subseteq cl(\xi)(\emptyset) & & \text{Given} \\
\mathfrak{B} \cup \{b\} \subseteq cl(\xi)(\emptyset) & & \text{Property of } \cup \\
\subseteq cl(\xi \cup (\mathfrak{B} \cup \{b\}){\to}c)(\emptyset) & & cl(-)(\mathscr{O}) \text{ monotone} \\
a = c & & \\
\in cl(\xi \cup (\mathfrak{B} \cup \{b\}){\to}c)(\emptyset) & & \text{By def. of } cl
\end{array}
$$

- **Case** $\mathfrak{D}{\to}a \neq \mathfrak{B}{\to}c$:

$$
\begin{array}{rll}
\mathfrak{D}{\to}a \in (\xi \cup \mathfrak{B}{\to}c) & & \text{Above} \\
\mathfrak{D}{\to}a \in \xi & & \text{As } \mathfrak{D}{\to}a \neq \mathfrak{B}{\to}c \\
\subseteq (\xi \cup (\mathfrak{B} \cup \{b\}){\to}c) & & \\
\mathfrak{D} \subseteq cl^k(\xi \cup \mathfrak{B}{\to}c)(\emptyset) & & \text{Above} \\
\subseteq cl^k(\xi \cup (\mathfrak{B} \cup \{b\}){\to}c)(\emptyset) & & \text{By i.h.} \\
a \in cl^{k+1}(\xi \cup (\mathfrak{B} \cup \{b\}){\to}c)(\emptyset) & & \text{By def.} \\
\subseteq cl(\xi \cup (\mathfrak{B} \cup \{b\}){\to}c)(\emptyset) & & \text{By def., } \cup \text{ property}
\end{array}
$$

□

Lemma B.4 (Equivalence of *cl* and det)**.** $\xi \vdash \mathfrak{D}$ det *if and only if* $\mathfrak{D} \subseteq cl(\xi)(\emptyset)$

Proof. There are two parts.

We first show the "only if" part by structural induction on $\xi \vdash a$ det, assuming $a \in \mathfrak{D}$.

- **Case**

$$\frac{\emptyset{\to}a \in \xi}{\xi \vdash a \text{ det}} \text{ DetUnit}$$

$a \in \{c \mid \mathfrak{B} \subseteq \emptyset \text{ for some } \mathfrak{B} \to c \in \xi\}$ As $\emptyset \to a \in \xi$ and $\emptyset \subseteq \emptyset$

$= \emptyset \cup \{c \mid \mathfrak{B} \subseteq \emptyset \text{ for some } \mathfrak{B} \to c \in \xi\}$

$= cl^0(\xi)(\emptyset) \cup \{c \mid \mathfrak{B} \subseteq cl^0(\xi)(\emptyset) \text{ for some } \mathfrak{B} \to c \in \xi\}$

$= cl^1(\xi)(\emptyset)$

$\subseteq \cup_{j \in \mathbb{N}} cl^j(\xi)(\emptyset)$

$= cl(\xi)(\emptyset)$

- **Case**

$$\frac{\xi_0 \vdash b \text{ det} \qquad \xi_0 \cup \mathfrak{B} \to c \vdash a \text{ det}}{\xi_0, (\mathfrak{B}, b) \to c \vdash a \text{ det}} \text{ DetCut}$$

$b \in cl(\xi_0)(\emptyset)$ By i.h.

$a \in cl(\xi_0 \cup \mathfrak{B} \to c)(\emptyset)$ By i.h.

$\subseteq cl(\xi_0, (\mathfrak{B}, b) \to c)(\emptyset)$ By Lemma B.3

We now show the second part, the "if" part.

It suffices to show that, for all $n \in \mathbb{N}$, we have $\xi \vdash cl^n(\xi)(\emptyset)$ det.

We proceed by induction on n.

- **Case** $n = 0$ holds vacuously.

- **Case** $n = k + 1$:

  By definition,

$$cl^{k+1}(\xi)(\emptyset) = cl^k(\xi)(\emptyset) \cup \left\{ a \;\middle|\; \mathfrak{C} \subseteq cl^k(\xi)(\emptyset) \text{ for some } \mathfrak{C} \to a \in \xi \right\}$$

Suppose $a \in cl^{k+1}(\xi)(\emptyset)$.

Either $a \in cl^k(\xi)(\emptyset)$ or $a \in \left\{ a \;\middle|\; \mathfrak{C} \subseteq cl^k(\xi)(\emptyset) \text{ for some } \mathfrak{C}{\to}a \in \xi \right\}$.

If $a \in cl^k(\xi)(\emptyset)$ then the goal $\xi \vdash a$ det follows from the induction hypothesis.

Assume $a \notin cl^k(\xi)(\emptyset)$. There exists $\mathfrak{C}{\to}a \in \xi$ such that $\mathfrak{C} \subseteq cl^k(\xi)(\emptyset)$.

If $\mathfrak{C} = \emptyset$ then $\xi \vdash a$ det by DetUnit.

Assume $\mathfrak{C} \neq \emptyset$.

$\xi = (\xi - (\mathfrak{C}{\to}a)), \mathfrak{C}{\to}a$	As $\mathfrak{C}{\to}a \in \xi$
$\mathfrak{C} \subseteq cl^{k+1}((\xi - (\mathfrak{C}{\to}a)), \mathfrak{C}{\to}a)(\emptyset)$	By equality
$\mathfrak{C} \subseteq cl^{k+1}(\xi - (\mathfrak{C}{\to}a))(\emptyset)$	By Lemma B.2
$\xi - (\mathfrak{C}{\to}a) \vdash cl^{k+1}(\xi - (\mathfrak{C}{\to}a))(\emptyset)$ det	By i.h.
$\xi - (\mathfrak{C}{\to}a) \vdash \mathfrak{C}$ det	As $\mathfrak{C} \subseteq cl^{k+1}(\xi - (\mathfrak{C}{\to}a))(\emptyset)$
$(\xi - (\mathfrak{C}{\to}a)), \emptyset{\to}a \vdash a$ det	By DetUnit
$(\xi - (\mathfrak{C}{\to}a)), \mathfrak{C}{\to}a \vdash a$ det	By #$\mathfrak{C}$ uses of DetCut
$\xi \vdash a$ det	By equality

□

Lemma B.5 (Weaken cl)**.** *If* $\mathfrak{B} \subseteq cl(\xi)(\mathfrak{A})$ *then* $cl(\xi)(\mathfrak{A}) = cl(\xi)(\mathfrak{A} \cup \mathfrak{B})$.

Proof. Because $cl(\mathscr{O})(-)$ is monotone, $cl(\xi)(\mathfrak{A}) \subseteq cl(\xi)(\mathfrak{A} \cup \mathfrak{B})$.

It is straightforward to show $cl^n(\xi)(\mathfrak{A} \cup \mathfrak{B}) \subseteq cl(\xi)(\mathfrak{A})$ for all $n \in \mathbb{N}$ by induction on n. For $n = 0$ we have $cl^0(\xi)(\mathfrak{A} \cup \mathfrak{B}) = \mathfrak{A} \cup \mathfrak{B} \subseteq cl(\xi)(\mathfrak{A})$. For $n = m + 1$ use the i.h. and unpack definitions. □

Lemma B.6 (*cl* transitive)**.** *If* $\mathfrak{D} \subseteq cl(\xi)(\mathfrak{A})$ *then* $b \in cl(\xi \cup \mathfrak{D}{\to}b)(\mathfrak{A})$.

Proof.

$\mathfrak{D} \subseteq cl(\xi)(\mathfrak{A})$ Given

$\subseteq cl(\xi \cup \mathfrak{D}{\to}b)(\mathfrak{A})$ $cl(-)(\mathscr{O})$ monotone

By definition, there exists $n \in \mathbb{N}$ such that $\mathfrak{D} \subseteq cl^n(\xi \cup \mathfrak{D}{\to}b)(\mathfrak{A})$. Thus,

$b \in cl^{n+1}(\xi \cup \mathfrak{D}{\to}b)(\mathfrak{A})$ By definition of cl^{n+1}

$\subseteq cl(\xi \cup \mathfrak{D}{\to}b)(\mathfrak{A})$ By definition of cl □

Lemma B.7 (Extraneous Assumptions)**.** $cl(\xi)(\mathfrak{D}) = cl(\xi)(\mathfrak{D} \cap FV(\xi)) \cup \mathfrak{D}$.

Proof. Straightforward. For the $\subseteq$ direction it suffices to show that for all $n \in \mathbb{N}$ we have $cl^n(\xi)(\mathfrak{D}) \subseteq cl(\xi)(\mathfrak{D} \cap FV(\xi)) \cup \mathfrak{D}$. We proceed by induction on n. The $n = 0$ case holds as $cl^0(\xi)(\mathfrak{D}) = \mathfrak{D} \subseteq _ \cup \mathfrak{D}$. As for the $n = m+1$ case, suppose $\mathfrak{A}{\to}\hat{a} \in \xi$ and $\mathfrak{A} \subseteq cl^m(\xi)(\mathfrak{D})$. By the i.h., $\mathfrak{A} \subseteq cl(\xi)(\mathfrak{D} \cap FV(\xi)) \cup \mathfrak{D}$. But $\mathfrak{A} \subseteq FV(\xi)$ so $\mathfrak{A} \subseteq cl(\xi)(\mathfrak{D} \cap FV(\xi)) \cup (\mathfrak{D} \cap FV(\xi)) = cl(\xi)(\mathfrak{D} \cap FV(\xi))$. Thus $b \in cl(\xi)(\mathfrak{D} \cap FV(\xi)) \subseteq cl(\xi)(\mathfrak{D} \cap FV(\xi)) \cup \mathfrak{D}$. The $\supseteq$ direction is similar. □

Lemma B.8 (Consequence for *cl*)**.**

If $\mathfrak{A} \subseteq cl(\xi)(\mathfrak{B})$ *and* $\mathfrak{A}{\to}a \in \xi$ *then* $a \in cl(\xi)(\mathfrak{B})$.

Proof. Straightforward. □

Lemma B.9 (Subtraction and *cl*)**.**

If $\mathfrak{A} \subseteq cl(\xi \cup (\xi' - \mathfrak{A}))(\mathfrak{B})$ *then* $cl(\xi \cup (\xi' - \mathfrak{A}))(\mathfrak{B}) \subseteq cl(\xi \cup \xi')(\mathfrak{B})$.

Proof. Straightforward consequence of the following statement

(which can be proved straightforwardly by induction on $m+n$, case analyzing m):

If $m \in \mathbb{N}$ and $n \in \mathbb{N}$ and $\mathfrak{A} \subseteq cl^m(\xi \cup (\xi' - \mathfrak{A}))(\mathfrak{B})$

then $cl^{m+n}(\xi \cup (\xi' - \mathfrak{A}))(\mathfrak{B}) \subseteq cl(\xi \cup \xi')(\mathfrak{B})$. □

Appendix C

Syntactic Metatheory of Declarative (Refined) System

Definition C.1 (Height and Structure)**.** Given a derivation $\mathscr{D}$ of judgment form $\mathscr{J}$ that is mutually recursive with judgment forms $\mathscr{J}_1, \ldots, \mathscr{J}_n$ ($n \geq 0$), we define the *height* $hgt(\mathscr{D})$ of $\mathscr{D}$ to be the number of rules concluding a judgment of form $\mathscr{J}$ or $\mathscr{J}_1$ or $\ldots$ or $\mathscr{J}_n$ that are used in the longest branch (that is, sequence of judgments of form $\mathscr{J}$ or $\mathscr{J}_1$ or $\ldots$ or $\mathscr{J}_n$) in $\mathscr{D}$. Further, we define the *structure* of $\mathscr{D}$ to be the tree structure of applications of rules in $\mathscr{D}$ concluding a judgment of form $\mathscr{J}$ or $\mathscr{J}_1$ or $\ldots$ or $\mathscr{J}_n$, and ultimately concluding the conclusion of $\mathscr{D}$.

Note that we overload "," and "$\mid$" as syntax and $-,-$ and $-\mid-$ as a list concatenation metaoperation on logical contexts (","), semantic substitutions (","), and algebras ("$\mid$").

Definition C.2 (Denotation of Syntactic Substitution)**.**

Assume $\mathscr{D} :: \Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma$. Define $[\![\mathscr{D}]\!]_\delta$ for each $\vdash \delta : \Theta_0; \Gamma_0$ as follows:

$$\left[\!\!\left[\frac{}{\Theta_0; \Gamma_0 \vdash \cdot : \cdot; \cdot} \right]\!\!\right]_\delta = \cdot$$

$$\left[\!\!\left[\frac{\mathscr{D}_0 :: \Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma \qquad \overline{\Theta_0} \vdash t : \tau \qquad \cdots}{\Theta_0; \Gamma_0 \vdash \sigma, t/a : \Theta, a \div \tau; \Gamma} \right]\!\!\right]_\delta = [\![\mathscr{D}_0]\!]_\delta, [\![t]\!]_{\lfloor \delta \rfloor} / a$$

$$\left[\!\!\left[\dfrac{\mathscr{D}_0 :: \Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma \qquad \overset{\mathsf{d}}{\div}\Theta_0 \vdash t : \tau \qquad \cdots}{\Theta_0;\Gamma_0 \vdash \sigma, t/a : \Theta, a \overset{\mathsf{d}}{\div} \tau[,a\,\mathrm{Id}];\Gamma}\right]\!\!\right]_\delta = [\![\mathscr{D}_0]\!]_\delta, [\![t]\!]_{\delta\restriction_{\overset{\mathsf{d}}{\div}\Theta_0}}/a$$

$$\left[\!\!\left[\dfrac{\mathscr{D}_0 :: \Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma \qquad \Theta_0 \vdash [\sigma]\varphi\ \mathsf{true}}{\Theta_0;\Gamma_0 \vdash \sigma : \Theta, \varphi;\Gamma}\right]\!\!\right]_\delta = [\![\mathscr{D}_0]\!]_\delta$$

$$\left[\!\!\left[\dfrac{\mathscr{D}_0 :: \Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma \qquad \mathscr{E} :: \Theta_0;\Gamma_0 \vdash v \Leftarrow [\lfloor\sigma\rfloor]P \qquad \cdots}{\Theta_0;\Gamma_0 \vdash \sigma, v : [\lfloor\sigma\rfloor]P/x : \Theta;\Gamma, x : P}\right]\!\!\right]_\delta = [\![\mathscr{D}_0]\!]_\delta, [\![\mathscr{E}]\!]_\delta/x$$

We may also write $[\![\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma]\!]_\delta$, if the derivation $\mathscr{D}$ is clear from context, or $[\![\sigma]\!]_\delta$ if $\mathscr{D}, \Theta_0, \Gamma_0, \Theta$, and Γ are clear from context. As we proceed, we will prove that this definition is sound, that is:

$$[\![\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma]\!] : \underbrace{[\![\Theta_0;\Gamma_0]\!]}_{\{\delta \mid \vdash \delta : \Theta_0;\Gamma_0\}} \to \underbrace{[\![\Theta;\Gamma]\!]}_{\{\delta \mid \vdash \delta : \Theta;\Gamma\}}$$

In particular, see Lemmas C.22, C.25, and E.28.

We often implicitly use the next thirteen lemmas.

Lemma C.1 (Filter Out Prog. Vars. Syn.)**.** *If* $\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma$ *then* $\Theta_0 \vdash \lfloor\sigma\rfloor : \Theta$.

Proof. By structural induction on the given syntactic substitution typing derivation. (Use obvious fact that $\lfloor -\rfloor$ is idempotent.) □

Lemma C.2 (Filter Out Propositions)**.**

(1) If $\vdash \delta : \Theta$ *then* $\vdash \delta : \overline{\Theta}$.

(2) If $\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma$ *then* $\Theta_0;\Gamma_0 \vdash \sigma : \overline{\Theta};\Gamma$.

(3) If $\Theta_0 \vdash \sigma : \Xi$ *then* $\overline{\Theta_0} \vdash \sigma : \Xi$.

Proof. Each part is proved by structural induction on the given derivation. □

Lemma C.3 (Stratify Sem. Subs. 1)**.** *If* $\vdash \delta : \Theta, \Theta_1$ *then there exist* δ' *and* δ_1 *such that* $\delta = \delta', \delta_1$ *and* $\vdash \delta' : \Theta$ *and* $\vdash \delta', \delta_1 : \Theta, \Theta_1$.

Proof. By structural induction on Θ_1. □

Lemma C.4 (Stratify Sem. Subs.)**.**

Assume $n \in \mathbb{N}$ *and read* $\Theta_1, \ldots, \Theta_m$ *as* $\cdot$ *for* $m = 0$.

If $\vdash \delta : \Theta, \Theta_1, \ldots, \Theta_n$ *then there exist* $\delta', \delta_1, \ldots, \delta_n$ *such that* $\delta = \delta', \delta_1, \ldots, \delta_n$ *and* $\vdash \delta' : \Theta$ *and* $\vdash \delta', \delta_1 : \Theta, \Theta_1$ *and* $\ldots$ *and* $\vdash \delta', \delta_1, \ldots, \delta_n : \Theta, \Theta_1, \ldots, \Theta_n$.

Proof. By induction on n. The $n = 0$ case is immediate (put $\delta' = \delta$).

Suppose $\vdash \delta : \Theta, \Theta_1, \ldots, \Theta_k, \Theta_{k+1}$. By Lemma C.3 (Stratify Sem. Subs. 1), there exist $\tilde{\delta}$ and δ_{k+1} such that $\vdash \tilde{\delta} : \Theta, \Theta_1, \ldots, \Theta_k$ and $\vdash \tilde{\delta}, \delta_{k+1} : \Theta, \Theta_1, \ldots, \Theta_k, \Theta_{k+1}$. By the i.h., there exist $\delta', \delta_1, \ldots, \delta_k$ such that $\tilde{\delta} = \delta', \delta_1, \ldots, \delta_k$ and $\vdash \delta' : \Theta$ and $\vdash \delta', \delta_1 : \Theta, \Theta_1$ and $\ldots$ and $\vdash \delta', \delta_1, \ldots, \delta_k : \Theta, \Theta_1, \ldots, \Theta_k$. Rewriting a semantic substitution above with the equation just obtained, we have $\vdash \delta', \delta_1, \ldots, \delta_k, \delta_{k+1} : \Theta, \Theta_1, \ldots, \Theta_k, \Theta_{k+1}$. □

Lemma C.5 (Sem. Subs. Length)**.** *If* $\vdash \delta : \Theta$ *then* $len(\delta) = len\big(\overline{\Theta}\big)$.

Proof. By structural induction on the given semantic substitution derivation. □

Lemma C.6 (Sorting Weakening)**.** *Assume* $\Xi \subseteq \Xi'$.

(1) If $\Xi \vdash t : \tau\ [\xi_t]$ *then* $\Xi' \vdash t : \tau\ [\xi_t]$ *by a derivation of equal structure.*

(2) If $\Xi; [\tau] \vdash \boldsymbol{t} : \kappa$ *then* $\Xi'; [\tau] \vdash \boldsymbol{t} : \kappa$ *by a derivation of equal structure.*

Proof. By structural induction on the given sorting derivation. Parts (1) and (2) are mutually recursive. □

Lemma C.7 (Ix. $[\![-]\!]$ Weak. Invariant)**.** *Assume* $\vdash \delta : \Xi$ *and* $\vdash \delta' : \Xi'$ *and* $\Xi \subseteq \Xi'$ *and* $\delta' \restriction_{\Xi} = \delta$.

(1) If $\Xi \vdash t : \tau\ [\xi_t]$ *then* $[\![\Xi \vdash t : \tau\ [\xi_t]]\!]_\delta = [\![\Xi' \vdash t : \tau\ [\xi_t]]\!]_{\delta'}$.

(2) If $\Xi; [\tau] \vdash \boldsymbol{t} : \kappa$ *then* $[\![\Xi; [\tau] \vdash \boldsymbol{t} : \kappa]\!]_\delta = [\![\Xi'; [\tau] \vdash \boldsymbol{t} : \kappa]\!]_{\delta'}$.

Proof. By structural induction on the given index sorting derivation. Parts (1) and (2) are mutually recursive. Use Lemma C.6 (Sorting Weakening). □

Lemma C.8 (Prop. Truth Weakening)**.** *If* $\Theta \vdash \varphi\ \mathsf{true}$ *and* $\Theta \subseteq \Theta'$ *then* $\Theta' \vdash \varphi\ \mathsf{true}$.

Proof. Assume $\vdash \delta : \Theta'$.

$\Theta \vdash \varphi\ \mathsf{true}$	Given
$\overline{\Theta} \vdash \varphi : \mathbb{B}$	Presupposed derivation
$\overline{\Theta'} \vdash \varphi : \mathbb{B}$	By Lemma C.6 (Sorting Weakening)
$[\![\varphi]\!]_\delta = [\![\varphi]\!]_{\delta\restriction_{\overline{\Theta}}}$	By Lemma C.7 (Ix. $[\![-]\!]$ Weak. Invariant)
$= \{\bullet\}$	By inversion on $\Theta \vdash \varphi\ \mathsf{true}$
$\Theta' \vdash \varphi\ \mathsf{true}$	By PropTrue

□

Lemma C.9 (Ix. Subs. Weakening)**.** *If* $\Theta_0 \vdash \sigma : \Theta$ *and* $\Theta_0 \subseteq \Theta_0'$ *then* $\Theta_0' \vdash \sigma : \Theta$.

Proof. By structural induction on $\Theta_0 \vdash \sigma : \Theta$.

- **Case**

$$\frac{}{\Theta \vdash \cdot : \cdot}\ \text{SubstEmpty}$$

  $\Theta_0' \vdash \cdot : \cdot$ By SubstEmpty

- **Case**

$$\frac{\Theta_0 \vdash \sigma' : \Theta' \qquad \overline{\Theta_0} \vdash t : \tau \qquad a \notin dom(\Theta')}{\Theta_0 \vdash \sigma', t/a : \Theta', a \div \tau}\ \text{SubstIx}$$

$\Theta_0 \vdash \sigma' : \Theta'$	Subderivation
$\Theta_0' \vdash \sigma' : \Theta'$	By i.h.
$\overline{\Theta_0} \vdash t : \tau$	Premise
$\overline{\Theta_0'} \vdash t : \tau$	By Lemma C.6 (Sorting Weakening)
$a \notin dom(\Theta')$	Premise
$\Theta_0' \vdash \sigma', t/a : \Theta', a \div \tau$	By SubstIx

- **Case** SubstIxDet: Similar to SubstIx case, but also uses $\overset{\mathrm{d}}{\div}\Theta_0 \subseteq \Theta_0$ and transitivity of $\subseteq$.
- **Case** SubstProp: Similar to SubstIx case.
- **Case** SubstVal: Impossible. □

Lemma C.10 (Ix. Subst. $[\![-]\!]$ Weak. Invariant)**.** *If* $\vdash \delta : \Theta_0$ *and* $\vdash \delta' : \Theta_0'$ *and* $\Theta_0 \subseteq \Theta_0'$ *and* $\delta'\restriction_{\Theta_0} = \delta$. *If* $\Theta_0 \vdash \sigma : \Theta$ *then* $[\![\Theta_0 \vdash \sigma : \Theta]\!]_\delta = [\![\Theta_0' \vdash \sigma : \Theta]\!]_{\delta'}$ *(for any such weakened derivation).*

Proof. By structural induction on $\Theta_0 \vdash \sigma : \Theta$. Use Lemma C.7 (Ix. $[\![-]\!]$ Weak. Invariant). □

Lemma C.11 (Ix. Id. Subs. Extension)**.** *If* $\Theta_0; \cdot \vdash \sigma : \Theta; \cdot$ *and* $a \notin dom(\Theta) \cup dom(\Theta_0)$ *then* $\Theta_0, a \div \tau; \cdot \vdash \sigma, a/a : \Theta, a \div \tau; \cdot$ *and* $\Theta_0, a \overset{\mathrm{d}}{\div} \tau; \cdot \vdash \sigma, a/a : \Theta, a \overset{\mathrm{d}}{\div} \tau; \cdot$ *and* $\Theta_0, a \overset{\mathrm{d}}{\div} \tau, a\,\mathrm{Id}; \cdot \vdash \sigma, a/a : \Theta, a \overset{\mathrm{d}}{\div} \tau, a\,\mathrm{Id}; \cdot$.

Proof. We prove $\Theta_0, a \div \tau; \cdot \vdash \sigma, a/a : \Theta, a \div \tau; \cdot$. The others are similar (use SubstIxDet and def. of $\overset{\mathrm{d}}{\div}-$).

$\Theta_0;\cdot \vdash \sigma : \Theta;\cdot$	Given
$a \div \tau \in (\Theta_0, a \div \tau)$	By def. of membership
$\Theta_0, a \div \tau;\cdot \vdash \sigma : \Theta;\cdot$	By Lemma C.9 (Ix. Subs. Weakening)
$\overline{\Theta_0}, a \div \tau \vdash a : \tau$	By IxVar
$\Theta_0, a \div \tau;\cdot \vdash \sigma, a/a : \Theta, a \div \tau$	By SubstIx

□

Substitution can affect ξ. To handle this, we define substitution on ξ. The definition uses the notion of free-variable image of a substitution at a set of index variables.

Definition C.3 (FV Image of a Substitution)**.** Assume $\Theta_0 \vdash \sigma : \Theta$.

Given a set $\mathfrak{D}$ of index variables, define the *free-variable image of* σ *at* $\mathfrak{D}$ by

$$\langle\sigma\rangle\mathfrak{D} = \bigcup_{a\in\mathfrak{D}} \begin{cases} \{a\} & \text{if } \sigma(a) \text{ is undefined} \\ FV(\sigma(a)) & \text{else} \end{cases}$$

If $\mathfrak{D} \subseteq dom(\sigma)$ then this definition is equivalent to $\langle\sigma\rangle\mathfrak{D} = \cup_{a\in\mathfrak{D}} FV(\sigma(a))$.

Definition C.4 (Substitution on ξ)**.** Assume $\Theta_0 \vdash \sigma : \Theta$. Define ${}^{\mathsf{d}}[\sigma]\xi$ by

$${}^{\mathsf{d}}[\sigma]\cdot = \cdot$$

$${}^{\mathsf{d}}[\sigma](\xi, \mathfrak{D}{\to}a) = \begin{cases} {}^{\mathsf{d}}[\sigma]\xi \cup (\langle\sigma\rangle\mathfrak{D}{\to}a) & \text{if } \sigma(a) \text{ is undefined} \\ {}^{\mathsf{d}}[\sigma]\xi \cup (\langle\sigma\rangle\mathfrak{D}{\to}\sigma(a)) & \text{if } \sigma(a) \text{ is a variable} \\ {}^{\mathsf{d}}[\sigma]\xi & \text{else} \end{cases}$$

If $pos(\xi) \subseteq dom(\sigma)$ then this definition is equivalent to the one given by

$$
\begin{aligned}
{}^{\mathsf{d}}[\sigma]\cdot &= \cdot \\
{}^{\mathsf{d}}[\sigma](\xi, \mathfrak{D}{\to}a) &= \begin{cases} {}^{\mathsf{d}}[\sigma]\xi \cup (\langle\sigma\rangle\mathfrak{D}{\to}\sigma(a)) & \text{if } \sigma(a) \text{ is a variable} \\ {}^{\mathsf{d}}[\sigma]\xi & \text{else} \end{cases}
\end{aligned}
$$

Lemma C.12 (Value-Det. Substitution)**.** *If* $\Theta_0 \vdash \sigma : \Theta$ *then* ${}^{\mathsf{d}\cdot}\Theta_0 \vdash \sigma\restriction_{{}^{\mathsf{d}\cdot}\Theta} : {}^{\mathsf{d}\cdot}\Theta$.

Proof. By structural induction on $\Theta_0 \vdash \sigma : \Theta$.

- **Case**

$$\dfrac{}{\Theta_0 \vdash \underbrace{\cdot}_{\sigma} : \underbrace{\cdot}_{\Theta}}\ \text{SubstEmpty}$$

$${}^{\mathsf{d}\cdot}\Theta_0 \vdash \underbrace{\cdot}_{\sigma\restriction_{{}^{\mathsf{d}\cdot}\Theta}} : \underbrace{\cdot}_{{}^{\mathsf{d}\cdot}\Theta} \quad \text{By SubstEmpty}$$

- **Case**

$$\dfrac{\Theta_0 \vdash \sigma' : \Theta' \qquad \overline{\Theta_0} \vdash t : \tau \qquad a \notin dom(\Theta')}{\Theta_0 \vdash \sigma', t/a : \Theta', a \div \tau}\ \text{SubstIx}$$

$\Theta_0 \vdash \sigma' : \Theta'$	Subderivation
${}^{\mathsf{d}\cdot}\Theta_0 \vdash \sigma'\restriction_{{}^{\mathsf{d}\cdot}\Theta'} : {}^{\mathsf{d}\cdot}\Theta'$	By i.h.
${}^{\mathsf{d}\cdot}\Theta_0 \vdash (\sigma', t/a)\restriction_{{}^{\mathsf{d}\cdot}\Theta'} : {}^{\mathsf{d}\cdot}\Theta'$	By def. of restriction ($a \notin dom({}^{\mathsf{d}\cdot}\Theta')$)
${}^{\mathsf{d}\cdot}\Theta_0 \vdash (\sigma', t/a)\restriction_{{}^{\mathsf{d}\cdot}(\Theta', a\div\tau)} : {}^{\mathsf{d}\cdot}(\Theta', a \div \tau)$	By def. of ${}^{\mathsf{d}\cdot}-$

- **Case**
$$\frac{\Theta_0 \vdash \sigma' : \Theta' \qquad \overset{\mathrm{d}}{\div}\Theta_0 \vdash t : \tau \qquad a \notin dom(\Theta')}{\Theta_0 \vdash \sigma', t/a : \Theta', a \overset{\mathrm{d}}{\div} \tau[, a\,\mathrm{Id}]}\ \text{SubstIxDet}$$

$\Theta_0 \vdash \sigma' : \Theta'$	Subderivation
$\overset{\mathrm{d}}{\div}\Theta_0 \vdash \sigma'\restriction_{\overset{\mathrm{d}}{\div}\Theta'} : \overset{\mathrm{d}}{\div}\Theta'$	By i.h.
$\overset{\mathrm{d}}{\div}\Theta_0 \vdash t : \tau$	Premise
$\overset{\mathrm{d}}{\div}(\overset{\mathrm{d}}{\div}\Theta_0) \vdash t : \tau$	Because $\overset{\mathrm{d}}{\div}-$ idempotent
$a \notin dom(\Theta')$	Premise
$a \notin dom(\overset{\mathrm{d}}{\div}\Theta')$	As $\overset{\mathrm{d}}{\div}\Theta' \subseteq \Theta'$
$\overset{\mathrm{d}}{\div}\Theta_0 \vdash \sigma'\restriction_{\overset{\mathrm{d}}{\div}\Theta'}, t/a : \overset{\mathrm{d}}{\div}\Theta', a \overset{\mathrm{d}}{\div} \tau[, a\,\mathrm{Id}]$	By SubstIxDet
$\overset{\mathrm{d}}{\div}\Theta_0 \vdash \sigma'\restriction_{\overset{\mathrm{d}}{\div}\Theta', a\overset{\mathrm{d}}{\div}\tau}, t/a : \overset{\mathrm{d}}{\div}\Theta', a \overset{\mathrm{d}}{\div} \tau[, a\,\mathrm{Id}]$	By prop. of restriction ($a \notin dom(\sigma')$)
$\overset{\mathrm{d}}{\div}\Theta_0 \vdash \sigma'\restriction_{\overset{\mathrm{d}}{\div}(\Theta', a\overset{\mathrm{d}}{\div}\tau)}, t/a : \overset{\mathrm{d}}{\div}(\Theta', a \overset{\mathrm{d}}{\div} \tau, a\,\mathrm{Id})$	Because $\overset{\mathrm{d}}{\div}\Theta' \subseteq \Theta'$
$\overset{\mathrm{d}}{\div}\Theta_0 \vdash (\sigma', t/a)\restriction_{\overset{\mathrm{d}}{\div}(\Theta', a\overset{\mathrm{d}}{\div}\tau)} : \overset{\mathrm{d}}{\div}(\Theta', a \overset{\mathrm{d}}{\div} \tau, a\,\mathrm{Id})$	By def. of restriction
$\overset{\mathrm{d}}{\div}\Theta_0 \vdash \sigma\restriction_{\overset{\mathrm{d}}{\div}\Theta} : \overset{\mathrm{d}}{\div}\Theta$	By equalities

- **Case** SubstProp: Similar to SubstIx case.

- **Case** SubstVal: Impossible. □

Definition C.5 (Id. Substitution)**.** The *identity substitution* on $(\Theta;\Gamma)$, written $id_{\Theta;\Gamma}$, is

defined as follows:

$$id_{\cdot;\cdot} = \cdot$$

$$id_{\Theta,a:\tau;\cdot} = id_{\Theta;\cdot}, a/a$$

$$id_{\Theta,\varphi;\cdot} = id_{\Theta;\cdot}$$

$$id_{\Theta;\Gamma,x:R} = id_{\Theta;\Gamma}, x : R/x$$

We may write id_Θ for $id_{\Theta;\cdot}$ and id_Γ for $id_{\cdot;\Gamma}$.

We may write Θ/Θ for id_Θ and Γ/Γ for id_Γ and $\Theta;\Gamma/\Theta;\Gamma$ for $id_{\Theta;\Gamma}$.

Definition C.6 (Id. Subst. Subtract)**.** Define $\sigma - id_{\Theta;\Gamma}$ by

$$\cdot - id_{\Theta;\Gamma} = \cdot$$

$$(\sigma, t/a) - id_{\Theta;\Gamma} = \begin{cases} \sigma - id_{\Theta;\Gamma} & \text{if } t = a \text{ and } a \in dom(\Theta) \\ (\sigma - id_{\Theta;\Gamma}), t/a & \text{else} \end{cases}$$

$$(\sigma, v : R/x) - id_{\Theta;\Gamma} = \begin{cases} \sigma - id_{\Theta;\Gamma} & \text{if } v = x \text{ and } x \in dom(\Gamma) \\ (\sigma - id_{\Theta;\Gamma}), v : R/x & \text{else} \end{cases}$$

Lemma C.13 (Ix. Id. Subst.)**.**

If $\Theta_0 \vdash \sigma : \Theta$ and $\mathscr{O}$ is the subject of derivation $\mathscr{D}$ under Θ

then for any Θ' we have $[\sigma]\mathscr{O} = [\sigma - id_{\Theta'}]\mathscr{O}$. In particular, $[\Theta/\Theta]\mathscr{O} = \mathscr{O}$.

Proof. By structural induction on $\mathscr{D}$. Straightforward. □

Lemma C.14 (ξ Subst. Union)**.** ${}^{\mathsf{d}}[\sigma](\xi_1 \cup \xi_2) = {}^{\mathsf{d}}[\sigma]\xi_1 \cup {}^{\mathsf{d}}[\sigma]\xi_2$

Proof. Assume $\mathfrak{D}'{\to}a' \in {}^{\mathsf{d}}[\sigma](\xi_1 \cup \xi_2)$. Then there exists $\mathfrak{D}{\to}a$ in either ξ_1 or ξ_2 such that $\langle\sigma\rangle\mathfrak{D} = \mathfrak{D}'$ and either $a' = \sigma(a)$ or $a' = a$ and $\sigma(a)$ is undefined. In any case, $\mathfrak{D}'{\to}a' \in {}^{\mathsf{d}}[\sigma]\xi_1 \cup {}^{\mathsf{d}}[\sigma]\xi_2$. Similarly, ${}^{\mathsf{d}}[\sigma](\xi_1 \cup \xi_2) \supseteq {}^{\mathsf{d}}[\sigma]\xi_1 \cup {}^{\mathsf{d}}[\sigma]\xi_2$. □

Lemma C.15 (ξ Subst. Monotone)**.** *If* $\xi' \supseteq \xi$ *then* ${}^{\mathsf{d}}[\sigma]\xi' \supseteq {}^{\mathsf{d}}[\sigma]\xi$.

Proof. Suppose $\xi' \supseteq \xi$. Assume $\mathfrak{D}'{\to}a' \in {}^{\mathsf{d}}[\sigma]\xi$. Then there exists $\mathfrak{D}{\to}a \in \xi$ such that $\langle\sigma\rangle\mathfrak{D} = \mathfrak{D}'$ and either $a' = \sigma(a)$ or $a' = a$ and $\sigma(a)$ is undefined. Because $\xi' \supseteq \xi$ and $\mathfrak{D}{\to}a \in \xi$, we have $\mathfrak{D}{\to}a \in \xi'$. We need to show that $\mathfrak{D}'{\to}a' \in {}^{\mathsf{d}}[\sigma]\xi'$, that is, that there exists $\mathfrak{B}{\to}b \in \xi'$ such that $\langle\sigma\rangle\mathfrak{B} = \mathfrak{D}'$ and either $a' = \sigma(b)$ or $a' = b$ and $\sigma(a')$ is undefined; but we have just shown that $\mathfrak{D}{\to}a$ is such an element. □

Lemma C.16 (Append Ix. Spine)**.** *If* $\Xi;[\tau] \vdash \boldsymbol{t} : \kappa'$ *and* $\Xi;[\kappa'] \vdash \boldsymbol{u} : \kappa$
then $\Xi;[\tau] \vdash \boldsymbol{t}, \boldsymbol{u} : \kappa$.

Proof. By structural induction on $\Xi;[\tau] \vdash \boldsymbol{t} : \kappa'$. Straightforward. □

Lemma C.17 (Ix. Syntactic Substitution)**.**

(1) If $\Xi \vdash t : \tau\ [\xi_t]$ *and* $\Xi_0 \vdash \sigma : \Xi$ *then* $\Xi_0 \vdash [\sigma]t : \tau\ [{}^{\mathsf{d}}[\sigma]\xi_t]$

(2) If $\Xi;[\tau] \vdash \boldsymbol{t} : \kappa$ *and* $\Xi_0 \vdash \sigma : \Xi$ *then* $\Xi_0;[\tau] \vdash [\sigma]\boldsymbol{t} : \kappa$.

(3) If $\Xi_0 \vdash u : \omega$ *and* $\Xi_0;[\omega] \vdash \boldsymbol{t} : \kappa$ *then* $\Xi_0 \vdash \langle u \mid \boldsymbol{t}\rangle : \kappa$.

Proof. Define $m \in \mathbb{N}$ by

$$m = \sum_{\substack{(t'/c) \in \sigma \\ \text{such that } t' \neq c}} \mathit{size}(\Xi(c))$$

if we are in part (1) or part (2), and $m = \mathit{size}(\omega)$ if we are in part (3) where *size* measures the structural size of a sort. Proceed by lexicographic induction, first, on m, second, on

the size of the part number where $(3) < (1)$ and $(3) < (2)$ and $(j) = (k)$ otherwise, third, the structure of t in part (1) and $\boldsymbol{t}$ in parts (2) and (3), and fourth, the height of the sorting derivation for t (part (1)).

(1) • **Case**

$$\frac{(a:\tau) \in \Xi}{\Xi \vdash \underbrace{a}_{t} : \tau\ [\underbrace{\cdot}_{\xi_t}]}\ \text{IxVar}$$

	$(a:\tau) \in \Xi$	Premise
	$\Xi_0 \vdash \sigma : \Xi$	Given
	$\Xi = \Xi_1, a:\tau, \Xi_2$	By inversion
	$\sigma = \sigma_1, u/a, \sigma_2$	$''$
	$\Xi_0 \vdash u : \tau\ [\xi_u]$	$''$
	$\xi_u \supseteq \cdot$	Empty set subset of every set
	$\Xi_0 \vdash u : \tau\ [\cdot]$	By IxSub (if $\xi' \neq \cdot$) or by $\xi' = \cdot$
	$[\sigma]t = [\sigma_1, u/a, \sigma_2]a$	By equalities
	$= u$	By def. of $[-]-$ and variable lookup
	$\Xi_0 \vdash u : \tau\ [^{\mathsf{d}}[\sigma]\cdot]$	By Def. C.4
☞	$\Xi_0 \vdash [\sigma]t : \tau\ [^{\mathsf{d}}[\sigma]\xi_t]$	By equalities

• **Case**

$$\frac{\Xi, a \div \kappa \vdash t_0 : \tau_0\ [\xi_{t_0}]}{\Xi \vdash \lambda a.t_0 : \kappa \Rightarrow \tau_0\ [\cdot]}\ \text{Ix}\lambda$$

	$\Xi_0 \vdash \sigma : \Xi$	Given
	$\Xi_0, a \div \kappa \vdash \sigma, a/a : \Xi, a \div \kappa$	By Lemma C.11 (Ix. Id. Subs. Extension)
	$\Xi, a \div \kappa \vdash t_0 : \tau_0\ [\xi_{t_0}]$	Subderivation
	$\Xi_0, a \div \kappa \vdash [\sigma, a/a]t_0 : \tau_0\ [\xi']$	By i.h. (same m; same part; smaller t)
	$\Xi_0, a \div \kappa \vdash [\sigma]t_0 : \tau_0\ [\xi']$	Identity subst.
	$\Xi_0 \vdash \lambda a.[\sigma]t_0 : \tau_0\ [\cdot]$	By Ixλ
	$\Xi_0 \vdash [\sigma](\lambda a.t_0) : \tau_0\ [\cdot]$	By def. of subst.
☞	$\Xi_0 \vdash [\sigma](\lambda a.t_0) : \tau_0\ [^{\mathsf{d}}[\sigma]\xi_t]$	By Def. C.4 and equality

- **Case**

$$\frac{(a : \omega) \in \Xi \qquad \Xi; [\omega] \vdash \boldsymbol{t} : \kappa}{\Xi \vdash a(\boldsymbol{t}) : \kappa\ [\cdot]}\ \text{IxApp}$$

	$(a : \omega) \in \Xi$	Premise
	$\Xi_0 \vdash \sigma : \Xi$	Given
	$\Xi_0 \vdash \sigma(a) : \omega$	By inversion on substitution typing (weakening if needed)
	$\Xi; [\omega] \vdash \boldsymbol{t} : \kappa$	Subderivation
	$\Xi_0; [\omega] \vdash [\sigma]\boldsymbol{t} : \kappa$	By i.h. (same m; same part size; smaller $t/\boldsymbol{t}$)
	$\Xi_0 \vdash \langle \sigma(a) \mid [\sigma]\boldsymbol{t} \rangle : \kappa\ [\xi']$	By i.h. (m same or smaller; smaller part)
	$\Xi_0 \vdash [\sigma](a(\boldsymbol{t})) : \kappa\ [\xi']$	By def. of $[-]-$
	$\xi' \supseteq \cdot$	Empty set subset of every set
	$\Xi_0 \vdash [\sigma](a(\boldsymbol{t})) : \kappa\ [\cdot]$	By IxSub (if $\xi' \neq \cdot$) or by $\xi' = \cdot$
☞	$\Xi_0 \vdash [\sigma](a(\boldsymbol{t})) : \kappa\ [{}^{\mathsf{d}}[\sigma]\xi_t]$	By Def. C.4 and equality

- **Case**

$$\frac{(a \stackrel{\mathsf{d}\cdot}{\div} \kappa) \in \Xi \qquad (u \stackrel{\mathsf{d}\cdot}{\div} \kappa) \notin \Xi \qquad \stackrel{\mathsf{d}\cdot}{\div}\Xi \vdash u : \kappa\ [\xi_u]}{\Xi \vdash a = u : \mathbb{B}\ [FV(u) \to a]}\ \text{Ix=L}$$

$\Xi_0 \vdash \sigma : \Xi$	Given
$\stackrel{\mathsf{d}\cdot}{\div}\Xi_0 \vdash \sigma\restriction_{\stackrel{\mathsf{d}\cdot}{\div}\Xi} : \stackrel{\mathsf{d}\cdot}{\div}\Xi$	By Lemma C.12 (Value-Det. Substitution)
$\stackrel{\mathsf{d}\cdot}{\div}\Xi \vdash u : \kappa\ [\xi_u]$	Subderivation
$\stackrel{\mathsf{d}\cdot}{\div}\Xi_0 \vdash [\sigma\restriction_{\stackrel{\mathsf{d}\cdot}{\div}\Xi}]u : \kappa\ [{}^{\mathsf{d}}[\sigma\restriction_{\stackrel{\mathsf{d}\cdot}{\div}\Xi}]\xi_u]$	By i.h. (same m; same part size; smaller $t/\boldsymbol{t}$)
$\stackrel{\mathsf{d}\cdot}{\div}\Xi_0 \vdash [\sigma]u : \kappa\ [{}^{\mathsf{d}}[\sigma]\xi_u]$	By property of restriction
$\Xi_0 \vdash [\sigma]u : \kappa\ [{}^{\mathsf{d}}[\sigma]\xi_u]$	By Lemma C.6 (Sorting Weakening)

– **Case**: $\sigma(a)$ is a variable.

$(\sigma(a) \stackrel{\mathsf{d}}{\div} \kappa) \in \Xi_0$ By inversion on SubstIxDet and on IxVar

$(u \stackrel{\mathsf{d}}{\div} \kappa) \notin \Xi$ Premise

$([\sigma]u \stackrel{\mathsf{d}}{\div} \kappa) \notin \Xi_0$ Follows from def. of subst.

$\Xi_0 \vdash \sigma(a) = [\sigma]u : \mathbb{B}\ [FV([\sigma]u) \rightarrow \sigma(a)]$ By Ix=L

☞ $\Xi_0 \vdash [\sigma](a = u) : \mathbb{B}\ [FV([\sigma]u) \rightarrow \sigma(a)]$ By def. of substitution

$FV([\sigma]u) \rightarrow \sigma(a) = \langle\sigma\rangle FV(u) \rightarrow \sigma(a)$ Follows from Def. C.3

$= {}^{\mathsf{d}}[\sigma](FV(u) \rightarrow a)$ By Def. C.4

$= {}^{\mathsf{d}}[\sigma]\xi_t$ By equality

– **Case**: $\sigma(a) = (t_1, t_2)$.

$\kappa = \kappa_1 \times \kappa_2$ By inversion (u not a variable)

$u = (u_1, u_2)$ ″

$\Xi_0 \vdash \sigma(a) : \kappa_1 \times \kappa_2\ [\xi]$ By inversion

$\Xi_0 \vdash t_1 : \kappa_1\ [\xi_{t_1}]$ By inversion

$\Xi_0 \vdash t_2 : \kappa_2\ [\xi_{t_2}]$ ″

$\xi = \xi_{t_1} \cup \xi_{t_2}$ ″

$\Xi_0 \vdash [\sigma]u_1 : \kappa_1\ [\xi'_{u_1}]$ By inversion

$\Xi_0 \vdash [\sigma]u_2 : \kappa_2\ [\xi'_{u_2}]$ ″

${}^{\mathsf{d}}[\sigma]\xi_u = \xi'_{u_1} \cup \xi'_{u_2}$ ″

∗ **Case**: t_1, t_2, $[\sigma]u_1$, $[\sigma]u_2$ are variables.

$\Xi_0 \vdash t_1 = [\sigma]u_1 : \mathbb{B}\ [\cdot]$ By Ix=LR

$\Xi_0 \vdash t_2 = [\sigma]u_2 : \mathbb{B}\ [\cdot]$ By Ix=LR

$\Xi_0 \vdash (t_1,t_2) = ([\sigma]u_1,[\sigma]u_2) : \mathbb{B}\ [\cdot]$ By Ix=$\times$

$\Xi_0 \vdash (t_1,t_2) = ([\sigma]u_1,[\sigma]u_2) : \mathbb{B}\ [^{\mathsf{d}}[\sigma](FV(u)\to a)]$ By def. of $^{\mathsf{d}}[-]-$ ($\sigma(a)$ not var.)

$\Xi_0 \vdash [\sigma](a = u) : \mathbb{B}\ [^{\mathsf{d}}[\sigma]\xi_t]$ By equalities and by def.

$*$ **Case**: t_1, t_2, $[\sigma]u_1$, $[\sigma]u_2$ are pairs.

$\Xi_0 \vdash t_1 = [\sigma]u_1 : \mathbb{B}\ [_ \cup \xi'_{u_1}]$ By Ix=$\times$

$\Xi_0 \vdash t_2 = [\sigma]u_2 : \mathbb{B}\ [_ \cup \xi'_{u_2}]$ By Ix=$\times$

$\Xi_0 \vdash (t_1,t_2) = ([\sigma]u_1,[\sigma]u_2) : \mathbb{B}\ [_ \cup \xi'_{u_1} \cup _ \cup \xi'_{u_2}]$ By Ix=$\times$

$\Xi_0 \vdash (t_1,t_2) = ([\sigma]u_1,[\sigma]u_2) : \mathbb{B}\ [\cdot]$ By IxSub (if needed)

$\Xi_0 \vdash (t_1,t_2) = ([\sigma]u_1,[\sigma]u_2) : \mathbb{B}\ [^{\mathsf{d}}[\sigma](FV(u)\to a)]$ By def. of $^{\mathsf{d}}[-]-$ ($\sigma(a)$ not var.)

$\Xi_0 \vdash [\sigma](a = u) : \mathbb{B}\ [^{\mathsf{d}}[\sigma]\xi_t]$ By equalities and by def.

$*$ **Case**: Ix=L applies for t_k, $[\sigma]u_k$ and Ix= applies for t_{3-k}, $[\sigma]u_{3-k}$. Similar to preceding subcase.

$*$ **Case**: Ix=R applies for t_k, $[\sigma]u_k$ and Ix= applies for t_{3-k}, $[\sigma]u_{3-k}$. Similar to preceding subcase.

– **Case**: $\sigma(a)$ is not a variable or a pair.

$(\sigma(a) \stackrel{\mathsf{d}}{\div} \kappa) \notin \Xi_0$	Because $\sigma(a)$ is not a variable
$(u \stackrel{\mathsf{d}}{\div} \kappa) \notin \Xi$	Premise
$([\sigma]u \stackrel{\mathsf{d}}{\div} \kappa) \notin \Xi_0$	Follows from def. of subst.
$\stackrel{\mathsf{d}}{\div}\Xi_0 \vdash \sigma(a) : \kappa\ [_]$	By inversion on substitution sorting
$\Xi_0 \vdash \sigma(a) : \kappa\ [_]$	By Lemma C.6 (Sorting Weakening)
$\Xi_0 \vdash \sigma(a) = [\sigma]u : \mathbb{B}\ [\cdot]$	By Ix=
$\Xi_0 \vdash [\sigma](a = u) : \mathbb{B}\ [\cdot]$	By def. of substitution
☞ $\Xi_0 \vdash [\sigma](a = u) : \mathbb{B}\ [{}^{\mathsf{d}}[\sigma]\xi_t]$	By def. of ${}^{\mathsf{d}}[-]-$ ($\sigma(a)$ not var.) and equality

- **Case** Ix=R: Similar to Ix=L case.
- **Case** Ix=LR: Similar to Ix=L case but with more cases to consider.
- **Case** IxSub: Straightforward. This is the only case using the fourth part of the induction metric. Use Lemma C.15 (ξ Subst. Monotone).
- **Case** Ix$\wedge$, Ix=$\times$: Straightforward. Use Lemma C.14 (ξ Subst. Union).
- The remaining cases are straightforward (for all uses of i.h., m stays the same, the part number is equal in size, and $t/\boldsymbol{t}$ gets smaller).

(2) Straightforward (for all uses of i.h., m stays the same, the part number is equal in size, and $t/\boldsymbol{t}$ gets smaller).

(3) Consider the case where u is a variable: Then $\langle u \mid \boldsymbol{t} \rangle = u(\boldsymbol{t})$ by definition of $\langle - \mid - \rangle$; by inversion on IxVar, $(u : \omega) \in \Xi_0$; by IxApp, $\Xi_0 \vdash u(\boldsymbol{t}) : \kappa$.

Now consider the case where u is not a variable. We case analyze rules concluding $\Xi_0; [\omega] \vdash \boldsymbol{t} : \kappa$.

- **Case**

$$\frac{}{\Xi_0; [\underbrace{\kappa}_{\omega}] \vdash \underbrace{\cdot}_{\boldsymbol{t}} : \kappa}\ \text{IxSpineNil}$$

$$\begin{aligned}
\langle u \mid \boldsymbol{t} \rangle &= \langle u \mid \cdot \rangle && \text{Current case} \\
&= u && \text{By def. of } \langle - \mid - \rangle \\
\Xi_0 &\vdash u : \kappa && \text{Given}
\end{aligned}$$

- **Case**

$$\frac{\Xi_0 \vdash t_0 : \kappa_0 \qquad \Xi_0; [\omega_0] \vdash \boldsymbol{t}_0 : \kappa}{\Xi_0; [\kappa_0 \Rightarrow \omega_0] \vdash t_0, \boldsymbol{t}_0 : \kappa}\ \text{IxSpineEntry}$$

$$\begin{aligned}
u &= \lambda b.\, u_0 && \text{By inversion on Ix}\lambda \\
\Xi_0, b \div \kappa_0 &\vdash u_0 : \omega_0\ [_] && ''
\end{aligned}$$

$$\begin{aligned}
\Xi_0 &\vdash t_0 : \kappa_0 && \text{Subderivation} \\
\Xi_0 &\vdash [t_0/b]u_0 : \omega_0\ [_] && \text{By i.h. (smaller } m\text{)}
\end{aligned}$$

$$\Xi_0; [\omega_0] \vdash \boldsymbol{t}_0 : \kappa \qquad \text{Subderivation}$$

$$\begin{aligned}
\langle u \mid \boldsymbol{t} \rangle &= \langle \lambda b.\, u_0 \mid t_0, \boldsymbol{t}_0 \rangle && \text{By equalities} \\
&= \langle [t_0/b]u_0 \mid \boldsymbol{t}_0 \rangle && \text{By def. of } \langle - \mid - \rangle \\
\Xi_0 &\vdash \langle [t_0/b]u_0 \mid \boldsymbol{t}_0 \rangle : \kappa && \text{By i.h. (smaller } m\text{)}
\end{aligned}$$

- **Case**

$$\frac{\Xi_0; [\omega_k] \vdash \boldsymbol{t}_0 : \kappa}{\Xi_0; [\omega_1 \times \omega_2] \vdash .k, \boldsymbol{t}_0 : \kappa}\ \text{IxSpineProj}_k$$

In case IxApp concludes $\Xi_0 \vdash u : \omega$, use Lemma C.16 (Append Ix. Spine), the def. of $\langle - \mid - \rangle$, and rule IxApp.

Consider the case where IxApp does not conclude $\Xi_0 \vdash u : \omega$.

$u = (u_1, u_2)$	By inversion on Ix$\times$
$\Xi_0 \vdash u_k : \omega_k$	$''$
$\Xi_0; [\omega_k] \vdash \boldsymbol{t}_0 : \kappa$	Subderivation

$\langle u \mid \boldsymbol{t} \rangle = \langle (u_1, u_2) \mid .k, \boldsymbol{t}_0 \rangle$	By equalities
$= \langle u_k \mid \boldsymbol{t}_0 \rangle$	By def. of $\langle - \mid - \rangle$
$\Xi_0 \vdash \langle u_k \mid \boldsymbol{t}_0 \rangle : \kappa$	By i.h. (smaller m)

□

Lemma C.18 (Subst. on Substitution)**.**

If $\Theta_0 \vdash \sigma : \Theta$ *and* $\Theta \vdash \sigma' : \Xi$ *then* $\Theta_0 \vdash [\sigma]\sigma' : \Xi$.

Proof. By structural induction on $\Theta \vdash \sigma' : \Xi$. The SubstIx and SubstIxDet cases use Lemma C.17 (Ix. Syntactic Substitution) and the SubstIxDet case uses Lemma C.12 (Value-Det. Substitution). □

Lemma C.19 (Hereditary Associativity)**.**

If $\Xi \vdash t : \tau$ *and* $\Xi; [\tau] \vdash \boldsymbol{t} : \kappa'$ *and* $\Xi; [\kappa'] \vdash \boldsymbol{u} : \kappa$ *then* $\langle t \mid \boldsymbol{t}, \boldsymbol{u} \rangle = \langle \langle t \mid \boldsymbol{t} \rangle \mid \boldsymbol{u} \rangle$.

Proof. By structural induction on $\Xi; [\tau] \vdash \boldsymbol{t} : \kappa'$. Straightforward. □

Lemma C.20 (Ix. Barendregt)**.** *Assume* $\Xi_0 \vdash \sigma : \Xi_1$ *and* $\Xi_1, \Xi_2 \vdash \sigma' : \Xi'$ *and* $dom(\Xi') \cap dom(\Xi_0) = \emptyset$ *and* $dom(\Xi') \neq \emptyset$.

(1) If $\Xi_1, \Xi', \Xi_2 \vdash t : \tau$ *then* $[\sigma][\sigma']t = [[\sigma]\sigma'][\sigma]t$.

(2) If $\Xi_1, \Xi', \Xi_2; [\tau] \vdash \boldsymbol{t} : \kappa$ *then* $[\sigma][\sigma']\boldsymbol{t} = [[\sigma]\sigma'][\sigma]\boldsymbol{t}$.

(3) If $\Xi_1, \Xi_2 \vdash u : \omega$ *and* $\Xi_1, \Xi_2; [\omega] \vdash \boldsymbol{t} : \kappa$ *then* $[\sigma]\langle u \mid \boldsymbol{t}\rangle = \langle [\sigma]u \mid [\sigma]\boldsymbol{t}\rangle$.

Proof. Define $m \in \mathbb{N}$ by

$$m = \sum_{c \in dom(\Xi')} size(\Xi'(c)) + \sum_{\substack{(t'/c) \in \sigma \\ \text{such that } t' \neq c}} size(\Xi_1(c))$$

if we are in part (1) or in part (2), and

$$m = size(\omega) + \sum_{\substack{(t'/c) \in \sigma \\ \text{such that } t' \neq c}} size(\Xi_1(c))$$

if we are in part (3), where *size* measures the structural size of a sort.

We proceed by lexicographic induction,

first, on m (parts (1), (2), and (3));

second, on the part number (of the lemma),

where pt. $(3) <$ pt. (1), pt. $(3) <$ pt. (2), pt. $(1) =$ pt. (2) and pt. $(k) =$ pt. (k);

third, on $t/\boldsymbol{t}$ structure, that is, on the structure of t (pt. (1)) and $\boldsymbol{t}$ (pts. (2) and (3)));

and fourth, on height of the sorting derivation for t in part (1).

Parts (1), (2), and (3) are mutually recursive.

We sometimes show substitution, sorting, and termination reasoning such as that found in part (1), case IxApp, beginning of subcase $a \in dom(\Xi')$, but often elide it.

(1) • **Case**

$$\frac{(a:\tau) \in (\Xi_1, \Xi', \Xi_2)}{\Xi_1, \Xi', \Xi_2 \vdash a:\tau}\ \text{IxVar}$$

– **Case** $a \in dom(\Xi')$

$$\begin{aligned}
[\sigma]([\sigma']t) &= [\sigma]([\sigma']a) && \text{Current case}\\
&= [\sigma](\sigma'(a)) && \text{By def. of } [-]-\\
&= [[\sigma]\sigma']a && \text{By def. of } [-]-\\
&= [[\sigma]\sigma']([\sigma]a) && \text{By def. of } [-]- \ (a \notin dom(\Xi_1))\\
&= [[\sigma]\sigma']([\sigma]t) && \text{Current case}
\end{aligned}$$

– **Case** $a \in dom(\Xi_1)$

$$\begin{aligned}
[\sigma]([\sigma']t) &= [\sigma]([\sigma']a) && \text{Current case}\\
&= [\sigma]a && \text{By def. of } [-]- \ (a \notin dom(\Xi'))\\
&= [[\sigma]\sigma']([\sigma]a) && \text{By def. of } [-]-\\
& && (FV([\sigma]a) \in dom(\Xi_0) \text{ and } dom(\Xi') \cap dom(\Xi_0) = \emptyset)\\
&= [[\sigma]\sigma']([\sigma]t) && \text{Current case}
\end{aligned}$$

– **Case** $a \in dom(\Xi_2)$

$$\begin{aligned}
[\sigma]([\sigma']t) &= [\sigma]([\sigma']a) && \text{Current case}\\
&= [\sigma]a && \text{By def. of } [-]- \ (a \notin dom(\Xi'))\\
&= a && \text{By def. of } [-]- \ (a \notin dom(\Xi_1))\\
&= [[\sigma]\sigma']a && \text{By def. of } [-]- \ (a \notin dom(\Xi'))\\
&= [[\sigma]\sigma']([\sigma]a) && \text{By def. of } [-]- \ (a \notin dom(\Xi_1))\\
&= [[\sigma]\sigma']([\sigma]t) && \text{Current case}
\end{aligned}$$

- **Case**
$$\frac{(a : \tau_a) \in (\Xi_1, \Xi', \Xi_2) \qquad \Xi_1, \Xi', \Xi_2; [\tau_a] \vdash \boldsymbol{t} : \kappa}{\Xi_1, \Xi', \Xi_2 \vdash a(\boldsymbol{t}) : \kappa}\ \text{IxApp}$$

  - **Case** $a \in dom(\Xi')$

$$\Xi_1, \Xi_2 \vdash \sigma'(a) : \Xi'(a) \qquad \text{By inversion}$$

$$(a : \tau_a) \in (\Xi_1, \Xi', \Xi_2) \qquad \text{Premise}$$

$$\tau_a = \Xi'(a) \qquad \text{By inversion}$$

$$\Xi_1, \Xi', \Xi_2; [\Xi'(a)] \vdash \boldsymbol{t} : \kappa \qquad \text{Rewrite subderivation}$$

$$\Xi_1, \Xi_2; [\Xi'(a)] \vdash [\sigma']\boldsymbol{t} : \kappa \qquad \text{By Lemma C.17}$$

$$\Xi_1, \Xi_2 \vdash \langle \sigma'(a) \mid [\sigma']\boldsymbol{t} \rangle : \kappa \qquad \text{By Lemma C.17}$$

$$\Xi_0, \Xi_2 \vdash [\sigma](\sigma'(a)) : \Xi'(a) \qquad \text{By Lemma C.17}$$

$$\Xi_0, \Xi', \Xi_2; [\Xi'(a)] \vdash [\sigma]\boldsymbol{t} : \kappa \qquad \text{By Lemma C.17}$$

$$\Xi_0, \Xi_2 \vdash [\sigma]\sigma' : \Xi' \qquad \text{By Lemma C.18}$$

$$\Xi_0, \Xi_2; [\Xi'(a)] \vdash [[\sigma]\sigma'][\sigma]\boldsymbol{t}) : \kappa \qquad \text{By Lemma C.17}$$

$$\Xi_0, \Xi_2 \vdash \langle [\sigma](\sigma'(a)) \mid [[\sigma]\sigma'][\sigma]\boldsymbol{t}) \rangle : \kappa \qquad \text{By Lemma C.17}$$

$$
\begin{array}{lll}
[\sigma]([\sigma']t) = [\sigma]([\sigma'](a(\boldsymbol{t}))) & & \text{Current case} \\
= [\sigma]\langle\sigma'(a) \mid [\sigma']\boldsymbol{t}\rangle & & \text{By def. of } [-]- \\
= \langle[\sigma](\sigma'(a)) \mid [\sigma]([\sigma']\boldsymbol{t})\rangle & & \text{By i.h. (same/smaller } m\text{; pt. } (3) < (1)) \\
= \langle[\sigma](\sigma'(a)) \mid [[\sigma]\sigma']([\sigma]\boldsymbol{t})\rangle & & \text{By i.h. (same } m\text{; pt. } (2) = (1);\ \boldsymbol{t} < a(\boldsymbol{t})) \\
= [[\sigma]\sigma'](a([\sigma]\boldsymbol{t})) & & \text{By def. of } [-]- \\
= [[\sigma]\sigma']([\sigma](a(\boldsymbol{t}))) & & \text{By def. of } [-]- \ (a \notin \mathit{dom}(\Xi_1)) \\
= [[\sigma]\sigma']([\sigma]t) & & \text{Current case}
\end{array}
$$

– **Case** $a \in \mathit{dom}(\Xi_1)$

$$
\begin{array}{lll}
[\sigma]([\sigma']t) = [\sigma]([\sigma'](a(\boldsymbol{t}))) & & \text{Current case} \\
= [\sigma](a([\sigma']\boldsymbol{t})) & & \text{By def. of } [-]- \ (a \notin \mathit{dom}(\Xi')) \\
= \langle\sigma(a) \mid [\sigma]([\sigma']\boldsymbol{t})\rangle & & \text{By def. of } [-]- \\
= \langle\sigma(a) \mid [[\sigma]\sigma']([\sigma]\boldsymbol{t})\rangle & & \text{By i.h.} \\
 & & \quad (\text{same } m\text{; pt. } (2) = (1);\ \boldsymbol{t} < a(\boldsymbol{t})) \\
= \langle[[\sigma]\sigma'](\sigma(a)) \mid [[\sigma]\sigma']([\sigma]\boldsymbol{t})\rangle & & \text{Because } \mathit{dom}(\Xi_0) \cap \mathit{dom}(\Xi') = \emptyset \\
= [[\sigma]\sigma']\langle\sigma(a) \mid [\sigma]\boldsymbol{t}\rangle & & \text{By i.h.} \\
 & & \quad (\text{same/smaller } m\text{; pt. } (3) < (1)) \\
= [[\sigma]\sigma']([\sigma](a(\boldsymbol{t}))) & & \text{By def. of } [-]- \\
= [[\sigma]\sigma']([\sigma]t) & & \text{Current case}
\end{array}
$$

where the second use of i.h. calls i.h. (part (3)) with $(\Xi_0/\Xi_0, [\sigma]\sigma')$ for σ and $\Xi_1(a)$ for ω and $\sigma(a)$ for u and $[\sigma]\boldsymbol{t}$ for $\boldsymbol{t}$ and (Ξ_0, Ξ_2) for Ξ_0 and

(Ξ_0, Ξ') for Ξ_1 and Ξ_2 for Ξ_2.

– **Case** $a \in dom(\Xi_2)$

$$\begin{aligned}
[\sigma]([\sigma'](a(\boldsymbol{t}))) &= [\sigma](a([\sigma']\boldsymbol{t})) && \text{By def. of } [-]- \\
& && (a \notin dom(\Xi') \text{ by inversion on ctx. WF}) \\
&= a([\sigma]([\sigma']\boldsymbol{t})) && \text{By def. of } [-]- \ (a \notin dom(\Xi_1)) \\
&= a([[\sigma]\sigma']([\sigma]\boldsymbol{t})) && \text{By i.h. (same } m\text{; same part; smaller } t/\boldsymbol{t}) \\
&= [[\sigma]\sigma']([\sigma](a(\boldsymbol{t}))) && \text{By def. of } [-]- \text{ (twice)}
\end{aligned}$$

• **Case**

$$\frac{\Xi_1, \Xi', \Xi_2, a \div \kappa \vdash t_0 : \tau_0}{\Xi_1, \Xi', \Xi_2 \vdash \lambda a.t_0 : \kappa \Rightarrow \tau_0}\ \mathrm{Ix}\lambda$$

$$\begin{aligned}
\Xi_0 \vdash \sigma : \Xi_1 && \text{Premise} \\
\Xi_1, \Xi_2 \vdash \sigma' : \Xi' && \text{Given} \\
\Xi_1, \Xi_2, a \div \kappa \vdash \sigma' : \Xi' && \text{By Lemma C.9 (Ix. Subs. Weakening)}
\end{aligned}$$

$$\Xi_1, \Xi', \Xi_2, a \div \kappa \vdash t_0 : \tau_0 \qquad \text{Subderivation}$$

$$\begin{aligned}
[\sigma][\sigma']t &= [\sigma][\sigma'](\lambda a.t_0) && \text{By equality} \\
&= \lambda a.[\sigma][\sigma']t_0 && \text{By def. of } [-]- \\
&= \lambda a.[[\sigma]\sigma'][\sigma]t_0 && \text{By i.h. (same } m\text{; same part; smaller } t\text{)} \\
&= [[\sigma]\sigma'][\sigma](\lambda a.t_0) && \text{By def. of } [-]- \\
&= [[\sigma]\sigma'][\sigma]t && \text{By equality}
\end{aligned}$$

- **Case** IxSub: Straightforward. This is the only case using the fourth part of the lexicographic induction.
- The remaining cases are straightforward (for all uses of i.h., m is the same, the part numbers are equal in size, and t gets smaller). For the Ix=L, Ix=R, and Ix=LR cases, apply IxVar and use Lemma C.6 (Sorting Weakening) as needed (not needed for Ix=LR case) to apply i.h. while avoiding a similar case analysis to that seen in the IxVar case.

(2) Straightforward (in all uses of i.h., m is the same, the part numbers are equal size according to the induction measure, and $t/\boldsymbol{t}$ gets smaller). We show one case:

- **Case**
$$\frac{\Xi_1, \Xi', \Xi_2 \vdash t_0 : \kappa_0 \qquad \Xi_1, \Xi', \Xi_2; [\tau_0] \vdash \boldsymbol{t}' : \kappa}{\Xi_1, \Xi', \Xi_2; [\kappa_0 \Rightarrow \tau_0] \vdash t_0, \boldsymbol{t}' : \kappa}\ \text{IxSpineEntry}$$

$$\begin{aligned}
[\sigma]([\sigma'](t_0, \boldsymbol{t}')) &= [\sigma]([\sigma']t_0, [\sigma']\boldsymbol{t}') && \text{By def. of } [-]- \\
&= [\sigma]([\sigma']t_0), [\sigma]([\sigma']\boldsymbol{t}') && \text{By def. of } [-]- \\
&= [[\sigma]\sigma']([\sigma]t_0), [\sigma]([\sigma']\boldsymbol{t}') && \text{By i.h.} \\
& && (\text{same } m; \text{ pt. } (1) = (2); t_0 < (t_0, \boldsymbol{t}')) \\
&= [[\sigma]\sigma']([\sigma]t_0), [[\sigma]\sigma']([\sigma]\boldsymbol{t}') && \text{By i.h.} \\
& && (\text{same } m; \text{ pt. } (2) = (2); \boldsymbol{t}' < (t_0, \boldsymbol{t}')) \\
&= [[\sigma]\sigma']([\sigma]t_0, [\sigma]\boldsymbol{t}') && \text{By def. of } [-]- \\
&= [[\sigma]\sigma']([\sigma](t_0, \boldsymbol{t}')) && \text{By def. of } [-]-
\end{aligned}$$

(3) • **Case** $\boldsymbol{t} \neq \cdot$ and u is a variable, c

– **Case** $c \in dom(\Xi_1)$ (hence $c \in dom(\sigma)$)

$$\begin{aligned}
[\sigma]\langle u \mid \boldsymbol{t}\rangle &= [\sigma]\langle c \mid \boldsymbol{t}\rangle && \text{By equality}\\
&= [\sigma](c(\boldsymbol{t})) && \text{By def. of } \langle - \mid -\rangle\\
&= \langle \sigma(c) \mid [\sigma]\boldsymbol{t}\rangle && \text{By def. of } [-]- \ (c \in dom(\sigma))\\
&= \langle [\sigma]c \mid [\sigma]\boldsymbol{t}\rangle && \text{By def. of } [-]-\\
&= \langle [\sigma]u \mid [\sigma]\boldsymbol{t}\rangle && \text{By equality}
\end{aligned}$$

– **Case** $c \in dom(\Xi_2)$ (hence $c \notin dom(\sigma)$)

$$\begin{aligned}
[\sigma]\langle t \mid \boldsymbol{t}\rangle &= [\sigma]\langle c \mid \boldsymbol{t}\rangle && \text{By equality}\\
&= [\sigma](c(\boldsymbol{t})) && \text{By def. of } \langle - \mid -\rangle\\
&= c([\sigma]\boldsymbol{t}) && \text{By def. of } [-]-\\
&= \langle c \mid [\sigma]\boldsymbol{t}\rangle && \text{By def. of } [-]-\\
&= \langle [\sigma]c \mid [\sigma]\boldsymbol{t}\rangle && \text{By def. of } [-]- \ (c \notin dom(\sigma))\\
&= \langle [\sigma]t \mid [\sigma]\boldsymbol{t}\rangle && \text{By equality}
\end{aligned}$$

Because we have considered this case, in the following cases of rule concluding $\Xi_1, \Xi_2; [\omega] \vdash \boldsymbol{t} : \kappa$ (except IxSpineNil), we need not consider the subcase where u is a variable.

- **Case**

$$\frac{}{\Xi_1, \Xi_2; [\kappa] \vdash \cdot : \kappa}\ \text{IxSpineNil}$$

$$
\begin{aligned}
[\sigma]\langle u \mid \boldsymbol{t}\rangle &= [\sigma]\langle u \mid \cdot\rangle && \text{Current case}\\
&= [\sigma]u && \text{By def. of } \langle - \mid -\rangle\\
&= \langle [\sigma]u \mid \cdot\rangle && \text{By def. of } \langle - \mid -\rangle\\
&= \langle [\sigma]u \mid [\sigma]\cdot\rangle && \text{By def. of } [-]-\\
&= \langle [\sigma]u \mid [\sigma]\boldsymbol{t}\rangle && \text{Current case}
\end{aligned}
$$

- **Case**

$$
\frac{\Xi_1, \Xi_2 \vdash t_0 : \kappa_0 \qquad \Xi_1, \Xi_2; [\omega_0] \vdash \boldsymbol{t}' : \kappa}{\Xi_1, \Xi_2; [\kappa_0 \Rightarrow \omega_0] \vdash t_0, \boldsymbol{t}' : \kappa}\ \text{IxSpineEntry}
$$

$$u = \lambda c.\, u_0 \qquad \text{By inversion on Ix}\lambda\ (u \text{ not a variable})$$

$$\Xi_1, \Xi_2, c \div \kappa_0 \vdash u_0 : \omega_0 \qquad ''$$

$$\Xi_1, \Xi_2 \vdash \lambda c.\, u_0 : \kappa_0 \Rightarrow \omega_0 \qquad ''$$

$$\Xi_1, \Xi_2 \vdash t_0 : \kappa_0 \qquad \text{Premise}$$

$$\Xi_1, \Xi_2 \vdash [t_0/c]u_0 : \omega_0 \qquad \text{By Lemma C.17 (Ix. Syntactic Substitution)}$$

with identity substitution reasoning

$$\Xi_1, \Xi_2; [\omega_0] \vdash \boldsymbol{t}' : \kappa \qquad \text{Subderivation}$$

$$\begin{aligned}
[\sigma]\langle u \mid \boldsymbol{t}\rangle &= [\sigma]\langle \lambda c.\, u_0 \mid t_0, \boldsymbol{t}'\rangle && \text{Current case}\\
&= [\sigma]\langle [t_0/c]u_0 \mid \boldsymbol{t}'\rangle && \text{By def. of } \langle - \mid -\rangle\\
&= \langle [\sigma]([t_0/c]u_0) \mid [\sigma]\boldsymbol{t}'\rangle && \text{By i.h. (smaller } m\text{)}\\
&= \langle [[\sigma]t_0/c]([\sigma]u_0) \mid [\sigma]\boldsymbol{t}'\rangle && \text{By i.h. (smaller } m\text{)}\\
& && (c \notin \mathit{dom}(\Xi_0) \text{ by variable convention})\\
& && (\Xi_1, c \div \kappa_0, \Xi_2 \vdash u_0 : \omega_0 \text{ as } \overline{\Xi_2} = \Xi_2)\\
&= \langle \lambda c.\, [\sigma]u_0 \mid [\sigma]t_0, [\sigma]\boldsymbol{t}'\rangle && \text{By def. of } \langle - \mid -\rangle\\
&= \langle [\sigma](\lambda c.\, u_0) \mid [\sigma](t_0, \boldsymbol{t}')\rangle && \text{By def. of } [-]-\\
&= \langle [\sigma]u \mid [\sigma]\boldsymbol{t}\rangle && \text{Current case}
\end{aligned}$$

- **Case**

$$\frac{\Xi_1, \Xi_2; [\omega_k] \vdash \boldsymbol{t}' : \kappa}{\Xi_1, \Xi_2; [\omega_1 \times \omega_2] \vdash .k, \boldsymbol{t}' : \kappa}\ \text{IxSpineProj}_k$$

– **Case**

$$\frac{(a:\tau)\in(\Xi_1,\Xi_2)\qquad \Xi_1,\Xi_2;[\tau]\vdash \boldsymbol{u}:\omega_1\times\omega_2}{\Xi_1,\Xi_2\vdash a(\boldsymbol{u}):\omega_1\times\omega_2}\ \text{IxApp}$$

$$\begin{aligned}
[\sigma]\langle u \mid \boldsymbol{t}\rangle &= [\sigma]\langle a(\boldsymbol{u}) \mid \boldsymbol{t}\rangle && \text{By equalities}\\
&= [\sigma](a(\boldsymbol{u},\boldsymbol{t})) && \text{By def. of } \langle - \mid -\rangle\\
&= \langle \boldsymbol{\sigma}(a) \mid [\sigma]\boldsymbol{u},[\sigma]\boldsymbol{t}\rangle && \text{By def. of } [-]-\\
&= \langle\langle \boldsymbol{\sigma}(a) \mid [\sigma]\boldsymbol{u}\rangle \mid [\sigma]\boldsymbol{t}\rangle && \text{By Lemma C.19 (Hereditary Associativity)}\\
& && \quad \text{with Lemma C.17 (Ix. Syntactic Substitution)}\\
& && \quad \text{and Lemma C.6 (Sorting Weakening)}\\
&= \langle [\sigma](a(\boldsymbol{u})) \mid [\sigma]\boldsymbol{t}\rangle && \text{By def. of } [-]-\\
&= \langle [\sigma]u \mid [\sigma]\boldsymbol{t}\rangle && \text{By equalities}
\end{aligned}$$

– **Case**

$$\frac{\Xi_1,\Xi_2\vdash u_1:\omega_1\qquad \Xi_1,\Xi_2\vdash u_2:\omega_2}{\Xi_1,\Xi_2\vdash (u_1,u_2):\omega_1\times\omega_2}\ \text{Ix}\times$$

$$\begin{aligned}
[\sigma]\langle u \mid \boldsymbol{t}\rangle &= [\sigma]\langle (u_1,u_2) \mid .k,\boldsymbol{t}'\rangle && \text{By equalities}\\
&= [\sigma]\langle u_k \mid \boldsymbol{t}'\rangle && \text{By def. of } \langle - \mid -\rangle\\
&= \langle [\sigma]u_k \mid [\sigma]\boldsymbol{t}'\rangle && \text{By i.h. (smaller } m)\\
&= \langle ([\sigma]u_1,[\sigma]u_2) \mid .k,[\sigma]\boldsymbol{t}'\rangle && \text{By def. of } \langle - \mid -\rangle\\
&= \langle [\sigma](u_1,u_2) \mid [\sigma](.k,\boldsymbol{t}')\rangle && \text{By def. of } [-]-\\
&= \langle [\sigma]u \mid [\sigma]\boldsymbol{t}\rangle && \text{By equalities} \qquad \square
\end{aligned}$$

Lemma C.21 (Sorting Soundness)**.**

(1) If $\Xi\vdash t:\tau\ [\xi_t]$ *and* $\vdash\delta:\Xi$ *then* $[\![t]\!]_\delta\in[\![\tau]\!]$.

(2) If $\Xi;[\tau] \vdash \boldsymbol{t} : \kappa$ *and* $\vdash \delta : \Xi$ *then* $[\![\boldsymbol{t}]\!]_\delta : [\![\tau]\!] \to [\![\kappa]\!]$.

Proof. By mutual induction on the structure of the given sorting derivation. □

Lemma C.22 (Ix. Subst. Typing Sound (No Prop))**.**

If $\Xi_0;\cdot \vdash \sigma : \Xi;\cdot$ *and* $\vdash \delta : \Xi_0;\cdot$ *then* $\vdash [\![\sigma]\!]_\delta : \Xi;\cdot$

Proof. By structural induction on the given substitution typing derivation.

- **Case**

$$\frac{}{\Xi_0;\cdot \vdash \cdot : \cdot;\cdot}\ \text{SubstEmpty}$$

$[\![\cdot]\!]_\delta = \cdot$ By def.

$\vdash \cdot : \cdot;\cdot$ By Emptyδ

- **Case**

$$\frac{\Xi_0;\cdot \vdash \sigma' : \Xi';\cdot \qquad \Xi_0 \vdash t : \tau \qquad a \notin dom(\Xi)}{\Xi_0;\cdot \vdash \sigma', t/a : \Xi, a \div \tau;\cdot}\ \text{SubstIx}$$

$\Xi_0;\cdot \vdash \sigma' : \Xi';\cdot$ Subderivation

$\vdash [\![\sigma']\!]_\delta : \Xi';\cdot$ By i.h.

$\Xi_0 \vdash t : \tau$ Premise

$[\![t]\!]_\delta \in [\![\tau]\!]$ By Lemma C.21 (Sorting Soundness)

$\vdash \underbrace{[\![\sigma']\!]_\delta, [\![t]\!]_\delta / a}_{[\![\sigma]\!]_\delta} : \underbrace{\Xi', a \div \tau}_{\Xi};\cdot$ By Ixδ

- **Case** SubstIxDet: Similar to SubstIx case. Use Lemma C.7 (Ix. $[\![-]\!]$ Weak. Invariant).

- **Case** SubstProp: Impossible.

- **Case** SubstVal: Impossible (because $\Gamma = \cdot$). □

Lemma C.23 (Compose Ix. Spine)**.**

If $\Xi; [\tau] \vdash \boldsymbol{t} : \kappa'$ *and* $\Xi; [\kappa'] \vdash \boldsymbol{u} : \kappa$ *and* $\vdash \delta : \Xi$

then $[\![\boldsymbol{t}, \boldsymbol{u}]\!]_\delta = [\![\boldsymbol{u}]\!]_\delta \circ [\![\boldsymbol{t}]\!]_\delta$.

Proof. By structural induction on $\Xi; [\tau] \vdash \boldsymbol{t} : \kappa'$. □

Lemma C.24 (Index Substitution Soundness)**.** *Assume* $\Xi_0; \cdot \vdash \sigma : \Xi; \cdot$ *and* $\vdash \delta : \Xi_0; \cdot$.

(1) If $\Xi \vdash t : \tau\ [\xi_t]$ *then* $[\![[\sigma]t]\!]_\delta = [\![t]\!]_{[\![\sigma]\!]_\delta}$ *(for any derivation* $\Xi_0 \vdash [\sigma]t : \tau\ [{}^{\mathsf{d}}[\sigma]\xi_t]$*).*

(2) If $\Xi; [\tau] \vdash \boldsymbol{t} : \kappa$ *then* $[\![[\sigma]\boldsymbol{t}]\!]_\delta = [\![\boldsymbol{t}]\!]_{[\![\sigma]\!]_\delta}$ *(for any derivation* $\Xi_0; [\tau] \vdash [\sigma]\boldsymbol{t} : \kappa$*).*

(3) If $\Xi_0 \vdash u : \omega$ *and* $\Xi_0; [\omega] \vdash \boldsymbol{t} : \kappa$ *then* $[\![\langle u \mid \boldsymbol{t} \rangle]\!]_\delta = [\![\boldsymbol{t}]\!]_\delta\, [\![u]\!]_\delta$.

Proof. Define $m \in \mathbb{N}$ by

$$m = \sum_{\substack{(t'/c) \in \sigma \\ \text{such that } t' \neq c}} \mathit{size}(\Xi(c))$$

if we are in part (1) or part (2), and $m = \mathit{size}(\omega)$ if we are in part (3), where *size* measures the structural size of a sort. Proceed by lexicographic induction, first, on m, second, on the part size, where part (3) is smaller than parts (1) and (2) and otherwise parts are equal in size, third, on the structure of $t/\boldsymbol{t}$, and fourth, on the height of the sorting derivation for t in part (1).

By Lemma C.2 (Filter Out Propositions), $\Xi_0; \cdot \vdash \sigma : \Xi; \cdot$. By Lemma C.22 (Ix. Subst. Typing Sound (No Prop)), $\vdash [\![\sigma]\!]_\delta : \Xi$, so we can apply $[\![t]\!]$ and $[\![\boldsymbol{t}]\!]$ to $[\![\sigma]\!]_\delta$.

(1) • **Case**

$$\frac{(a : \tau) \in \Xi}{\Xi \vdash a : \tau\ [\cdot]}\ \text{IxVar}$$

$$\begin{aligned}
[\![[\sigma]t]\!]_\delta &= [\![[\sigma]a]\!]_\delta && \text{By equality}\\
&= [\![\sigma(a)]\!]_\delta && \text{By def. of } [-]-\\
&= [\![\sigma]\!]_\delta(a) && \text{By def. of } [\![-]\!] \text{ (Def. C.2) and lookup}\\
&= [\![a]\!]_{[\![\sigma]\!]_\delta} && \text{By def. of } [\![-]\!]\\
&= [\![t]\!]_{[\![\sigma]\!]_\delta} && \text{By equality}
\end{aligned}$$

• **Case**

$$\frac{\Xi, a \div \kappa \vdash t_0 : \tau_0\ [\xi_t]}{\Xi \vdash \lambda a.t_0 : \kappa \Rightarrow \tau_0\ [\cdot]}\ \text{Ix}\lambda$$

$$\begin{aligned}
[\![[\sigma]t]\!]_\delta &= [\![[\sigma](\lambda a.t_0)]\!]_\delta && \text{By equality}\\
&= [\![\lambda a.[\sigma]t_0]\!]_\delta && \text{By def. of } [-]-\\
&= d \mapsto [\![[\sigma]t_0]\!]_{\delta,d/a} && \text{By def. of } [\![-]\!]\\
&= d \mapsto [\![[\sigma, a/a]t_0]\!]_{\delta,d/a} && \text{Identity subst.}\\
&= d \mapsto [\![t_0]\!]_{[\![\sigma,a/a]\!]_{\delta,d/a}} && \text{By i.h. (same } m\text{; same part; smaller } t\text{)}\\
& && \quad \text{with Lemma C.11 (Ix. Id. Subs. Extension)}\\
&= d \mapsto [\![t_0]\!]_{[\![\sigma]\!]_{\delta,d/a},[\![a]\!]_{\delta,d/a}/a} && \text{By def. of } [\![-]\!]\\
&= d \mapsto [\![t_0]\!]_{[\![\sigma]\!]_{\delta,d/a},d/a} && \text{By def. of } [\![-]\!] \text{ and lookup}\\
&= d \mapsto [\![t_0]\!]_{[\![\sigma]\!]_\delta,d/a} && \text{By Lemma C.10}\\
& && \quad \text{with Lemma C.9 (Ix. Subs. Weakening)}\\
&= [\![\lambda a.t_0]\!]_{[\![\sigma]\!]_\delta} && \text{By def. of } [\![-]\!]\\
&= [\![t]\!]_{[\![\sigma]\!]_\delta} && \text{By equality}
\end{aligned}$$

- **Case**

$$\frac{(a : \omega) \in \Xi \qquad \Xi; [\omega] \vdash u : \kappa}{\Xi \vdash a(u) : \kappa\,[\cdot]}\ \text{IxApp}$$

$$\begin{aligned}
[\![[\sigma]t]\!]_\delta &= [\![[\sigma](a(u))]\!]_\delta && \text{By equality}\\
&= [\![\langle \sigma(a) \mid [\sigma]u\rangle]\!]_\delta && \text{By def. of } [-]-\\
& && \quad \text{with Lemma C.17 (Ix. Syntactic Substitution), pt. (3)}\\
&= [\![[\sigma]u]\!]_\delta\, [\![\sigma(a)]\!]_\delta && \text{By i.h. (same or smaller } m\text{; smaller part)}\\
&= [\![u]\!]_{[\![\sigma]\!]_\delta}\, [\![\sigma(a)]\!]_\delta && \text{By i.h. (same } m\text{; same part size; smaller } t/\boldsymbol{t})\\
&= [\![u]\!]_{[\![\sigma]\!]_\delta}\, ([\![\sigma]\!]_\delta\,(a)) && \text{By def.}\\
&= [\![a(u)]\!]_{[\![\sigma]\!]_\delta} && \text{By def.}\\
&= [\![t]\!]_{[\![\sigma]\!]_\delta} && \text{By equality}
\end{aligned}$$

- **Case** IxSub: Straightforward but this is the only case that uses the fourth part of the lexicographic induction.
- The remaining cases are straightforward (for all uses of i.h., m is the same, the part is the same, and t is smaller).

(2) Straightforward (for all uses of i.h., m is the same, the part size is the same, and $t/\boldsymbol{t}$ is smaller).

(3) By Lemma C.17 (Ix. Syntactic Substitution), $\Xi \vdash \langle u \mid \boldsymbol{t}\rangle : \kappa$. We first consider the case where $\boldsymbol{t} \neq \cdot$ and u is a variable. We then case analyze $\Xi_0; [\omega] \vdash \boldsymbol{t} : \kappa$ and may assume u is not a variable when $\boldsymbol{t} \neq \cdot$.

- **Case** $\boldsymbol{t} \neq \cdot$ and u is a variable

$$\begin{aligned}
[\![\langle u \mid \boldsymbol{t} \rangle]\!]_\delta &= [\![u(\boldsymbol{t})]\!]_\delta && \text{By def. of } \langle - \mid - \rangle \\
&= [\![\boldsymbol{t}]\!]_\delta \, [\![u]\!]_\delta && \text{By def. of } [\![-]\!]
\end{aligned}$$

- **Case**

$$\frac{}{\Xi_0; [\kappa] \vdash \cdot : \kappa}\ \text{IxSpineNil}$$

$$\begin{aligned}
[\![\langle u \mid \boldsymbol{t} \rangle]\!]_\delta &= [\![\langle u \mid \cdot \rangle]\!]_\delta && \text{By equality} \\
&= [\![u]\!]_\delta && \text{By def. of } \langle - \mid - \rangle \\
&= id_{[\![\kappa]\!]} \, [\![u]\!]_\delta && \text{By def. of } id_{[\![\kappa]\!]} \\
&= [\![\cdot]\!]_\delta \, [\![u]\!]_\delta && \text{By def. of } [\![-]\!] \\
&= [\![\boldsymbol{t}]\!]_\delta \, [\![u]\!]_\delta && \text{By equality}
\end{aligned}$$

- **Case**

$$\frac{\Xi_0 \vdash t_0 : \kappa_0 \qquad \Xi_0; [\omega_0] \vdash \boldsymbol{t}' : \kappa}{\Xi_0; [\kappa_0 \Rightarrow \omega_0] \vdash t_0, \boldsymbol{t}' : \kappa}\ \text{IxSpineEntry}$$

$u = \lambda a.\, u_0$ By inversion on given sorting derivations

(u not a variable)

$$
\begin{aligned}
[\![\langle \lambda a.u_0 \mid t_0, \boldsymbol{t}' \rangle]\!]_\delta &= [\![\langle [t_0/a]u_0 \mid \boldsymbol{t}' \rangle]\!]_\delta && \text{By def. of } \langle - \mid - \rangle \\
&= [\![\langle [id_\Xi, t_0/a]u_0 \mid \boldsymbol{t}' \rangle]\!]_\delta && \text{Identity substitution} \\
& && \quad \text{(use weakening for typing)} \\
&= [\![\boldsymbol{t}']\!]_\delta \, [\![[id_\Xi, t_0/a]u_0]\!]_\delta && \text{By i.h. (smaller } m\text{)} \\
& && \quad \text{with Lemma C.17} \\
&= [\![\boldsymbol{t}']\!]_\delta \, [\![[t_0/a]u_0]\!]_\delta && \text{Identity subst.} \\
&= [\![\boldsymbol{t}']\!]_\delta \, [\![u_0]\!]_{\delta, [\![t_0]\!]_\delta/a} && \text{By i.h. (smaller } m\text{)} \\
&= [\![\boldsymbol{t}']\!]_\delta \, [\![u_0]\!]_{\delta, [\![t_0]\!]_\delta/a} && \text{By def. of } [\![-]\!] \\
&= [\![\boldsymbol{t}']\!]_\delta \, ([\![\lambda a.u_0]\!]_\delta \, [\![t_0]\!]_\delta) && \text{By def. of } [\![-]\!] \\
&= [\![t_0, \boldsymbol{t}']\!]_\delta \, [\![\lambda a.u_0]\!]_\delta && \text{By def. of } [\![-]\!]
\end{aligned}
$$

- **Case**

$$\frac{\Xi_0; [\omega_k] \vdash \boldsymbol{t}' : \kappa}{\Xi_0; [\omega_1 \times \omega_2] \vdash .k, \boldsymbol{t}' : \kappa} \, \mathrm{IxSpineProj}_k$$

  - **Case**

$$\frac{\Xi_0 \vdash u_1 : \omega_1 \qquad \Xi_0 \vdash u_2 : \omega_2}{\Xi_0 \vdash (u_1, u_2) : \omega_1 \times \omega_2} \, \mathrm{Ix}\times$$

$$
\begin{aligned}
[\![\langle u \mid \boldsymbol{t} \rangle]\!]_\delta &= [\![\langle (u_1, u_2) \mid .k, \boldsymbol{t}' \rangle]\!]_\delta && \text{By equalities} \\
&= [\![\langle u_k \mid \boldsymbol{t}' \rangle]\!]_\delta && \text{By def. of } \langle - \mid - \rangle \\
&= [\![\boldsymbol{t}']\!]_\delta \, [\![u_k]\!]_\delta && \text{By i.h. (smaller } m\text{)} \\
&= [\![\boldsymbol{t}']\!]_\delta \, (\pi_k \, [\![(u_1, u_2)]\!]_\delta) && \text{By def. of } [\![-]\!] \text{ and } \pi_k \\
&= [\![.k, \boldsymbol{t}']\!]_\delta \, [\![(u_1, u_2)]\!]_\delta && \text{By def. of } [\![-]\!] \\
&= [\![\boldsymbol{t}]\!]_\delta \, [\![u]\!]_\delta && \text{By equalities}
\end{aligned}
$$

– **Case**
$$\frac{(a : \omega) \in \Xi_0 \qquad \Xi_0; [\omega] \vdash u : \omega}{\Xi_0 \vdash a(u) : \omega\ [\cdot]}\ \text{IxApp}$$

$$\begin{aligned}
[\![\langle u \mid t \rangle]\!]_\delta &= [\![\langle a(u) \mid t \rangle]\!]_\delta && \text{By equality} \\
&= [\![a(u,t)]\!]_\delta && \text{By def. of } \langle - \mid - \rangle \\
&= [\![u,t]\!]_\delta\, \delta(a) && \text{By def. of } [\![-]\!] \\
&= [\![t]\!]_\delta\, ([\![u]\!]_\delta\, \delta(a)) && \text{By Lemma C.23 (Compose Ix. Spine)} \\
&= [\![t]\!]_\delta\, [\![a(u)]\!]_\delta && \text{By def. of } [\![-]\!] \\
&= [\![t]\!]_\delta\, [\![u]\!]_\delta && \text{By equality}
\end{aligned}$$

□

Lemma C.25 (Index Subst. Typing Sound)**.**

If $\Theta_0 \vdash \sigma : \Theta$ *and* $\vdash \delta : \Theta_0$ *then* $\vdash [\![\sigma]\!]_\delta : \Theta$.

Proof. By structural induction on the derivation $\Theta_0 \vdash \sigma : \Theta$.

- **Case** SubstEmpty: Similar to SubstEmpty case of Lemma C.22.
- **Case** SubstIx: Similar to SubstIx case of Lemma C.22.
- **Case** SubstIxDet: Similar to SubstIx case of Lemma C.22. Use Lemma C.7 (Ix. $[\![-]\!]$ Weak. Invariant).
- **Case**
$$\frac{\Theta_0 \vdash \sigma : \Theta' \qquad \Theta_0 \vdash [\sigma]\varphi\ \mathsf{true}}{\Theta_0 \vdash \sigma : \Theta', \varphi}\ \text{SubstProp}$$

$\vdash \delta : \Theta_0$ Given

$\Theta_0 \vdash \sigma : \Theta'$ Subderivation

$\vdash [\![\sigma]\!]_\delta : \Theta'$ By i.h.

$\Theta_0 \vdash [\sigma]\varphi\ \mathsf{true}$ Premise

$\Theta_0 \vdash [\sigma]\varphi : \mathbb{B}$ Presupposed derivation

$[\![\varphi]\!]_{[\![\sigma]\!]_\delta} = [\![[\sigma]\varphi]\!]_\delta$ By Lemma C.24 (Index Substitution Soundness)

$= \{\bullet\}$ By inversion on PropTrue

$\vdash [\![\sigma]\!]_\delta : \Theta', \varphi$ By Propδ

- **Case** SubstVal: Impossible. □

Lemma C.26 (Sem. Subs. Entry)**.**

(1) If $\vdash \delta_1 : \Theta_1$ *and* $\vdash \delta_1, \delta_2 : \Theta_1, \Theta_2$ *and* $d \in [\![\tau]\!]$ *and* $a \notin dom(\Theta_1, \Theta_2)$
then $\vdash \delta_1, d/a, \delta_2 : \Theta_1, a : \tau, \Theta_2$.

(2) If $\vdash \delta_1 : \Theta_1$ *and* $\vdash \delta_1, \delta_2 : \Theta_1, \Theta_2$ *and* $\Theta_1 \vdash \varphi : \mathbb{B}$ *and* $[\![\varphi]\!]_{\delta_1} = \{\bullet\}$
then $\vdash \delta_1, \delta_2 : \Theta_1, \varphi, \Theta_2$.

Proof. Each part is proved by structural induction on Θ_2. In part (1), use Lemma C.7 (Ix. $[\![-]\!]$ Weak. Invariant) in the $\Theta_2 = \Theta_2', \psi$ case. □

Lemma C.27 (Prop. Truth Equiv. Relation)**.**

(1) If $\Xi \vdash t : \kappa$ *then* $\Xi \vdash t = t\ \mathsf{true}$.

(2) If $\Theta \vdash t_1 = t_2\ \mathsf{true}$ *then* $\Theta \vdash t_2 = t_1\ \mathsf{true}$.

(3) If $\Theta \vdash t_1 = t_2$ true *and* $\Theta \vdash t_2 = t_3$ true *then* $\Theta \vdash t_1 = t_3$ true.

Proof. (1) Assume $\vdash \delta : \Xi$.

$[\![t]\!]_\delta \in [\![\kappa]\!]$	By Lemma C.21 (Sorting Soundness)
$[\![t]\!]_\delta = [\![t]\!]_\delta$	Equality on $[\![\kappa]\!]$ is reflexive
$\Xi \vdash t = t$ true	By PropTrue

(2) Assume $\vdash \delta : \Theta$. We are given $\Theta \vdash t_1 = t_2$ true, which presupposes $\Theta \vdash t_1 = t_2 : \mathbb{B}$. By inversion on the index sorting equality rules and by Lemma C.21 (Sorting Soundness) we know $[\![t_1]\!]_\delta$ and $[\![t_2]\!]_\delta$ are elements of $[\![\kappa]\!]$ for some κ.

$[\![t_1 = t_2]\!]_\delta = \{\bullet\}$	By inversion on PropTrue
$[\![t_1]\!]_\delta = [\![t_2]\!]_\delta$	By def. of $[\![-]\!]$
$[\![t_2]\!]_\delta = [\![t_1]\!]_\delta$	Equality on $[\![\kappa]\!]$ is symmetric
$[\![t_2 = t_1]\!]_\delta = \{\bullet\}$	By def. of $[\![-]\!]$
$\Theta \vdash t_2 = t_1$ true	By PropTrue

(3) Similar to part (2). □

Lemma C.28 (Assumption)**.** *If* $(\Theta_1, \varphi, \Theta_2)$ ctx *then* $\Theta_1, \varphi, \Theta_2 \vdash \varphi$ true.

Proof. Suppose $\vdash \delta : \Theta_1, \varphi, \Theta_2$. By Lemma C.4 (Stratify Sem. Subs), $\delta = \delta_1, \delta_2$ and $\vdash \delta_1 : \Theta_1, \varphi$. By inversion on Prop$\delta$, $[\![\varphi]\!]_{\delta_1} = \{\bullet\}$. By Lemma C.7 (Ix. $[\![-]\!]$ Weak. Invariant), $[\![\varphi]\!]_\delta = \{\bullet\}$. By PropTrue, $\Theta_1, \varphi, \Theta_2 \vdash \varphi$ true. □

Lemma C.29 (Consequence)**.**

If $\Theta_1, \psi, \Theta_2 \vdash \varphi$ true *and* $\Theta_1 \vdash \psi$ true *then* $\Theta_1, \Theta_2 \vdash \varphi$ true.

Proof. By inversion on $\Theta_1, \psi, \Theta_2 \vdash \varphi$ true, for all $\vdash \delta : \Theta_1, \psi, \Theta_2$, we have $[\![\varphi]\!]_\delta = \{\bullet\}$. Suppose $\vdash \delta : \Theta_1, \Theta_2$. By Lemma C.4 (Stratify Sem. Subs), there exist δ_1, δ_2 such that

$\delta = \delta_1, \delta_2$ and $\vdash \delta_1 : \Theta_1$ and $\vdash \delta_1, \delta_2 : \Theta_1, \Theta_2$; these together with the given $\Theta_1 \vdash \psi$ true yields $\vdash \delta : \Theta_1, \psi, \Theta_2$ by Lemma C.26 (Sem. Subs. Entry). Eliminating the above "for all", we have $[\![\varphi]\!]_\delta = \{\bullet\}$. By PropTrue, $\Theta_1, \Theta_2 \vdash \varphi$ true. □

Lemma C.30 (Prop. Truth Syn. Subs.)**.**

If $\Theta_0 \vdash \sigma : \Theta$ *and* $\Theta \vdash \varphi$ true *then* $\Theta_0 \vdash [\sigma]\varphi$ true.

Proof. Only one rule can conclude $\Theta \vdash \varphi$ true:

- **Case**
$$\frac{\text{for all } \delta, \text{ if } \vdash \delta : \Theta \text{ then } [\![\varphi]\!]_\delta = \{\bullet\}}{\Theta \vdash \varphi \text{ true}}\ \text{PropTrue}$$

$\vdash \delta : \Theta_0$ Suppose

$\Theta_0 \vdash \sigma : \Theta$ Given

$\vdash [\![\sigma]\!]_\delta : \Theta$ By Lemma C.25 (Index Subst. Typing Sound)

$[\![[\sigma]\varphi]\!]_\delta = [\![\varphi]\!]_{[\![\sigma]\!]_\delta}$ By Lemma C.24 (Index Substitution Soundness)

$= \{\bullet\}$ By premise (for all δ, if $\vdash \delta : \Theta$ then $[\![\varphi]\!]_\delta = \{\bullet\}$)

$\Theta_0 \vdash [\sigma]\varphi$ true By PropTrue □

Lemma C.31 (Subst. Inconsistent)**.** *If* $\Theta_0 \vdash \sigma : \Theta$ *and* $\Theta \vdash \mathsf{ff}$ true *then* $\Theta_0 \vdash \mathsf{ff}$ true.

Proof. Follows from Lemma C.30 (Prop. Truth Syn. Subs) and the definition of substitution. □

Lemma C.32 (Id. Subst. Typing)**.** *For all* Θ ctx*, we have* $\Theta; \cdot \vdash id_{\Theta;\cdot} : \Theta; \cdot$.

Proof. By structural induction on Θ ctx. Use Lemma C.9 (Ix. Subs. Weakening), Lemma C.13 (Ix. Id. Subst) and Lemma C.28 (Assumption). □

Lemma C.33 (Ix. Equiv. Sound)**.** *Assume* $\vdash \delta : \Theta$.

(1) If $\Theta \vdash u \equiv t : \tau$ *then* $[\![u]\!]_\delta = [\![t]\!]_\delta$.

(2) If $\Theta; [\tau] \vdash \boldsymbol{u} \equiv \boldsymbol{t} : \kappa$ *then* $[\![\boldsymbol{u}]\!]_\delta = [\![\boldsymbol{t}]\!]_\delta$.

Proof. By mutual induction on the structure of the given equivalence derivation. □

Lemma C.34 (Ix. Equiv. Reflexive)**.**

(1) If $\Xi \vdash t : \tau\ [\xi_t]$ *then* $\Xi \vdash t \equiv t : \tau$.

(2) If $\Xi; [\tau] \vdash \boldsymbol{t} : \kappa$ *then* $\Xi; [\tau] \vdash \boldsymbol{t} \equiv \boldsymbol{t} : \kappa$.

Proof. By mutual induction on the structure of the given sorting derivation. □

Lemma C.35 (Ix. Equiv. Transitive)**.**

(1) If $\Theta \vdash t_1 \equiv t_2 : \tau$ *and* $\Theta \vdash t_2 \equiv t_3 : \tau$ *then* $\Theta \vdash t_1 \equiv t_3 : \tau$.

(2) If $\Theta; [\tau] \vdash \boldsymbol{t} \equiv \boldsymbol{t}' : \kappa$ *and* $\Theta; [\tau] \vdash \boldsymbol{t}' \equiv \boldsymbol{u} : \kappa$ *then* $\Theta; [\tau] \vdash \boldsymbol{t} \equiv \boldsymbol{u} : \kappa$.

Proof. By mutual induction on the structure of the given equivalence derivations. Use Lemma C.27 (Prop. Truth Equiv. Relation) as needed. □

Lemma C.36 (Ix. Equiv. Weakening)**.** *If* $\Theta \vdash u \equiv t : \tau$ *and* $\Theta \subseteq \Theta'$
then $\Theta' \vdash u \equiv t : \tau$ *by a derivation of equal structure.*

Proof. By structural induction on $\Theta \vdash u \equiv t : \tau$. The Ix≡SMT case uses Lemma C.8 (Prop. Truth Weakening). □

Lemma C.37 (Ix. App. Respects Equivalence)**.**
If $\Theta \vdash u_1 \equiv u_2 : \tau$ *and* $\Theta; [\tau] \vdash \boldsymbol{t}_1 \equiv \boldsymbol{t}_2 : \kappa$ *then* $\Theta \vdash \langle u_1 \mid \boldsymbol{t}_1 \rangle = \langle u_2 \mid \boldsymbol{t}_2 \rangle$ true.

Proof. Follows straightforwardly from Lemma C.17 (Ix. Syntactic Substitution), Lemma C.33 (Ix. Equiv. Sound), and Lemma C.24 (Index Substitution Soundness) (part (3)). □

Lemma C.38 (Type/Functor Barendregt)**.** *Assume* $\Xi_0 \vdash \sigma : \Xi_1$ *and* $\Xi_1, \Xi_2 \vdash \sigma' : \Xi'$ *and* $dom(\Xi_0) \cap dom(\Xi') = \emptyset$ *and* $dom(\Xi') \neq \emptyset$.

(1) If $\Xi_1, \Xi', \Xi_2 \vdash A\ \mathsf{type}[\xi]$ *then* $[\sigma][\sigma']A = [[\sigma]\sigma'][\sigma]A$.

(2) If $\Xi_1, \Xi', \Xi_2 \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi]$ *then* $[\sigma][\sigma'](\mathscr{M}(F)) = [[\sigma]\sigma'][\sigma](\mathscr{M}(F))$.

(3) If $\Xi_1, \Xi', \Xi_2 \vdash \mathscr{F}\ \mathsf{functor}[\xi]$ *then* $[\sigma][\sigma']\mathscr{F} = [[\sigma]\sigma'][\sigma]\mathscr{F}$.

Proof. Part (1) by structural induction on the given type well-formedness derivation. The DeclTp$\wedge$ and DeclTp$\supset$ cases of part (1) use (repeated) Lemma C.20 (Ix. Barendregt). Use Lemma C.9 (Ix. Subs. Weakening) in the DeclTp$\exists$ and DeclTp$\forall$ cases.

Part (2) by structural induction. in order to use Lemma C.20 (Ix. Barendregt), weaken the latter using Lemma C.6 (Sorting Weakening) with $\overset{\mathsf{d}}{\div}(\Xi_1, \Xi', \Xi_2) \subseteq (\Xi_1, \Xi', \Xi_2)$. Also use the fact that the algebra and functor are closed (because its input Ξ is empty).

Part (3) by structural induction, using part (1). □

Lemma C.39 (Algebra Barendregt)**.** *Assume* $\Xi_0 \vdash \sigma : \Xi_1$ *and* $\Xi_1, \Xi_2 \vdash \sigma' : \Xi'$ *and* $dom(\Xi_0) \cap dom(\Xi') = \emptyset$ *and* $dom(\Xi') \neq \emptyset$.
If $\Xi_1, \Xi', \Xi_2 \vdash q \Rightarrow t : \hat{P}(\tau) \Rightarrow \tau$ *then* $[\sigma][\sigma'](q \Rightarrow t) = [[\sigma]\sigma'][\sigma](q \Rightarrow t)$.

Proof. By structural induction on $\Xi_1, \Xi', \Xi_2 \vdash q \Rightarrow t : \hat{P}(\tau) \Rightarrow \tau$. The DeclAlg$\oplus$ case is impossible. The DeclAlg$I$ case uses Lemma C.20 (Ix. Barendregt). The DeclAlgId case uses Lemma C.11 (Ix. Id. Subs. Extension). The DeclAlg$\exists$ case uses (repeated) Lemma C.11 (Ix. Id. Subs. Extension). The DeclAlgConst case uses Lemma C.38 (Type/Functor Barendregt). □

Lemma C.40 (Subst. Algebra Pattern)**.**

If $\alpha \circ \mathsf{inj}_k \mathrel{\mathring{=}} \alpha_k$ *then* $[\sigma]\alpha \circ \mathsf{inj}_k \mathrel{\mathring{=}} [\sigma]\alpha_k$ *for all* σ.

Proof. By structural induction on $\alpha \circ \mathsf{inj}_k \mathrel{\mathring{=}} \alpha_k$. □

Appendix C.1 Structural Properties

Lemma C.41 (Ix.-Level Weakening)**.** *Assume* $\Xi \subseteq \Xi'$ *for parts mentioning* Ξ*, and assume* $\Theta \subseteq \Theta'$ *for the other parts, those mentioning* Θ.

(1) If $\mathscr{D} :: \Xi \vdash A\ \mathsf{type}[\xi]$ *then* $\Xi' \vdash A\ \mathsf{type}[\xi]$.

(2) If $\mathscr{D} :: \Xi \vdash \mathscr{F}\ \mathsf{functor}[\xi]$ *then* $\Xi' \vdash \mathscr{F}\ \mathsf{functor}[\xi]$.

(3) If $\mathscr{D} :: \Xi \vdash \alpha : F(\tau) \Rightarrow \tau$ *then* $\Xi' \vdash \alpha : F(\tau) \Rightarrow \tau$.

(4) If $\mathscr{D} :: \Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi]$ *then* $\Xi' \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi]$.

(5) If $\mathscr{D} :: \Xi \vdash \Gamma\ \mathsf{ctx}$ *then* $\Xi' \vdash \Gamma\ \mathsf{ctx}$.

(6) If $\mathscr{D} :: \Theta \vdash t \equiv t' : \tau$ *then* $\Theta' \vdash t \equiv t' : \tau$.

(7) If $\mathscr{D} :: \Theta; [\tau] \vdash \boldsymbol{t} \equiv \boldsymbol{t}' : \kappa$ *then* $\Theta'; [\tau] \vdash \boldsymbol{t} \equiv \boldsymbol{t}' : \kappa$.

(8) If $\mathscr{D} :: \Theta \vdash \Theta_1 \equiv \Theta_2\ \mathsf{ctx}$ *then* $\Theta' \vdash \Theta_1 \equiv \Theta_2\ \mathsf{ctx}$.

(9) If $\mathscr{D} :: \Theta \vdash A \equiv^{\pm} B$ *then* $\Theta' \vdash A \equiv^{\pm} B$.

(10) If $\mathscr{D} :: \Theta \vdash \mathscr{M}'(F) \equiv \mathscr{M}(F)$ *then* $\Theta' \vdash \mathscr{M}'(F) \equiv \mathscr{M}(F)$.

(11) If $\mathscr{D} :: \Theta \vdash A \leq^{\pm} B$ *then* $\Theta' \vdash A \leq^{\pm} B$.

(12) If $\mathscr{D} :: \Xi \vdash \alpha; F \leq_\tau \beta; G$ *then* $\Xi' \vdash \alpha; F \leq_\tau \beta; G$.

(13) *If* $\mathscr{D} :: \Xi \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)$ *then* $\Xi' \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)$.

(14) *If* $\mathscr{D} :: \Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Theta; R$ *then* $\Xi' \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Theta; R$.

(15) *If* $\mathscr{D} :: \Theta; \Gamma \vdash h \Rightarrow P$ *then* $\Theta'; \Gamma \vdash h \Rightarrow P$.

(16) *If* $\mathscr{D} :: \Theta; \Gamma \vdash g \Rightarrow \uparrow P$ *then* $\Theta'; \Gamma \vdash g \Rightarrow \uparrow P$.

(17) *If* $\mathscr{D} :: \Theta; \Gamma \vdash v \Leftarrow P$ *then* $\Theta'; \Gamma \vdash v \Leftarrow P$.

(18) *If* $\mathscr{D} :: \Theta; \Gamma \vdash e \Leftarrow N$ *then* $\Theta'; \Gamma \vdash e \Leftarrow N$.

(19) *If* $\mathscr{D} :: \Theta; \Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$ *then* $\Theta'; \Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$.

(20) *If* $\mathscr{D} :: \Theta; \Gamma; [N] \vdash s \Rightarrow \uparrow P$ *then* $\Theta'; \Gamma; [N] \vdash s \Rightarrow \uparrow P$.

Moreover, each consequent (weakened) derivation has the same structure as $\mathscr{D}$.

Proof. Each part is proved by structural induction on the given derivation $\mathscr{D}$. Parts (1), (2), (3), and (4) are mutually recursive, as are parts (6) and (7); parts (9) and (10); parts (11), (12), and (13); and parts (15) through (20). Use Lemma C.6 (Sorting Weakening), Lemma C.8 (Prop. Truth Weakening), and Lemma C.36 (Ix. Equiv. Weakening) as needed, as well as previous parts. □

Lemma C.42 (Ix.-Level Subs. Weakening)**.** *If* $\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma$ *and* $\Theta_0 \subseteq \Theta_0'$ *then* $\Theta_0'; \Gamma_0 \vdash \sigma : \Theta; \Gamma$.

Proof. Similar to Lemma C.9 (Ix. Subs. Weakening), but the SubstVal case uses (the value typing part of) Lemma C.41 (Ix.-Level Weakening). □

Lemma C.43 (Index Id. Subs. Extension)**.** *If* $\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma$ *and* $a \notin dom(\Theta) \cup dom(\Theta_0)$ *then* $\Theta_0, a \div \tau; \Gamma_0 \vdash \sigma, a/a : \Theta, a \div \tau; \Gamma$ *and* $\Theta_0, a \overset{\mathsf{d}}{\div} \tau; \Gamma_0 \vdash \sigma, a/a : \Theta, a \overset{\mathsf{d}}{\div} \tau; \Gamma$ *and* $\Theta_0, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id}; \Gamma_0 \vdash \sigma, a/a : \Theta, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id}; \Gamma$.

Proof. We show the "$\div$" part; the "$\overset{\mathsf{d}}{\div}$" parts are similar (use SubstIxDet and $\overset{\mathsf{d}}{\div}-$ def.).

$\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma$	Given
$\Theta_0, a \div \tau; \Gamma_0 \vdash \sigma : \Theta; \Gamma$	By Lemma C.42 (Ix.-Level Subs. Weakening)
$\overline{\Theta_0}, a \div \tau \vdash a : \tau$	By IxVar
$\Theta_0, a \div \tau; \Gamma_0 \vdash \sigma, a/a : \Theta, a \div \tau; \Gamma$	By SubstIx

□

Lemma C.44 (Assumption Subst. Extension)**.** *Assume* $\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma$.

If $\overline{\Theta} \vdash \varphi : \mathbb{B}$ *then* $\Theta_0, [\lfloor \sigma \rfloor]\varphi; \Gamma_0 \vdash \sigma : \Theta, \varphi; \Gamma$.

Proof. Assume $\overline{\Theta} \vdash \varphi : \mathbb{B}$.

$\overline{\Theta_0} \vdash [\lfloor \sigma \rfloor]\varphi : \mathbb{B}$	By Lemma C.17 (Ix. Syntactic Substitution)
$\Theta_0, [\lfloor \sigma \rfloor]\varphi; \Gamma_0 \vdash \sigma : \Theta; \Gamma$	By Lemma C.42 (Ix.-Level Subs. Weakening)
$\Theta_0, [\lfloor \sigma \rfloor]\varphi \vdash [\lfloor \sigma \rfloor]\varphi\ \mathsf{true}$	By Lemma C.28 (Assumption)
$\Theta_0, [\lfloor \sigma \rfloor]\varphi; \Gamma_0 \vdash \sigma : \Theta, \varphi; \Gamma$	By SubstProp

□

Lemma C.45 (Ix.-Level Id. Subs. Extension)**.** *Assume* $\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma$.

If $(\Theta, \Theta')\ \mathsf{ctx}$ *and* $(\Theta_0, \Theta')\ \mathsf{ctx}$ *then* $\Theta_0, [\lfloor \sigma \rfloor]\Theta'; \Gamma_0 \vdash \sigma, id_{\Theta'} : \Theta, \Theta'; \Gamma$.

Proof. By structural induction on the given substitution typing derivation, analyzing cases for the structure of Θ'. The $\Theta' = (\Theta'_1, a : \tau)$ case uses Lemma C.43 (Index Id. Subs. Extension). The $\Theta' = (\Theta'_1, \varphi)$ case uses Lemma C.44 (Assumption Subst. Extension). □

Lemma C.46 (Det. FV)**.** *If* $\xi \vdash \mathfrak{D}\ \mathsf{det}$ *then* $\mathfrak{D} \subseteq FV(\xi)$.

Proof. Assume $a \in \mathfrak{D}$. It is straightforward to prove $a \in FV(\xi)$ by structural induction on $\xi \vdash a\ \mathsf{det}$. □

Lemma C.47 (ξ FV)**.** *(1) If* $\Xi \vdash t : \tau\ [\xi]$ *then* $FV(\xi) \subseteq dom(\overset{\mathsf{d}}{\div}\Xi)$.

(2) If $\Xi \vdash A\ \mathsf{type}[\xi]$ *then* $FV(\xi) \subseteq dom(\overset{\mathsf{d}}{\div}\Xi)$.

(3) If $\Xi \vdash \mathscr{F}\ \mathsf{functor}[\xi]$ *then* $FV(\xi) \subseteq dom(\overset{\mathsf{d}}{\div}\Xi)$.

Proof. By inspection of the rules, only value-determined variables are added to ξ. □

Lemma C.48 ($\langle-\rangle(-)$ Equation)**.**
If $\Xi_0 \vdash \sigma : \Xi$ *and* $\mathfrak{D} \subseteq dom(\overset{\mathsf{d}}{\div}\Xi) \cup \mathfrak{A}$
and $dom(\Xi_0) \cap \mathfrak{A} = \emptyset$ *and* $dom(\Xi) \cap \mathfrak{A} = \emptyset$
then $\langle\sigma\rangle(\mathfrak{D} - \mathfrak{A}) = \langle\sigma\rangle\mathfrak{D} - \mathfrak{A}$.

Proof. ($\subseteq$) Assume $a' \in \langle\sigma\rangle(\mathfrak{D} - \mathfrak{A})$. Then there exists $a \in (\mathfrak{D} - \mathfrak{A})$ such that *either* $\sigma(a)$ is defined and $a' \in FV(\sigma(a))$ *or* $\sigma(a)$ is undefined and $a = a'$. In either case, we have $a' \in \langle\sigma\rangle\mathfrak{D} - \mathfrak{A}$.

($\supseteq$) Assume $b' \in \langle\sigma\rangle\mathfrak{D} - \mathfrak{A}$. Then $b' \notin \mathfrak{A}$ and there exists $b \in \mathfrak{D}$ such that *either* $\sigma(b)$ is defined and $b' \in FV(\sigma(b))$ *or* $\sigma(b)$ is undefined and $b = b'$. In the first case, $b \notin \mathfrak{A}$ because otherwise we contradict the fact that $\sigma(b)$ is defined. In the second case, $b \notin \mathfrak{A}$ because otherwise we contradict the fact that $b' \notin \mathfrak{A}$. In either case, $b' \in \langle\sigma\rangle(\mathfrak{D} - \mathfrak{A})$. □

Lemma C.49 ($^{\mathsf{d}}[-]-$ Equation)**.**
If $\Xi_0 \vdash \sigma : \Xi$ *and* $FV(\xi) \subseteq dom(\overset{\mathsf{d}}{\div}\Xi) \cup \mathfrak{A}$
and $dom(\Xi_0) \cap \mathfrak{A} = \emptyset$ *and* $dom(\Xi) \cap \mathfrak{A} = \emptyset$
then $^{\mathsf{d}}[\sigma](\xi - \mathfrak{A}) = {}^{\mathsf{d}}[\sigma]\xi - \mathfrak{A}$.

Proof. ($\subseteq$) Assume $\mathfrak{B}' \to a' \in {}^{\mathsf{d}}[\sigma](\xi - \mathfrak{A})$. Then there exist $\mathfrak{B}$ and a such that $\mathfrak{B} \to a \in \xi$ and $\langle\sigma\rangle(\mathfrak{B} - \mathfrak{A}) = \mathfrak{B}'$ and $a \notin \mathfrak{A}$ and *either* $\sigma(a)$ is defined and $\sigma(a) = a'$ *or* $\sigma(a)$ is undefined and $a = a'$. In either case, we know $a' \notin \mathfrak{A}$. By Lemma C.48, $\langle\sigma\rangle(\mathfrak{B} - \mathfrak{A}) = \langle\sigma\rangle\mathfrak{B} - \mathfrak{A}$ so $\mathfrak{B}' = \langle\sigma\rangle\mathfrak{B} - \mathfrak{A}$. It follows that $\mathfrak{B}' \to a' \in {}^{\mathsf{d}}[\sigma]\xi - \mathfrak{A}$.

($\supseteq$) Assume $\mathfrak{C}'{\to}b' \in {}^{\mathsf{d}}[\sigma]\xi - \mathfrak{A}$. Then $b' \notin \mathfrak{A}$ and there exist $\mathfrak{C}$ and b such that $\mathfrak{C}{\to}b \in \xi$ and $\mathfrak{C}' = \langle\sigma\rangle\mathfrak{C} - \mathfrak{A}$ and *either* $\sigma(b) = b'$ *or* $\sigma(b)$ is undefined and $b = b'$. In either case, we know $b \notin \mathfrak{A}$. By Lemma C.48, $\langle\sigma\rangle\mathfrak{C} - \mathfrak{A} = \langle\sigma\rangle(\mathfrak{C} - \mathfrak{A})$. It follows that $\mathfrak{C}'{\to}b' \in {}^{\mathsf{d}}[\sigma](\xi - \mathfrak{A})$. □

Lemma C.50 (Det. Preservation)**.** *If* $\Xi_0 \vdash \sigma : \Xi$ *and* $\xi - {}^{\mathsf{d}}\!\div\Xi \vdash \mathfrak{D}$ det *and* $FV(\xi) \subseteq dom({}^{\mathsf{d}}\!\div\Xi) \cup \mathfrak{D} \cup \mathfrak{A}$ *and* $dom(\Xi_0) \cap (\mathfrak{D} \cup \mathfrak{A}) = \emptyset$ *and* $dom(\Xi) \cap (\mathfrak{D} \cup \mathfrak{A}) = \emptyset$ *then* ${}^{\mathsf{d}}[\sigma]\xi - {}^{\mathsf{d}}\!\div\Xi_0 \vdash \mathfrak{D}$ det.

Proof. By the given conditions and definitions it follows that $\xi - {}^{\mathsf{d}}\!\div\Xi = {}^{\mathsf{d}}[\sigma]\xi - {}^{\mathsf{d}}\!\div\Xi_0$. □

Lemma C.51 (WF Syn. Substitution)**.** *Assume* $\Xi_0 \vdash \sigma : \Xi$.

(1) If $\Xi \vdash A$ type$[\xi]$ *then* $\Xi_0 \vdash [\sigma]A$ type$[{}^{\mathsf{d}}[\sigma]\xi]$.

(2) If $\Xi \vdash \mathscr{F}$ functor$[\xi]$ *then* $\Xi_0 \vdash [\sigma]\mathscr{F}$ functor$[{}^{\mathsf{d}}[\sigma]\xi]$.

(3) If $\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$ *then* $\Xi_0 \vdash [\sigma]\alpha : ([\sigma]F)(\tau) \Rightarrow \tau$.

(4) If $\Xi \vdash \mathscr{M}'(F)$ msmts$[\xi]$ *then* $\Xi_0 \vdash [\sigma](\mathscr{M}'(F))$ msmts$[{}^{\mathsf{d}}[\sigma]\xi]$.

Proof. By structural induction on the given well-formedness derivation. Part (2) uses part (1) and part (3) uses part (2).

(1) • **Case**

$$\frac{}{\Xi \vdash \underbrace{0}_{A} \text{ type}[\underbrace{\cdot}_{\xi}]}\ \text{DeclTp0}$$

	$\Xi_0 \vdash 0$ type$[\cdot]$	By DeclTp0
	$\Xi_0 \vdash [\sigma]0$ type$[\cdot]$	By def. of subst.
☞	$\Xi_0 \vdash [\sigma]0$ type$[{}^{\mathsf{d}}[\sigma]\cdot]$	By Def. C.4

- **Case** DeclTp1: Similar to DeclTp0 case.

- **Case**

$$\frac{\Xi \vdash P_1\ \mathsf{type}[\xi_1] \qquad \Xi \vdash P_2\ \mathsf{type}[\xi_2]}{\Xi \vdash P_1 + P_2\ \mathsf{type}[\cdot]}\ \text{DeclTp}+$$

	$\Xi_0 \vdash \sigma : \Xi$	Given
	$\Xi_0 \vdash [\sigma]P_1\ \mathsf{type}[{}^{\mathsf{d}}[\sigma]\xi_1]$	By i.h.
	$\Xi_0 \vdash [\sigma]P_2\ \mathsf{type}[{}^{\mathsf{d}}[\sigma]\xi_2]$	By i.h.
	$\Xi_0 \vdash [\sigma]P_1 + [\sigma]P_2\ \mathsf{type}[\cdot]$	By DeclTp+
	$\Xi_0 \vdash [\sigma](P_1 + P_2)\ \mathsf{type}[\cdot]$	By def. of subst.
☞	$\Xi_0 \vdash [\sigma](P_1 + P_2)\ \mathsf{type}[{}^{\mathsf{d}}[\sigma]\cdot]$	By Def. C.4

- **Case**

$$\frac{\Xi \vdash R_1\ \mathsf{type}[\xi_1] \qquad \Xi \vdash R_2\ \mathsf{type}[\xi_2]}{\Xi \vdash R_1 \times R_2\ \mathsf{type}[\xi_1 \cup \xi_2]}\ \text{DeclTp}\times$$

$\Xi_0 \vdash [\sigma]R_1\ \mathsf{type}[{}^{\mathsf{d}}[\sigma]\xi_1]$	By i.h.
$\Xi_0 \vdash [\sigma]R_2\ \mathsf{type}[{}^{\mathsf{d}}[\sigma]\xi_2]$	By i.h.
$\Xi_0 \vdash [\sigma]R_1 \times [\sigma]R_2\ \mathsf{type}[{}^{\mathsf{d}}[\sigma]\xi_1 \cup {}^{\mathsf{d}}[\sigma]\xi_2]$	By DeclTp$\times$
$\Xi_0 \vdash [\sigma](R_1 \times R_2)\ \mathsf{type}[{}^{\mathsf{d}}[\sigma]\xi_1 \cup {}^{\mathsf{d}}[\sigma]\xi_2]$	By def. of subst.
$\Xi_0 \vdash [\sigma](R_1 \times R_2)\ \mathsf{type}[{}^{\mathsf{d}}[\sigma](\xi_1 \cup \xi_2)]$	By Lemma C.14 (ξ Subst. Union)

- **Case**

$$\frac{\Xi, {}^{\mathsf{d}}\Xi \vdash Q\ \mathsf{type}[\xi_Q] \qquad \xi_Q - \overset{\mathsf{d}}{\div} \Xi \vdash {}^{\mathsf{d}}\Xi\ \mathsf{det}}{\Xi \vdash \exists^{\mathsf{d}}\Xi.\, Q\ \mathsf{type}[\xi_Q - {}^{\mathsf{d}}\Xi]}\ \text{DeclTp}\exists$$

$\Xi_0 \vdash \sigma : \Xi$ Given

$\Xi_0, {}^{\mathsf{d}}\Xi \vdash \sigma, {}^{\mathsf{d}}\Xi / {}^{\mathsf{d}}\Xi : \Xi, {}^{\mathsf{d}}\Xi$ By Lemma C.43 (Index Id. Subs. Extension)

$\Xi, {}^{\mathsf{d}}\Xi \vdash Q\ \mathsf{type}[\xi_Q]$ Subderivation

$\Xi_0, {}^{\mathsf{d}}\Xi \vdash [\sigma, {}^{\mathsf{d}}\Xi / {}^{\mathsf{d}}\Xi]Q\ \mathsf{type}[{}^{\mathsf{d}}[\sigma, {}^{\mathsf{d}}\Xi / {}^{\mathsf{d}}\Xi]\xi_Q]$ By i.h.

$\Xi_0, {}^{\mathsf{d}}\Xi \vdash [\sigma]Q\ \mathsf{type}[{}^{\mathsf{d}}[\sigma]\xi_Q]$ Identity substitution

$\xi_Q - \overset{\mathsf{d}}{\div} \Xi \vdash {}^{\mathsf{d}}\Xi\ \mathsf{det}$ Premise

$FV(\xi_Q) \subseteq dom(\overset{\mathsf{d}}{\div}(\Xi, {}^{\mathsf{d}}\Xi))$ By Lemma C.47 (ξ FV)

$= dom(\overset{\mathsf{d}}{\div}\Xi, {}^{\mathsf{d}}\Xi)$ By def. of $\overset{\mathsf{d}}{\div} -$ and $dom(-)$

$\mathfrak{D} \subseteq FV(\xi_Q - \overset{\mathsf{d}}{\div}\Xi)$ Straightforward

$\subseteq dom({}^{\mathsf{d}}\Xi)$ By definition of subtraction

$dom(\Xi_0) \cap dom({}^{\mathsf{d}}\Xi) = \emptyset$ By variable convention

$dom(\Xi) \cap dom({}^{\mathsf{d}}\Xi) = \emptyset$ By variable convention

${}^{\mathsf{d}}[\sigma]\xi_Q - \overset{\mathsf{d}}{\div}\Xi_0 \vdash {}^{\mathsf{d}}\Xi\ \mathsf{det}$ By Lemma C.50 (Det. Preservation)

$\Xi_0 \vdash \exists {}^{\mathsf{d}}\Xi.\, [\sigma]Q\ \mathsf{type}[{}^{\mathsf{d}}[\sigma]\xi_Q - {}^{\mathsf{d}}\Xi]$ By DeclTp$\exists$

$\Xi_0 \vdash [\sigma]\exists {}^{\mathsf{d}}\Xi.\, Q\ \mathsf{type}[{}^{\mathsf{d}}[\sigma]\xi_Q - {}^{\mathsf{d}}\Xi]$ By def. of subst.

$\Xi_0 \vdash [\sigma]\exists {}^{\mathsf{d}}\Xi.\, Q\ \mathsf{type}[{}^{\mathsf{d}}[\sigma](\xi_Q - {}^{\mathsf{d}}\Xi)]$ By Lemma C.49

- **Case** DeclTp$\wedge$: Straightforward. Use Lemma C.14 (ξ Subst. Union).

- **Case**
$$\frac{\Xi \vdash N\ \mathsf{type}[\xi]}{\Xi \vdash {\downarrow}N\ \mathsf{type}[\cdot]}\ \text{DeclTp}{\downarrow}$$

$\Xi_0 \vdash [\sigma]N\ \mathsf{type}[_]$ By i.h.

$\Xi_0 \vdash [\sigma]{\downarrow}N\ \mathsf{type}[\cdot]$ By DeclTp$\downarrow$ and def. of subst.

$\Xi_0 \vdash [\sigma]{\downarrow}N\ \mathsf{type}[^{\mathsf{d}}[\sigma]\cdot]$ By Def. C.4

- **Case**

$$\frac{\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi]}{\Xi \vdash \{\nu : \mu F \mid \mathscr{M}(F)\}\ \mathsf{type}[\xi]}\ \text{DeclTp}\mu$$

$\Xi_0 \vdash [\sigma]\mathscr{M}(F)\ \mathsf{msmts}[^{\mathsf{d}}[\sigma]\xi]$ By i.h.

$\Xi_0 \vdash [\sigma]\mathscr{M}([\sigma]F)\ \mathsf{msmts}[^{\mathsf{d}}[\sigma]\xi]$ As $FV(F) = \emptyset$ (follows from premise)

$\Xi_0 \vdash \{\nu : \mu[\sigma]F \mid [\sigma]\mathscr{M}([\sigma]F)\}\ \mathsf{type}[^{\mathsf{d}}[\sigma]\xi]$ By DeclTpμ

$\Xi_0 \vdash [\sigma]\{\nu : \mu F \mid \mathscr{M}(F)\}\ \mathsf{type}[^{\mathsf{d}}[\sigma]\xi]$ By def.

- **Case** DeclTp$\forall$: Similar to case for dual rule DeclTp$\exists$.
- **Case** DeclTp$\supset$: Similar to case for dual rule DeclTp$\wedge$.
- **Case** DeclTp$\rightarrow$: Similar to case for DeclTp$\times$.
- **Case** DeclTp$\uparrow$: Similar to case for dual rule DeclTp$\downarrow$.

(2) Similar to part (1) but simpler.

(3)

- **Case**

$$\frac{\alpha \circ \mathsf{inj}_1 \stackrel{\circ}{=} \alpha_1 \qquad \Xi \vdash \alpha_1 : F_1(\tau) \Rightarrow \tau \qquad \alpha \circ \mathsf{inj}_2 \stackrel{\circ}{=} \alpha_2 \qquad \Xi \vdash \alpha_2 : F_2(\tau) \Rightarrow \tau}{\Xi \vdash \alpha : (F_1 \oplus F_2)(\tau) \Rightarrow \tau}\ \text{DeclAlg}\oplus$$

$\alpha \circ \mathrm{inj}_k \stackrel{\circ}{=} \alpha_k$ Premises

$([\sigma]\alpha) \circ \mathrm{inj}_k \stackrel{\circ}{=} [\sigma]\alpha_k$ By Lemma C.40

$\Xi_0 \vdash [\sigma]\alpha_k : ([\sigma]F_k)(\tau) \Rightarrow \tau$ By i.h.

$\Xi_0 \vdash [\sigma]\alpha : ([\sigma]F_1 \oplus [\sigma]F_2)(\tau) \Rightarrow \tau$ By DeclAlg$\oplus$

$\Xi_0 \vdash [\sigma]\alpha : ([\sigma](F_1 \oplus F_2))(\tau) \Rightarrow \tau$ By definition of substitution

- **Case**

$$\frac{\stackrel{\mathrm{d}}{\div}\Xi \vdash t : \tau}{\Xi \vdash () \Rightarrow t : I(\tau) \Rightarrow \tau}\ \text{DeclAlg}I$$

$\Xi_0 \vdash \sigma : \Xi$ Given

$\stackrel{\mathrm{d}}{\div}\Xi_0 \vdash \sigma\restriction_{\stackrel{\mathrm{d}}{\div}\Xi} : \stackrel{\mathrm{d}}{\div}\Xi$ By Lemma C.12 (Value-Det. Substitution)

$\stackrel{\mathrm{d}}{\div}\Xi_0 \vdash [\sigma]t : \tau$ By Lemma C.17 (Ix. Syntactic Substitution) and property of restriction/substitution

$\Xi_0 \vdash () \Rightarrow [\sigma]t : I(\tau) \Rightarrow \tau$ By DeclAlg$I$

$\Xi_0 \vdash [\sigma](() \Rightarrow t) : ([\sigma]I)(\tau) \Rightarrow \tau$ By def. of subst.

- **Case**

$$\frac{\Xi, a \stackrel{\mathrm{d}}{\div} \tau, a\,\mathrm{Id} \vdash q \Rightarrow t : \hat{P}(\tau) \Rightarrow \tau}{\Xi \vdash (a,q) \Rightarrow t : (\mathrm{Id} \otimes \hat{P})(\tau) \Rightarrow \tau}\ \text{DeclAlgId}$$

$\Xi_0 \vdash \sigma : \Xi$ Given

$\stackrel{\mathrm{d}}{\div}\Xi_0 \vdash \sigma\restriction_{\stackrel{\mathrm{d}}{\div}\Xi} : \stackrel{\mathrm{d}}{\div}\Xi$ By Lemma C.12 (Value-Det. Substitution)

$\Xi_0, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id} \vdash [\sigma](q \Rightarrow t) : ([\sigma]\hat{P})(\tau) \Rightarrow \tau$ By i.h., identity subst.

$\Xi_0, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id} \vdash q \Rightarrow [\sigma]t : ([\sigma]\hat{P})(\tau) \Rightarrow \tau$ By def. of subst.

$\Xi_0 \vdash (a, q) \Rightarrow [\sigma]t : (\mathrm{Id} \otimes [\sigma]\hat{P})(\tau) \Rightarrow \tau$ By DeclAlgId

☞ $\Xi_0 \vdash [\sigma]((a, q) \Rightarrow t) : ([\sigma](\mathrm{Id} \otimes \hat{P}))(\tau) \Rightarrow \tau$ By def. of subst.

- **Case**

$$\frac{\Xi \vdash Q\ \mathsf{type}[\xi_Q] \qquad \Xi \vdash q \Rightarrow t : \hat{P}(\tau) \Rightarrow \tau}{\Xi \vdash (\top, q) \Rightarrow t : (\underline{Q} \otimes \hat{P})(\tau) \Rightarrow \tau}\ \text{DeclAlgConst}$$

$\Xi \vdash Q\ \mathsf{type}[\xi_Q]$ Premise

$\Xi_0 \vdash [\sigma]Q\ \mathsf{type}[_]$ By part (1)

$\Xi \vdash q \Rightarrow t : \hat{P}(\tau) \Rightarrow \tau$ Subderivation

$\Xi_0 \vdash q \Rightarrow [\sigma]t : ([\sigma]\hat{P})(\tau) \Rightarrow \tau$ By i.h. and def. of subst.

$\Xi_0 \vdash (\top, q) \Rightarrow [\sigma]t : ([\sigma]\underline{Q} \otimes [\sigma]\hat{P})(\tau) \Rightarrow \tau$ By DeclAlgConst

$\Xi_0 \vdash [\sigma]((\top, q) \Rightarrow t) : [\sigma](\underline{Q} \otimes \hat{P})(\tau) \Rightarrow \tau$ By def. of subst.

- **Case**

$$\frac{\Xi, {}^{\mathsf{d}}\Xi' \vdash (\top, q) \Rightarrow t : (\underline{Q} \otimes \hat{P})(\tau) \Rightarrow \tau}{\Xi \vdash (\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q) \Rightarrow t : (\underline{\exists^{\mathsf{d}}\Xi'.\, Q} \otimes \hat{P})(\tau) \Rightarrow \tau}\ \text{DeclAlg}\exists$$

By (repeated) Lemma C.43 (Index Id. Subs. Extension),

$$\Xi_0, {}^{\mathsf{d}}\Xi' \vdash \sigma, {}^{\mathsf{d}}\Xi' / {}^{\mathsf{d}}\Xi' : \Xi, {}^{\mathsf{d}}\Xi'$$

By i.h. and def. of substitution and identity substitution,

$$\Xi_0, {}^{\mathsf{d}}\Xi' \vdash (\top, q) \Rightarrow [\sigma]t : ([\sigma]\underline{Q} \otimes [\sigma]\hat{P})(\tau) \Rightarrow \tau$$

By DeclAlg$\exists$,

$$\Xi_0 \vdash (\mathtt{pk}(^{\mathsf{d}}\Xi', \top), q) \Rightarrow [\sigma]t : (\underline{\exists^{\mathsf{d}}\Xi'.\,[\sigma]Q} \otimes [\sigma]\hat{P})(\tau) \Rightarrow \tau$$

By definition of substitution,

$$\Xi_0 \vdash [\sigma]((\mathtt{pk}(^{\mathsf{d}}\Xi', \top), q) \Rightarrow t) : ([\sigma](\underline{\exists^{\mathsf{d}}\Xi'.\,Q} \otimes \hat{P}))(\tau) \Rightarrow \tau$$

(4) • **Case**

$$\frac{\cdot \vdash F\ \mathsf{functor}[_]}{\Xi \vdash \cdot_F\ \mathsf{msmts}[\cdot]}$$

$\cdot \vdash F\ \mathsf{functor}[_]$	Premise
$\cdot \vdash [\sigma]F\ \mathsf{functor}[_]$	As $FV(F) = \emptyset$
$\Xi_0 \vdash \cdot_{[\sigma]F}\ \mathsf{msmts}[\cdot]$	By same rule
$\Xi_0 \vdash \cdot_{[\sigma]F}\ \mathsf{msmts}[^{\mathsf{d}}[\sigma]\cdot]$	By Def. C.4
$\Xi_0 \vdash [\sigma](\cdot_F)\ \mathsf{msmts}[^{\mathsf{d}}[\sigma]\cdot]$	By def. of subst.

• **Case**

$$\frac{\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi] \qquad \cdot \vdash \alpha : F(\tau) \Rightarrow \tau \qquad {}^{\mathsf{d}}\dot{-}\Xi; [\tau] \vdash \boldsymbol{t} : \kappa \qquad (t\ {}^{\mathsf{d}}\dot{-}\ \kappa) \in \Xi}{\Xi \vdash \mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\, \nu\, \boldsymbol{t} =_\tau t\ \mathsf{msmts}[\xi \cup FV(\boldsymbol{t}) \to t]}$$

$\Xi_0 \vdash \sigma : \Xi$	Given
$\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi]$	Subderivation
$\Xi_0 \vdash [\sigma](\mathscr{M}(F))\ \mathsf{msmts}[\sigma \vdash \xi\ \mathsf{det}]$	By i.h.
$\cdot \vdash \alpha : F(\tau) \Rightarrow \tau$	Premise
$\cdot \vdash [\sigma]\alpha : ([\sigma]F)(\tau) \Rightarrow \tau$	As $FV(\alpha, F) = \emptyset$
$\overset{\mathsf{d}}{\div}\Xi_0 \vdash \sigma\restriction_{\overset{\mathsf{d}}{\div}\Xi} : \overset{\mathsf{d}}{\div}\Xi$	By Lemma C.12 (Value-Det. Substitution)
$\overset{\mathsf{d}}{\div}\Xi; [\tau] \vdash \boldsymbol{t} : \kappa$	Premise
$\overset{\mathsf{d}}{\div}\Xi_0; [\tau] \vdash [\sigma\restriction_{\overset{\mathsf{d}}{\div}\Xi}]\boldsymbol{t} : \kappa$	By Lemma C.17 (Ix. Syntactic Substitution)
$\overset{\mathsf{d}}{\div}\Xi_0; [\tau] \vdash [\sigma]\boldsymbol{t} : \kappa$	By restriction property
	$(FV(\boldsymbol{t}) \subseteq dom(\overset{\mathsf{d}}{\div}\Xi))$
$(t \overset{\mathsf{d}}{\div} \kappa) \in \Xi$	Premise
$\overset{\mathsf{d}}{\div}\Xi_0 \vdash \sigma(t) : \kappa$	By inversion on substitution typing

– **Case** $\sigma(t) \in dom(\Xi_0)$

By a rule,

$$\Xi_0 \vdash [\sigma](\mathscr{M}(F)), (\mathsf{fold}_{[\sigma]F}\ [\sigma]\alpha)\ \nu\ [\sigma]\boldsymbol{t} =_\tau [\sigma]t\ \mathsf{msmts}[\lceil\mathsf{d}\rceil[\sigma]\xi \cup FV([\sigma]\boldsymbol{t}) \to \sigma(t)]$$

$\Xi_0 \vdash [\sigma](\mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\ \nu\ \boldsymbol{t} =_\tau t)\ \mathsf{msmts}[\lceil\mathsf{d}\rceil[\sigma]\xi \cup FV([\sigma]\boldsymbol{t}) \to \sigma(t)]$	By def.
$\Xi_0 \vdash [\sigma](\mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\ \nu\ \boldsymbol{t} =_\tau t)\ \mathsf{msmts}[\lceil\mathsf{d}\rceil[\sigma]\xi \cup \langle\sigma\rangle FV(\boldsymbol{t}) \to \sigma(t)]$	By Def. C.4 and Def. C.3
$\Xi_0 \vdash [\sigma](\mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\ \nu\ \boldsymbol{t} =_\tau t)\ \mathsf{msmts}[\lceil\mathsf{d}\rceil[\sigma]\xi \cup \lceil\mathsf{d}\rceil[\sigma](FV(\boldsymbol{t}) \to t)]$	By Def. C.4
$\Xi_0 \vdash [\sigma](\mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\ \nu\ \boldsymbol{t} =_\tau t)\ \mathsf{msmts}[\lceil\mathsf{d}\rceil[\sigma](\xi \cup FV(\boldsymbol{t}) \to t)]$	Lemma C.14

– **Case** $\sigma(t) \notin dom(\Xi_0)$

$\Xi_0 \vdash [\sigma](\mathscr{M}(F)), (\mathsf{fold}_{[\sigma]F}\ [\sigma]\alpha)\, \nu\, [\sigma]\boldsymbol{t} =_\tau [\sigma]t\ \mathsf{msmts}[^{\mathsf{d}}[\sigma]\xi]$ By rule

$\Xi_0 \vdash [\sigma](\mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\, \nu\, \boldsymbol{t} =_\tau t)\ \mathsf{msmts}[^{\mathsf{d}}[\sigma]\xi]$ By def.

$\Xi_0 \vdash [\sigma](\mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\, \nu\, \boldsymbol{t} =_\tau t)\ \mathsf{msmts}[^{\mathsf{d}}[\sigma]\xi \cup \emptyset]$ $\emptyset$ unit for $\cup$

$\Xi_0 \vdash [\sigma](\mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\, \nu\, \boldsymbol{t} =_\tau t)\ \mathsf{msmts}[^{\mathsf{d}}[\sigma]\xi \cup {}^{\mathsf{d}}[\sigma](FV(\boldsymbol{t}) \to t)]$ By Def. C.4

$\Xi_0 \vdash [\sigma](\mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\, \nu\, \boldsymbol{t} =_\tau t)\ \mathsf{msmts}[^{\mathsf{d}}[\sigma](\xi \cup FV(\boldsymbol{t}) \to t)]$ Lemma C.14

- **Case**

$$\frac{\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi] \qquad \cdot \vdash \alpha : F(\tau) \Rightarrow \tau \qquad {}^{\mathsf{d}\div}\Xi; [\tau] \vdash \boldsymbol{t} : \kappa \qquad (t\ {}^{\mathsf{d}\div}\ \kappa) \notin \Xi \qquad {}^{\mathsf{d}\div}\Xi \vdash t : \kappa}{\Xi \vdash \mathscr{M}(F), (\mathsf{fold}_F\ \alpha)\, \nu\, \boldsymbol{t} =_\tau t\ \mathsf{msmts}[\xi]}$$

Similar to previous case. □

Lemma C.52 (Det. Weakening)**.** *If* $\xi \vdash \mathfrak{D}\ \mathsf{det}$ *and* $\xi \subseteq \xi'$ *then* $\xi' \vdash \mathfrak{D}\ \mathsf{det}$.

Proof. Assume $a \in \mathfrak{D}$. It is straightforward to prove $\xi' \vdash a\ \mathsf{det}$ by structural induction on $\xi \vdash a\ \mathsf{det}$. (Alternatively, use Lemma B.4 (Equivalence of *cl* and det) and the fact that $cl(-)(\mathscr{O})$ is monotone.) □

Lemma C.53 (Subtraction to Cut)**.** *If* $\xi \vdash \mathfrak{B}\ \mathsf{det}$ *and* $\xi - \mathfrak{B} \vdash \mathfrak{D}\ \mathsf{det}$ *then* $\xi \vdash \mathfrak{D}\ \mathsf{det}$.

Proof. By Lemma B.4 (Equivalence of *cl* and det) and Lemma B.9 (Subtraction and *cl*), as well as monotonocity of $cl(-)(\mathscr{O})$ and idempotency of $\cup$. □

Lemma C.54 (liftapps WF)**.**
If $\overrightarrow{a\ {}^{\mathsf{d}\div}\ \tau} = \overrightarrow{a}\ {}^{\mathsf{d}\div}\ \mathscr{M}(F)$ *and* $\Xi, \overrightarrow{a\ {}^{\mathsf{d}\div}\ \tau, a\ \mathrm{Id}} \vdash \wr \overrightarrow{q \Rightarrow t'}; \hat{I}; \mathscr{M}(F) \wr \mathrel{\hat{=}} \Xi'', \overrightarrow{\psi''}; R''$

and $\mathscr{O}_1'$ *is a subterm of* R'' *and* $\mathscr{O}_2'$ *is a subterm of* $\overrightarrow{\psi''}$*, each WF under* $\Xi_1, \overrightarrow{a \stackrel{\mathsf{d}\cdot}{\div} \tau, a\,\mathrm{Id}}, \Xi''$
and $\Xi; \Xi''; zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash \mathscr{O}_1' \rightsquigarrow \breve{\Xi}_1; \mathscr{M}_1(F); \mathscr{O}_1$
and $\Xi; \Xi''; zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash \mathscr{O}_2' \rightsquigarrow \breve{\Xi}_2; \mathscr{M}_2(F); \mathscr{O}_2$
and $\breve{\Xi} = \breve{\Xi}_1 \cup \breve{\Xi}_2$ *and* $\mathscr{M}'(F) = \mathscr{M}_1(F) \cup \mathscr{M}_2(F)$
and $dom(\Xi') \cap dom(\Xi, \overrightarrow{a \stackrel{\mathsf{d}\cdot}{\div} \tau, a\,\mathrm{Id}}, \Xi'', \breve{\Xi}) = \emptyset$ *and* $\rho = \Xi'/\breve{\Xi}$ *is a variable renaming*
then $\Xi, \breve{\Xi}, \Xi'' \vdash \mathscr{M}'(F)\ \mathsf{msmts}[\xi_{\mathscr{M}'}]$ *and* $cl((\xi_{[\rho]\mathscr{M}'} - \stackrel{\mathsf{d}\cdot}{\div} \Xi) - \Xi'')(\emptyset) = dom(\Xi')$
and $\mathscr{O}_1$ *and* $\mathscr{O}_2$ *are each well-formed under* $\Xi, \breve{\Xi}, \Xi''$ *and, in particular,*
if $\mathscr{O}_1'$ *is a type* R_0'' *or measurement list* $\mathscr{M}_0''$ *then* $\xi_{\mathscr{O}_1'} - \stackrel{\mathsf{d}\cdot}{\div}(\Xi, \overrightarrow{a \stackrel{\mathsf{d}\cdot}{\div} \tau}) = \xi_{[\rho]\mathscr{O}_1} - \stackrel{\mathsf{d}\cdot}{\div}(\Xi, \Xi')$,
and if $\mathscr{O}_2'$ *is an index* t_0 *then* $\xi_{\mathscr{O}_2'} - \stackrel{\mathsf{d}\cdot}{\div}(\Xi, \overrightarrow{a \stackrel{\mathsf{d}\cdot}{\div} \tau}) = \xi_{[\rho]\mathscr{O}_2} - \stackrel{\mathsf{d}\cdot}{\div}(\Xi, \Xi')$.

Proof. By structural induction on $\mathscr{O}_1'$ or $\mathscr{O}_2'$. Use Lemma B.8 (Consequence for *cl*) in the key case where $\mathscr{O}_1'$ or $\mathscr{O}_2'$ is $a_k(\boldsymbol{u})$ for some $a_k \in \overrightarrow{a}$ (and $\boldsymbol{u}$). □

We sometimes implicitly use the following lemma (which uses the preceding lemma).

Lemma C.55 (Unrolling Output WF)**.**
If $\Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \wr \stackrel{\circ}{=} {}^{\mathsf{d}}\Theta; R$ *and* $\Xi \vdash G\ \mathsf{functor}[\xi_G]$
then there exists ξ *such that* $\Xi \vdash \exists^{\mathsf{d}}\Theta.\, R \wedge {}^{\mathsf{d}}\Theta\ \mathsf{type}[\xi]$ *and* $\xi_G \subseteq \xi$.

Proof. By structural induction on the given unrolling derivation. Use Lemma C.53 (Subtraction to Cut) (in the $\wr \mathrm{Const} \wr$ case), Lemma C.54 (liftapps WF) (in the $\wr \mathrm{Id} \wr$ case), Lemma B.8 (Consequence for *cl*) (in the $\wr \mathrm{Id} \wr$ case), Lemma B.9 (Subtraction and *cl*) (in the $\wr \mathrm{Id} \wr$ case), Lemma B.4 (Equivalence of *cl* and det) (in the $\wr \mathrm{Id} \wr$ case), $cl(-)(\mathscr{O})$ monotone (in the $\wr \mathrm{Id} \wr$ case), Lemma C.52 (Det. Weakening) ($\wr \mathrm{Const} \wr$, $\wr \mathrm{Id} \wr$ cases), Lemma C.47 (ξ FV), Lemma C.41 (Ix.-Level Weakening), Lemma C.6 (Sorting Weakening), and Lemma C.17 (Ix. Syntactic Substitution). □

Lemma C.56 (liftapps Syntactic Substitution)**.**
If $\Xi;\Xi';\overrightarrow{(a,(\mathsf{fold}_F\ \alpha)\,\nu_ =_\tau _)} \vdash \mathscr{O} \rightsquigarrow \check{\Xi};\mathscr{M}'(F);\mathscr{O}'$ *and* $\Xi_0 \vdash \sigma : \Xi$
then $\Xi_0;\Xi';\overrightarrow{(a,(\mathsf{fold}_F\ \alpha)\,\nu_ =_\tau _)} \vdash [\sigma]\mathscr{O} \rightsquigarrow [\sigma]\check{\Xi};[\sigma]\mathscr{M}'(F);[\sigma]\mathscr{O}'$.

Proof. By structural induction on the given liftapps derivation. □

Lemma C.57 (Unrolling Syntactic Substitution)**.** *If* $\Xi \vdash \wr\overrightarrow{\beta};G;\mathscr{M}(F)\wr \triangleq {}^{\mathsf{d}}\Theta;R$
and $\Xi_0 \vdash \sigma : \Xi$ *then* $\Xi_0 \vdash \wr[\sigma]\overrightarrow{\beta};[\sigma]G;[\sigma]\mathscr{M}([\sigma]F)\wr \triangleq [\sigma]{}^{\mathsf{d}}\Theta;[\sigma]R$.
Moreover, the height and structure of the unrolling derivation remain the same.

Proof. By structural induction on the unrolling derivation. Use Lemma C.40 (Subst. Algebra Pattern), Lemma C.12 (Value-Det. Substitution), Lemma C.51 (WF Syn. Substitution), Lemma C.17 (Ix. Syntactic Substitution), Lemma C.56 (liftapps Syntactic Substitution), Lemma C.6 (Sorting Weakening), Lemma C.20. □

Lemma C.58 (Ix. Equiv. Syn. Subs.)**.** *Assume* $\Theta_0 \vdash \sigma : \Theta$.

(1) If $\Theta \vdash u \equiv t : \tau$ *then* $\Theta_0 \vdash [\sigma]u \equiv [\sigma]t : \tau$.

(2) If $\Theta;[\tau] \vdash \boldsymbol{t}_1 \equiv \boldsymbol{t}_2 : \kappa$ *then* $\Theta_0;[\tau] \vdash [\sigma]\boldsymbol{t}_1 \equiv [\sigma]\boldsymbol{t}_2 : \kappa$.

Proof. By mutual induction on the structure of the given equivalence derivation. Use Lemma C.30 (Prop. Truth Syn. Subs) and Lemma C.11 (Ix. Id. Subs. Extension). □

Lemma C.59 (Sub. Syn. Subs.)**.**

(1) If $\Theta \vdash A \leq^{\pm} B$ *and* $\Theta_0 \vdash \sigma : \Theta$
then $\Theta_0 \vdash [\sigma]A \leq^{\pm} [\sigma]B$ *by a derivation of equal structure and height.*

(2) If $\Xi \vdash \alpha;F \leq_\tau \beta;G$ *and* $\Xi_0 \vdash \sigma : \Xi$
then $\Xi_0 \vdash [\sigma]\alpha;[\sigma]F \leq_\tau [\sigma]\beta;[\sigma]G$ *by a derivation of equal structure and height.*

(3) If $\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)$ *and* $\Theta_0 \vdash \sigma : \Theta$

then $\Theta_0 \vdash [\sigma]\mathscr{M}'(F') \geq [\sigma]\mathscr{M}(F)$ *by a derivation of equal structure and height.*

Proof. Each part by structural induction on the given subtyping/submeasuring derivation. Part (2) uses part (1) and part (1) uses part (3). Use these lemmas: Lemma C.13 (Ix. Id. Subst), Lemma C.17 (Ix. Syntactic Substitution), Lemma C.9 (Ix. Subs. Weakening), Lemma C.12 (Value-Det. Substitution), Lemma C.58 (Ix. Equiv. Syn. Subs), Lemma C.38 (Type/Functor Barendregt), Lemma C.40 (Subst. Algebra Pattern), Lemma C.39 (Algebra Barendregt). □

Appendix C.2 Subtyping Properties

Definition C.7. Given Θ ctx and $\overline{\Theta} \vdash \Gamma$ ctx and $\overline{\Theta} \vdash \Gamma'$ ctx, define $\Theta \vdash \Gamma \leq^{+} \Gamma'$ by pointwise subtyping of the variables' types, assuming $dom(\Gamma) = dom(\Gamma')$.

Lemma C.60 (Ix. Equiv. Consequence)**.**

(1) If $\Theta_1, \varphi, \Theta_2 \vdash t_1 \equiv t_2 : \tau$ *and* $\Theta_1 \vdash \varphi$ true *then* $\Theta_1, \Theta_2 \vdash t_1 \equiv t_2 : \tau$.

(2) If $\Theta_1, \varphi, \Theta_2; [\tau] \vdash \boldsymbol{t}_1 \equiv \boldsymbol{t}_2 : \kappa$ *and* $\Theta_1 \vdash \varphi$ true *then* $\Theta_1, \Theta_2; [\tau] \vdash \boldsymbol{t}_1 \equiv \boldsymbol{t}_2 : \kappa$.

Proof. By mutual induction on the structure of the given equivalence derivation. The Ix≡SMT case uses Lemma C.29 (Consequence). □

Lemma C.61 (Subtyping Consequence)**.**

If $\Theta_1 \vdash \varphi$ true *and* $\Theta_1, \varphi, \Theta_2 \vdash A \leq^{\pm} B$ *then* $\Theta_1, \Theta_2 \vdash A \leq^{\pm} B$.

Proof. By structural induction on the given subtyping derivation, analyzing cases for the latter's concluding rule. Use Lemma C.29 (Consequence) when necessary (e.g., for the $\leq^{+}\wedge$R case). The judgment $\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)$ is proposition independent because its

functors and algebras are closed and the operation $\overset{\mathsf{d}}{\div} -$ removes propositions from logical contexts. □

Lemma C.62 (Subtyping Reflexive)**.**

(1) If $\Xi \vdash A\ \mathsf{type}[\xi_A]$ *then* $\Xi \vdash A \leq^{\pm} A$.

(2) If $\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$ *then* $\Xi \vdash \alpha;F \leq_\tau \alpha;F$.

(3) If $\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi]$ *then* $\Xi \vdash \mathscr{M}(F) \geq \mathscr{M}(F)$.

Proof. By mutual induction on the structure of the given type or algebra formation derivation. Straightforward.

Use Lemma C.6 (Sorting Weakening) and Lemma C.29 (Consequence) together with both left and right subtyping rules in cases DeclTp$\wedge$ and DeclTp$\supset$. Cases DeclTp$\exists$ and DeclTp$\forall$ follow a similar pattern as cases DeclTp$\wedge$ and DeclTp$\supset$ but uses IxVar rather than Lemma C.29 (Consequence) and does not use Lemma C.6 (Sorting Weakening).

Use Lemma C.34 (Ix. Equiv. Reflexive) and Lemma C.27 (Prop. Truth Equiv. Relation) in part (3).

Use Lemma C.34 (Ix. Equiv. Reflexive) in the DeclAlg$I$ case of part (2). □

Lemma C.63 (Subtyping Transitive)**.**

(1) If $\Theta \vdash P \leq^+ \widetilde{P}$ *and* $\Theta \vdash \widetilde{P} \leq^+ P'$ *then* $\Theta \vdash P \leq^+ P'$.

(2) If $\Theta \vdash N' \leq^- \widetilde{N}$ *and* $\Theta \vdash \widetilde{N} \leq^- N$ *then* $\Theta \vdash N' \leq^- N$.

(3) If $\Xi \vdash \alpha;F \leq_\tau \widetilde{\beta};\widetilde{G}$ *and* $\Xi \vdash \widetilde{\beta};\widetilde{G} \leq_\tau \beta;G$ *then* $\Xi \vdash \alpha;F \leq_\tau \beta;G$.

(4) If $\Xi \vdash \mathscr{M}'(F') \geq \widetilde{\mathscr{M}}(\widetilde{F})$ *and* $\Xi \vdash \widetilde{\mathscr{M}}(\widetilde{F}) \geq \mathscr{M}(F)$ *then* $\Xi \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)$.

Proof. By mutual induction on the structure of the two given derivations.

(1)

- **Case**

$$\dfrac{\Theta, {}^{\mathsf{d}}\Xi' \vdash Q \leq^{+} \widetilde{P}}{\Theta \vdash \exists^{\mathsf{d}}\Xi'.\, Q \leq^{+} \widetilde{P}} \leq^{+}\exists\mathrm{L} \qquad \dfrac{\vdots}{\Theta \vdash \widetilde{P} \leq^{+} P'}$$

$\Theta, {}^{\mathsf{d}}\Xi' \vdash Q \leq^{+} \widetilde{P}$	Subderivation
$\Theta \vdash \widetilde{P} \leq^{+} P'$	Given
$\Theta, {}^{\mathsf{d}}\Xi' \vdash \widetilde{P} \leq^{+} P'$	By Lemma C.41 (Ix.-Level Weakening)
$\Theta, {}^{\mathsf{d}}\Xi' \vdash Q \leq^{+} P'$	By i.h.
$\Theta \vdash \exists^{\mathsf{d}}\Xi'.\, Q \leq^{+} P'$	By $\leq^{+}\exists\mathrm{L}$

- **Case**

$$\dfrac{\Theta, \overrightarrow{\varphi} \vdash R \leq^{+} \widetilde{P}}{\Theta \vdash R \wedge \overrightarrow{\varphi} \leq^{+} \widetilde{P}} \leq^{+}\wedge\mathrm{L} \qquad \dfrac{\vdots}{\Theta \vdash \widetilde{P} \leq^{+} P'}$$

Similar to preceding case.

- **Case**

$$\dfrac{}{\Theta \vdash 1 \leq^{+} 1} \leq^{+}1 \qquad \dfrac{}{\Theta \vdash 1 \leq^{+} 1} \leq^{+}1$$

$\Theta \vdash 1 \leq^{+} 1$ By $\leq^{+}1$

- **Case**

$$\dfrac{}{\Theta \vdash 0 \leq^{+} 0} \leq^{+}0 \qquad \dfrac{}{\Theta \vdash 0 \leq^{+} 0} \leq^{+}0$$

$\Theta \vdash 0 \leq^{+} 0$ By $\leq^{+}0$

- **Case**
$$\frac{\Theta \vdash R_1 \leq^+ \widetilde{R}_1 \qquad \Theta \vdash R_2 \leq^+ \widetilde{R}_2}{\Theta \vdash R_1 \times R_2 \leq^+ \widetilde{R}_1 \times \widetilde{R}_2} \leq^+\times \qquad \frac{\Theta \vdash \widetilde{R}_1 \leq^+ R'_1 \qquad \Theta \vdash \widetilde{R}_2 \leq^+ R'_2}{\Theta \vdash \widetilde{R}_1 \times \widetilde{R}_2 \leq^+ R'_1 \times R'_2} \leq^+\times$$

  $\Theta \vdash R_1 \leq^+ \widetilde{R}_1$ Subderivation

  $\Theta \vdash \widetilde{R}_1 \leq^+ R'_1$ Subderivation

  $\Theta \vdash R_1 \leq^+ R'_1$ By i.h.

  $\Theta \vdash R_2 \leq^+ R'_2$ Similarly

  $\Theta \vdash R_1 \times R_2 \leq^+ R'_1 \times R'_2$ By $\leq^+\times$

- **Case**
$$\frac{\Theta \vdash P_1 \leq^+ \widetilde{P}_1 \qquad \Theta \vdash P_2 \leq^+ \widetilde{P}_2}{\Theta \vdash P_1 + P_2 \leq^+ \widetilde{P}_1 + \widetilde{P}_2} \leq^+ + \qquad \frac{\Theta \vdash \widetilde{P}_1 \leq^+ P'_1 \qquad \Theta \vdash \widetilde{P}_2 \leq^+ P'_2}{\Theta \vdash \widetilde{P}_1 + \widetilde{P}_2 \leq^+ P'_1 + P'_2} \leq^+ +$$

  Similar to $\leq^+\times$ case.

- **Case**
$$\frac{\vdots}{\Theta \vdash \underbrace{R}_{P} \leq^+ \underbrace{\widetilde{R}}_{\widetilde{P}}} \qquad \frac{\Theta \vdash \widetilde{R} \leq^+ R' \qquad \Theta \vdash \overrightarrow{\varphi}\,\mathsf{true}}{\Theta \vdash \widetilde{R} \leq^+ R' \wedge \overrightarrow{\varphi}} \leq^+ \wedge \mathrm{R}$$

  $\Theta \vdash R \leq^+ \widetilde{R}$ Given

  $\Theta \vdash \widetilde{R} \leq^+ R'$ Subderivation

  $\Theta \vdash R \leq^+ R'$ By i.h.

  $\Theta \vdash \overrightarrow{\varphi}\,\mathsf{true}$ Subderivation

  $\Theta \vdash R \leq^+ R' \wedge \overrightarrow{\varphi}$ By $\leq^+ \wedge \mathrm{R}$

- **Case**
$$\frac{\vdots}{\Theta \vdash \underbrace{R}_{P} \leq^+ \underbrace{\widetilde{R}}_{\widetilde{P}}} \qquad \frac{\overset{\mathsf{d}}{\div}\Theta \vdash \sigma : {}^{\mathsf{d}}\Xi' \qquad \Theta \vdash \widetilde{R} \leq^+ [\sigma]Q}{\Theta \vdash \widetilde{R} \leq^+ \exists^{\mathsf{d}}\Xi'.\,Q} \leq^+ \exists \mathrm{R}$$

Similar to $\leq^{+}\wedge$R case.

- **Case**

$$\frac{\Theta \vdash \mathscr{M}'(F') \geq \widetilde{\mathscr{M}}(\widetilde{F})}{\Theta \vdash \left\{\nu : \mu F' \;\middle|\; \mathscr{M}'(F')\right\} \leq^{+} \left\{\nu : \mu \widetilde{F} \;\middle|\; \widetilde{\mathscr{M}}(\widetilde{F})\right\}} \leq^{+}\mu$$

  - **Case**

$$\frac{\Theta \vdash \widetilde{\mathscr{M}}(\widetilde{F}) \geq \mathscr{M}(F)}{\Theta \vdash \left\{\nu : \mu \widetilde{F} \;\middle|\; \widetilde{\mathscr{M}}(\widetilde{F})\right\} \leq^{+} \{\nu : \mu F \mid \mathscr{M}(F)\}} \leq^{+}\mu$$

    $\Theta \vdash \mathscr{M}'(F') \geq \widetilde{\mathscr{M}}(\widetilde{F})$ Subderivation

    $\Theta \vdash \widetilde{\mathscr{M}}(\widetilde{F}) \geq \mathscr{M}(F)$ Subderivation

    $\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)$ By i.h.

    $\Theta \vdash \{\nu : \mu F' \mid \mathscr{M}'(F')\} \leq^{+} \{\nu : \mu F \mid \mathscr{M}(F)\}$ By $\leq^{+}\mu$

- **Case**

$$\frac{\Theta \vdash N' \leq^{-} \widetilde{N}}{\Theta \vdash \downarrow N' \leq^{+} \downarrow \widetilde{N}} \leq^{+}\downarrow \qquad \frac{\Theta \vdash \widetilde{N} \leq^{-} N}{\Theta \vdash \downarrow \widetilde{N} \leq^{+} \downarrow N} \leq^{+}\downarrow$$

  $\Theta \vdash N' \leq^{-} \widetilde{N}$ Subderivation

  $\Theta \vdash \widetilde{N} \leq^{-} N$ Subderivation

  $\Theta \vdash N' \leq^{-} N$ By i.h.

  $\Theta \vdash \downarrow N' \leq^{-} \downarrow N$ By $\leq^{+}\downarrow$

(2) Similar to part (1). First consider the cases where $\leq^{-}\forall$R or $\leq^{-}\supset$R concludes $\Theta \vdash \widetilde{N} \leq^{-} N$, and then case analyze $\Theta \vdash N' \leq^{-} \widetilde{N}$.

(3) Similar to part (1) but simpler. First consider the case where Meas$\leq\exists$L concludes $\Xi \vdash \alpha; F \leq_{\tau} \widetilde{\beta}; \widetilde{G}$ and then case analyze $\Xi \vdash \widetilde{\beta}; \widetilde{G} \leq_{\tau} \beta; G$. In the Meas$\leq I$ case, use Lemma C.35 (Ix. Equiv. Transitive).

(4) It suffices to show that, for all $(\mathsf{fold}_F\ \alpha)\,v\,\boldsymbol{t} =_\tau t \in \mathscr{M}(F)$,

there exists $(\mathsf{fold}_{F'}\ \alpha')\,v\,\boldsymbol{t}' =_\tau t' \in \mathscr{M}'(F')$ such that the following three lines hold:

$$\cdot \vdash \alpha'; F' \leq_\tau \alpha; F$$

$$\stackrel{\mathsf{d}}{\div}\Theta; [\tau] \vdash \boldsymbol{t}' \equiv \boldsymbol{t} : \kappa$$

$$\stackrel{\mathsf{d}}{\div}\Theta \vdash t' = t\ \mathsf{true}$$

Assume $(\mathsf{fold}_F\ \alpha)\,v\,\boldsymbol{t} =_\tau t \in \mathscr{M}(F)$.

By inversion on $\Xi \vdash \widetilde{\mathscr{M}}(\widetilde{F}) \geq \mathscr{M}(F)$

there exists $(\mathsf{fold}_{\widetilde{F}}\ \widetilde{\alpha})\,v\,\widetilde{\boldsymbol{t}} =_\tau \widetilde{t} \in \widetilde{\mathscr{M}}(\widetilde{F})$ such that the following three lines hold:

$$\cdot \vdash \widetilde{\alpha}; \widetilde{F} \leq_\tau \alpha; F$$

$$\stackrel{\mathsf{d}}{\div}\Theta; [\tau] \vdash \widetilde{\boldsymbol{t}} \equiv \boldsymbol{t} : \kappa$$

$$\stackrel{\mathsf{d}}{\div}\Theta \vdash \widetilde{t} = t\ \mathsf{true}$$

By inversion on $\Xi \vdash \mathscr{M}'(F') \geq \widetilde{\mathscr{M}}(\widetilde{F})$

there exists $(\mathsf{fold}_{F'}\ \alpha')\,v\,\boldsymbol{t}' =_\tau t' \in \mathscr{M}'(F')$

such that the following three lines hold:

$$\cdot \vdash \alpha; F \leq_\tau \widetilde{\alpha}; \widetilde{F}$$

$$\stackrel{\mathsf{d}}{\div}\Theta; [\tau] \vdash \boldsymbol{t}' \equiv \widetilde{\boldsymbol{t}} : \kappa$$

$$\stackrel{\mathsf{d}}{\div}\Theta \vdash t' = \widetilde{t}\ \mathsf{true}$$

$\cdot \vdash \alpha; F \leq_\tau \alpha; F$ By i.h.

$\stackrel{\mathsf{d}}{\div}\Theta; [\tau] \vdash \boldsymbol{t}' \equiv \boldsymbol{t} : \kappa$ By Lemma C.35 (Ix. Equiv. Transitive)

$\stackrel{\mathsf{d}}{\div}\Theta \vdash t' = t\ \mathsf{true}$ By Lemma C.27 (Prop. Truth Equiv. Relation) □

Appendix C.3 Substitution Lemma and Subsumption Admissibility

Lemma C.64 (Prog.-Level Weakening)**.** *Assume* $\Gamma \subseteq \Gamma'$ *and* $\overline{\Theta} \vdash \Gamma'$ ctx.

(1) If $\Theta;\Gamma \vdash h \Rightarrow P$ *then* $\Theta;\Gamma' \vdash h \Rightarrow P$.

(2) If $\Theta;\Gamma \vdash g \Rightarrow \uparrow P$ *then* $\Theta;\Gamma' \vdash g \Rightarrow \uparrow P$.

(3) If $\Theta;\Gamma \vdash v \Leftarrow P$ *then* $\Theta;\Gamma' \vdash v \Leftarrow P$.

(4) If $\Theta;\Gamma \vdash e \Leftarrow N$ *then* $\Theta;\Gamma' \vdash e \Leftarrow N$.

(5) If $\Theta;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$ *then* $\Theta;\Gamma';[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$.

(6) If $\Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P$ *then* $\Theta;\Gamma';[N] \vdash s \Rightarrow \uparrow P$.

Moreover, each of the consequent derivations have the same structure and height as the given derivation.

Proof. By mutual induction on the structure of the given typing derivation. □

Lemma C.65 (Prog.-Level Subs. Weakening)**.**
If $\Theta;\Gamma_0 \vdash \sigma : \Theta;\Gamma$ *and* $\Gamma_0 \subseteq \Gamma_0'$ *and* $\overline{\Theta} \vdash \Gamma'$ ctx *then* $\Theta;\Gamma_0' \vdash \sigma : \Theta;\Gamma$.

Proof. By structural induction on the given substitution typing derivation, case analyzing its concluding rule. Similar to Lemma C.42 (Ix.-Level Subs. Weakening). The SubstVal case uses Lemma C.64 (Prog.-Level Weakening). □

Lemma C.66 (Id. Subst. Extension)**.** *Assume* $\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma$. *If* $\overline{\Theta} \vdash R$ type$[\xi]$ *and* $x \notin dom(\Gamma) \cup dom(\Gamma_0)$ *then* $\Theta_0;\Gamma_0, x : [\lfloor\sigma\rfloor]R \vdash \sigma, x : [\lfloor\sigma\rfloor]R / x : \Theta;\Gamma, x : R$.

Proof.

$\Theta_0 \vdash \lfloor\sigma\rfloor : \Theta$	By Lemma C.1 (Filter Out Prog. Vars. Syn)
$\overline{\Theta_0} \vdash [\lfloor\sigma\rfloor]R\ \mathsf{type}[_]$	By Lemma C.51 (WF Syn. Substitution)
$\Theta_0;\Gamma_0,x : [\lfloor\sigma\rfloor]R \vdash \sigma : \Theta;\Gamma$	By Lemma C.65 (Prog.-Level Subs. Weakening)
$\Theta_0 \vdash [\lfloor\sigma\rfloor]R \leq^+ [\lfloor\sigma\rfloor]R$	By Lemma C.62 (Subtyping Reflexive)
$\Theta_0;\Gamma_0,x : [\lfloor\sigma\rfloor]R \vdash x \Leftarrow [\lfloor\sigma\rfloor]R$	By Decl$\Leftarrow$Var
$\Theta_0;\Gamma_0,x : [\lfloor\sigma\rfloor]R \vdash \sigma,x : [\lfloor\sigma\rfloor]R/x : \Theta;\Gamma,x : R$	By SubstVal

□

Lemma C.67 (Typing Consequence)**.** *Assume* $\Theta_1 \vdash \varphi\ \mathsf{true}$.

(1) If $\Theta_1,\varphi,\Theta_2;\Gamma \vdash h \Rightarrow P$ *then* $\Theta_1,\Theta_2;\Gamma \vdash h \Rightarrow P$.

(2) If $\Theta_1,\varphi,\Theta_2;\Gamma \vdash g \Rightarrow \uparrow P$ *then* $\Theta_1,\Theta_2;\Gamma \vdash g \Rightarrow \uparrow P$.

(3) If $\Theta_1,\varphi,\Theta_2;\Gamma \vdash v \Leftarrow P$ *then* $\Theta_1,\Theta_2;\Gamma \vdash v \Leftarrow P$.

(4) If $\Theta_1,\varphi,\Theta_2;\Gamma \vdash e \Leftarrow N$ *then* $\Theta_1,\Theta_2;\Gamma \vdash e \Leftarrow N$.

(5) If $\Theta_1,\varphi,\Theta_2;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$ *then* $\Theta_1,\Theta_2;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$.

(6) If $\Theta_1,\varphi,\Theta_2;\Gamma;[N] \vdash s \Rightarrow \uparrow P$ *then* $\Theta_1,\Theta_2;\Gamma;[N] \vdash s \Rightarrow \uparrow P$.

Proof. By mutual induction on the program typing derivation, and case analysis on the rule concluding the latter, using Lemma C.61 (Subtyping Consequence) for the Decl$\Leftarrow$Var case of the value typing part and for the Decl$\Leftarrow$`rec` case of the expression typing part, using Lemma C.29 (Consequence) when necessary (e.g., for the Decl$\Leftarrow\wedge$ case of the value typing part). □

Lemma C.68 (Index-Level Substitution)**.** *Assume* $\Theta_0;\cdot \vdash \sigma : \Theta;\cdot$.

(1) If $\Theta;\Gamma \vdash h \Rightarrow P$ *then* $\Theta_0;[\sigma]\Gamma \vdash [\sigma]^{\mathsf{h}}h \Rightarrow [\sigma]P$.

(2) If $\Theta;\Gamma \vdash g \Rightarrow \uparrow P$ *then* $\Theta_0;[\sigma]\Gamma \vdash [\sigma]g \Rightarrow [\sigma]\uparrow P$.

(3) If $\Theta;\Gamma \vdash v \Leftarrow P$ *then* $\Theta_0;[\sigma]\Gamma \vdash [\sigma]v \Leftarrow [\sigma]P$.

(4) If $\Theta;\Gamma \vdash e \Leftarrow N$ *then* $\Theta_0;[\sigma]\Gamma \vdash [\sigma]e \Leftarrow [\sigma]N$.

(5) If $\Theta;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$ *then* $\Theta_0;[\sigma]\Gamma;[[\sigma]P] \vdash [\sigma]\{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow [\sigma]N$.

(6) If $\Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P$ *then* $\Theta_0;[\sigma]\Gamma;[[\sigma]N] \vdash [\sigma]s \Rightarrow [\sigma]\uparrow P$.

Moreover, the structure and height of the consequent derivation are the same as those of the given derivation.

Proof. By mutual induction on the structure of the given program typing derivation. Case analyze rules concluding the given typing derivation. Each case is straightforward. Use Lemma C.51 (WF Syn. Substitution), Lemma C.59 (Sub. Syn. Subs), Lemma C.38 (Type-/Functor Barendregt), Lemma C.17 (Ix. Syntactic Substitution), Lemma C.30 (Prop. Truth Syn. Subs), Lemma C.31 (Subst. Inconsistent), Lemma C.57 (Unrolling Syntactic Substitution), Lemma C.45 (Ix.-Level Id. Subs. Extension), and Lemma C.13 (Ix. Id. Subst) as needed. Note that, given $\Theta;\Gamma \vdash \mathscr{J}$ or $\Theta;\Gamma;[P] \vdash \mathscr{J}$, for every $(x : A) \in \Gamma$ where $\overline{\Theta} \vdash A$ type$[_]$ we have $\overline{\Theta_0} \vdash [\sigma]A$ type$[_]$ by Lemma C.51 (WF Syn. Substitution). The structure/height condition holds immediately in the base cases; it holds in the inductive cases by the i.h. and the fact that we only re-apply the same rule as the given case. □

Lemma C.69 (Ix. Equiv. Symmetric)**.**

(1) If $\Theta \vdash u \equiv t : \tau$ *then* $\Theta \vdash t \equiv u : \tau$.

(2) If $\Theta;[\tau] \vdash \boldsymbol{u} \equiv \boldsymbol{t} : \kappa$ *then* $\Theta;[\tau] \vdash \boldsymbol{t} \equiv \boldsymbol{u} : \kappa$.

Proof. By mutual induction on the structure of the given equivalence derivation, using Lemma C.27 (Prop. Truth Equiv. Relation) in one case. □

Lemma C.70 (Equiv. Resp. Prp. Truth)**.**

If $\Theta \vdash u$ true *and* $\Theta \vdash u \equiv t : \mathbb{B}$ *then* $\Theta \vdash t$ true*.*

Proof. Follows from Lemma C.33 (Ix. Equiv. Sound) and PropTrue. □

Lemma C.71 (Ctx. Equiv. Compat.)**.** *Assume* $\Theta_1 \vdash \Theta \equiv \Theta'$ ctx*.*

(1) If $\vdash \delta : \Theta_1, \Theta, \Theta_2$ *then* $\vdash \delta : \Theta_1, \Theta', \Theta_2$*.*

(2) If $\Theta_1, \Theta, \Theta_2 \vdash t$ true *then* $\Theta_1, \Theta', \Theta_2 \vdash t$ true*.*

(3) If $\Theta_1, \Theta, \Theta_2 \vdash u \equiv t : \tau$ *then* $\Theta_1, \Theta', \Theta_2 \vdash u \equiv t : \tau$*.*

(4) If $\Theta_1, \Theta, \Theta_2; [\tau] \vdash \boldsymbol{u} \equiv \boldsymbol{t} : \kappa$ *then* $\Theta_1, \Theta', \Theta_2; [\tau] \vdash \boldsymbol{u} \equiv \boldsymbol{t} : \kappa$*.*

(5) If $\Theta_1, \Theta, \Theta_2 \vdash A \leq^{\pm} B$ *then* $\Theta_1, \Theta', \Theta_2 \vdash A \leq^{\pm} B$*.*

(6) If $\Theta_1, \Theta, \Theta_2; \Gamma \vdash h \Rightarrow P$ *then* $\Theta_1, \Theta', \Theta_2; \Gamma \vdash h \Rightarrow P$*.*

(7) If $\Theta_1, \Theta, \Theta_2; \Gamma \vdash g \Rightarrow \uparrow P$ *then* $\Theta_1, \Theta', \Theta_2; \Gamma \vdash g \Rightarrow \uparrow P$*.*

(8) If $\Theta_1, \Theta, \Theta_2; \Gamma \vdash v \Leftarrow P$ *then* $\Theta_1, \Theta', \Theta_2; \Gamma \vdash v \Leftarrow P$*.*

(9) If $\Theta_1, \Theta, \Theta_2; \Gamma \vdash e \Leftarrow N$ *then* $\Theta_1, \Theta', \Theta_2; \Gamma \vdash e \Leftarrow N$*.*

(10) If $\Theta_1, \Theta, \Theta_2; \Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$
then $\Theta_1, \Theta', \Theta_2; \Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$*.*

(11) If $\Theta_1, \Theta, \Theta_2; \Gamma; [N] \vdash s \Rightarrow \uparrow P$ *then* $\Theta_1, \Theta', \Theta_2; \Gamma; [N] \vdash s \Rightarrow \uparrow P$*.*

Further, none of the consequent derivations change in structure or height.

Proof. All necessary proposition-independent derivations (derivations under a Ξ) with $\overline{\Theta'}$ instead of $\overline{\Theta}$ follow from $\overline{\Theta} = \overline{\Theta'}$ (which is easy to check) and weakening.

(1) By lexicographic induction, first, on the structure of Θ_2, and second, on the structure of Θ.

(2)

$\vdash \delta : \Theta_1, \Theta', \Theta_2$	Assume
$\Theta_1 \vdash \Theta' \equiv \Theta$ ctx	By repeated Lemma C.69 (Ix. Equiv. Symmetric) (and rules)
$\vdash \delta : \Theta_1, \Theta, \Theta_2$	By part (2)
$\Theta_1, \Theta, \Theta_2 \vdash t$ true	Given
$[\![t]\!]_\delta = \{\bullet\}$	By inversion on PropTrue
$\Theta_1, \Theta', \Theta_2 \vdash t$ true	By PropTrue

(3) By structural induction on the index equivalence derivation, using part (2) as needed. This part is mutually recursive with part (4).

(4) By structural induction on the given index spine equivalence derivation. This part is mutually recursive with part (3).

(5) By structural induction on the given subtyping derivation, using parts (2) and (4) as needed.

Parts (6) through (11) are proved by mutual induction on the structure of the program typing derivation, using previous parts as needed. □

Lemma C.72 (Equal Ix. Equalities)**.** *If* $\Theta \vdash u = u'$ true *and* $\Theta \vdash t = t'$ true *then* $\Theta \vdash (u = t) = (u' = t')$ true.

Proof. Follows from definition of PropTrue and Lemma C.27 (Prop. Truth Equiv. Relation). □

Lemma C.73 (Unroll Subst. Undo)**.**

If $\overset{\mathsf{d}\cdot}{\div}\Xi \vdash \sigma : {}^{\mathsf{d}}\Xi_1$ *and* $\Xi, {}^{\mathsf{d}}\Xi' \vdash \wr\overrightarrow{q \Rightarrow [\sigma]u}; \hat{P}; \mathscr{M}(F)\wr \mathrel{\hat{=}} {}^{\mathsf{d}}\Theta; R$

and $\Xi \vdash \mathscr{M}(F)$ $\mathsf{msmts}[_]$ *and* $\Xi, {}^{\mathsf{d}}\Xi_1, {}^{\mathsf{d}}\Xi' \vdash q \Rightarrow u : \hat{P}(\tau) \Rightarrow \tau$

then there exist ${}^{\mathsf{d}}\Theta'$ *and* R' *such that* $\Xi, {}^{\mathsf{d}}\Xi' \vdash \wr\overrightarrow{q \Rightarrow u}; \hat{P}; \mathscr{M}(F)\wr \mathrel{\hat{=}} {}^{\mathsf{d}}\Theta'; R'$

and $[\sigma]{}^{\mathsf{d}}\Theta' = {}^{\mathsf{d}}\Theta$ *and* $[\sigma]R' = R$ *and* $\overline{{}^{\mathsf{d}}\Theta'} = \overline{{}^{\mathsf{d}}\Theta}$*;*

moreover, if $\Xi, {}^{\mathsf{d}}\Xi_1 \vdash R_1 \wedge \overrightarrow{\psi}_1$ $\mathsf{type}[\xi_1]$ *and* $\xi_1 - \overset{\mathsf{d}\cdot}{\div}\Xi \vdash {}^{\mathsf{d}}\Xi_1$ det

then $\Xi, {}^{\mathsf{d}}\Xi' \vdash \wr\overrightarrow{(\mathrm{pk}({}^{\mathsf{d}}\Xi_1, \top), q) \Rightarrow u}; \underline{\exists {}^{\mathsf{d}}\Xi_1. R_1 \wedge \overrightarrow{\psi}_1} \otimes \hat{P}; \mathscr{M}(F)\wr \mathrel{\hat{=}} {}^{\mathsf{d}}\Xi_1, {}^{\mathsf{d}}\Theta', \overrightarrow{\psi}_1; R_1 \times R'.$

Proof. By structural induction on $\Xi, {}^{\mathsf{d}}\Xi' \vdash \wr\overrightarrow{q \Rightarrow [\sigma]u}; \hat{P}; \mathscr{M}(F)\wr \mathrel{\hat{=}} {}^{\mathsf{d}}\Theta; R$.

Use Lemma C.20 (Ix. Barendregt) and Lemma C.55 (Unrolling Output WF). □

Lemma C.74. *If* $\overrightarrow{a \overset{\mathsf{d}\cdot}{\div} \tau} = \overrightarrow{a} \overset{\mathsf{d}\cdot}{\div} \mathscr{M}(F)$ *and* $\Xi \vdash \mathscr{M}(F) \geq_{\equiv} \mathscr{M}'(F')$

and $\Xi; \Xi''; zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash \mathscr{O}_2 \rightsquigarrow \breve{\Xi}; \mathscr{M}_1(F); \mathscr{O}_1$

and $\mathscr{O}_2$ *is a subterm of an unrolling output of principal functor* $\hat{I}$

and $\rho = \Xi'/\breve{\Xi}$ *is a variable renaming (where* Ξ' *is fresh)*

and (if $\mathscr{O}_2$ *is not a type or measurements)* $\mathscr{O}_2$ *is judg. equivalent to* $\mathscr{O}_2'$ *under* $(\Xi, \Xi'', \overrightarrow{a \overset{\mathsf{d}\cdot}{\div} \tau, a\,\mathrm{Id}})$

or (if $\mathscr{O}_2$ *is a type)* $\mathscr{D} :: \Xi, \Xi'', \overrightarrow{a \overset{\mathsf{d}\cdot}{\div} \tau, a\,\mathrm{Id}} \vdash \mathscr{O}_2 \leq^{+} \mathscr{O}_2'$

and every subderivation $\Xi, \Xi'', \overrightarrow{a \overset{\mathsf{d}\cdot}{\div} \tau, a\,\mathrm{Id}} \vdash \mathscr{M}_2 \geq \mathscr{M}_2'$ *that is a premise of* $\leq^{+}\mu$ *in* $\mathscr{D}$

satisfies $\Xi, \Xi'', \overrightarrow{a \overset{\mathsf{d}\cdot}{\div} \tau, a\,\mathrm{Id}} \vdash \mathscr{M}_2 \geq_{\equiv} \mathscr{M}_2'$

or (if $\mathscr{O}_2$ *is measurements)* $\Xi, \Xi'', \overrightarrow{a \overset{\mathsf{d}\cdot}{\div} \tau, a\,\mathrm{Id}} \vdash \mathscr{O}_2 \geq_{\equiv} \mathscr{O}_2'$

then $\Xi; \Xi''; zip(\overrightarrow{a})(\mathscr{M}'(F')) \vdash \mathscr{O}_2' \rightsquigarrow \breve{\Xi}'; \mathscr{M}_1'(F); \mathscr{O}_1'$

and $\Xi, \Xi', \Xi'' \vdash [\rho]\mathscr{M}_1(F) \geq_{\equiv} [\rho]\mathscr{M}_1'(F')$

and (if $\mathscr{O}_2$ *is not a type or measurements)* $[\rho]\mathscr{O}_1$ *is judg. equivalent to* $[\rho]\mathscr{O}_1'$ *under* (Ξ, Ξ', Ξ'')

or (if $\mathscr{O}_2$ *is a type)* $\mathscr{D}' :: \Xi, \Xi', \Xi'' \vdash [\rho]\mathscr{O}_1 \leq^{+} [\rho]\mathscr{O}_1'$

and every subderivation $\Xi, \Xi', \Xi'' \vdash \mathscr{M}_2' \geq \mathscr{M}_2''$ *that is a premise of* $\leq^+\mu$ *in* $\mathscr{D}$
satisfies $\Xi, \Xi', \Xi'' \vdash \mathscr{M}_2' \geq_{\equiv} \mathscr{M}_2''$
or (if $\mathscr{O}_2$ *is measurements)* $\Xi, \Xi', \Xi'' \vdash [\rho]\mathscr{O}_1 \geq_{\equiv} [\rho]\mathscr{O}_1'$.

Proof. By structural induction on $\mathscr{O}_2$. Use congruence lemmas and reflexivity lemmas. □

Lemma C.75 (Unroll to Supertype)**.** *If* ${}^{\mathsf{d}}\Xi, \Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Theta; R$
and $\Xi \vdash \overrightarrow{\beta}; G \leq_{\overrightarrow{\tau}} \overrightarrow{\beta'}; G'$ *and* ${}^{\mathsf{d}}\Xi \vdash \mathscr{M}(F) \geq_{\equiv} \mathscr{M}'(F')$
then there exist $\mathscr{D}$, ${}^{\mathsf{d}}\Theta'$, *and* R' *such that* ${}^{\mathsf{d}}\Xi, \Xi \vdash \wr \overrightarrow{\beta'}; G'; \mathscr{M}'(F') \wr \doteq {}^{\mathsf{d}}\Theta'; R'$
and $\mathscr{D} :: {}^{\mathsf{d}}\Xi, \Xi \vdash \exists^{\mathsf{d}}\Theta. R \wedge {}^{\mathsf{d}}\Theta \leq^+ \exists^{\mathsf{d}}\Theta'. R' \wedge {}^{\mathsf{d}}\Theta'$;
moreover, if $G = \hat{I}$
then ${}^{\mathsf{d}}\Xi, \overset{\mathsf{d}\cdot}{\div}\Xi, \overline{{}^{\mathsf{d}}\Theta} \vdash {}^{\mathsf{d}}\Theta - \overline{{}^{\mathsf{d}}\Theta} \equiv {}^{\mathsf{d}}\Theta' - \overline{{}^{\mathsf{d}}\Theta'} : \mathbb{B}$
and every subderivation ${}^{\mathsf{d}}\Xi, \overset{\mathsf{d}\cdot}{\div}\Xi, \overline{{}^{\mathsf{d}}\Theta} \vdash \mathscr{M}_1 \geq \mathscr{M}_1'$ *that is a premise of* $\leq^+\mu$ *in* $\mathscr{D}$
satisfies ${}^{\mathsf{d}}\Xi, \overset{\mathsf{d}\cdot}{\div}\Xi, \overline{{}^{\mathsf{d}}\Theta} \vdash \mathscr{M}_1 \geq_{\equiv} \mathscr{M}_1'$,
and the $\leq^+\exists$R*-witness* ${}^{\mathsf{d}}\Xi, \overset{\mathsf{d}\cdot}{\div}\Xi, \overline{{}^{\mathsf{d}}\Theta} \vdash \sigma : \overline{{}^{\mathsf{d}}\Theta'}$ *of* $\mathscr{D}$
is the identity substitution on $\overline{{}^{\mathsf{d}}\Theta'}$ *and* $dom({}^{\mathsf{d}}\Theta') = dom({}^{\mathsf{d}}\Theta)$.

Proof. By induction on the height of ${}^{\mathsf{d}}\Xi, \Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Theta; R$. For each case, the types in ${}^{\mathsf{d}}\Xi, \Xi \vdash \exists^{\mathsf{d}}\Theta. R \wedge {}^{\mathsf{d}}\Theta \leq^+ \exists^{\mathsf{d}}\Theta'. R' \wedge {}^{\mathsf{d}}\Theta'$ are well-formed by Lemma C.55 (Unrolling Output WF).

Case analyze the given unrolling derivation and consider ${}^{\mathsf{d}}\Xi, \Xi \vdash \overrightarrow{\beta}; G \leq_\tau \overrightarrow{\beta'}; G'$ subcases. Use Lemma C.73 (Unroll Subst. Undo) in the $\wr$Const$\wr$ subcase. Use Lemma C.37 (Ix. App. Respects Equivalence), Lemma C.72 (Equal Ix. Equalities), Lemma C.69 (Ix. Equiv. Symmetric), Lemma C.28 (Assumption), Lemma C.71 (Ctx. Equiv. Compat), and Lemma C.36 (Ix. Equiv. Weakening) in the $\wr I \wr$ case.

The $\wr\mathrm{Id}\wr$ case relies on using the "moreover" conditions. These conditions together with the definition of judgmental index equivalence (so that Id variables in judgmentally

equivalent index terms must struturally be in the same positions) makes the needed $\rightsquigarrow$ premises have outputs with the needed relations to the original $\rightsquigarrow$ premises outputs: Lemma C.74. □

Lemma C.76. *If* $\overrightarrow{a \overset{\mathsf{d}}{\div} \tau} = \overrightarrow{a} \overset{\mathsf{d}}{\div} \mathscr{M}(F)$ *and* $\Xi \vdash \mathscr{M}(F) \geq_{\equiv} \mathscr{M}'(F')$
and $\Xi; \Xi''; zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash \mathscr{O}_2 \rightsquigarrow \breve{\Xi}; \mathscr{M}_1(F); \mathscr{O}_1$
and $\mathscr{O}_2$ *is a subterm of an unrolling output of principal functor* $\hat{I}$
and $\rho = \Xi'/\breve{\Xi}$ *is a variable renaming (where* Ξ' *is fresh)*
and (if $\mathscr{O}_2$ *is not a type or measurements)* $\mathscr{O}_2'$ *is judg. equivalent to* $\mathscr{O}_2$ *under* $(\Xi, \Xi'', \overrightarrow{a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id}})$
or (if $\mathscr{O}_2$ *is a type)* $\mathscr{D} :: \Xi, \Xi'', \overrightarrow{a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id}} \vdash \mathscr{O}_2' \leq^{+} \mathscr{O}_2$
and every subderivation $\Xi, \Xi'', \overrightarrow{a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id}} \vdash \mathscr{M}_2' \geq \mathscr{M}_2$ *that is a premise of* $\leq^{+}\mu$ *in* $\mathscr{D}$
satisfies $\Xi, \Xi'', \overrightarrow{a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id}} \vdash \mathscr{M}_2' \geq_{\equiv} \mathscr{M}_2$
or (if $\mathscr{O}_2$ *is measurements)* $\Xi, \Xi'', \overrightarrow{a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id}} \vdash \mathscr{O}_2' \geq_{\equiv} \mathscr{O}_2$
then $\Xi; \Xi''; zip(\overrightarrow{a})(\mathscr{M}'(F')) \vdash \mathscr{O}_2' \rightsquigarrow \breve{\Xi}'; \mathscr{M}_1'(F); \mathscr{O}_1'$
and $\Xi, \Xi', \Xi'' \vdash [\rho]\mathscr{M}_1'(F') \geq_{\equiv} [\rho]\mathscr{M}_1(F)$
and (if $\mathscr{O}_2$ *is not a type or measurements)* $[\rho]\mathscr{O}_1'$ *is judg. equivalent to* $[\rho]\mathscr{O}_1$ *under* (Ξ, Ξ', Ξ'')
or (if $\mathscr{O}_2$ *is a type)* $\mathscr{D}' :: \Xi, \Xi', \Xi'' \vdash [\rho]\mathscr{O}_1' \leq^{+} [\rho]\mathscr{O}_1$
and every subderivation $\Xi, \Xi', \Xi'' \vdash \mathscr{M}_2'' \geq \mathscr{M}_2'$ *that is a premise of* $\leq^{+}\mu$ *in* $\mathscr{D}$
satisfies $\Xi, \Xi', \Xi'' \vdash \mathscr{M}_2'' \geq_{\equiv} \mathscr{M}_2'$
or (if $\mathscr{O}_2$ *is measurements)* $\Xi, \Xi', \Xi'' \vdash [\rho]\mathscr{O}_1' \geq_{\equiv} [\rho]\mathscr{O}_1$.

Proof. By structural induction on $\mathscr{O}_2$. Use congruence lemmas and reflexivity lemmas. □

Lemma C.77 (Unroll to Subtype)**.** *If* $^{\mathsf{d}}\Xi, \Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Theta; R$
and $\Xi \vdash \overrightarrow{\beta'}; G' \leq_{\overrightarrow{\tau}} \overrightarrow{\beta}; G$ *and* $^{\mathsf{d}}\Xi \vdash \mathscr{M}'(F') \geq_{\equiv} \mathscr{M}(F)$
then there exist $\mathscr{D}$, $^{\mathsf{d}}\Theta'$, *and* R' *such that* $^{\mathsf{d}}\Xi, \Xi \vdash \wr \overrightarrow{\beta'}; G'; \mathscr{M}'(F') \wr \doteq {}^{\mathsf{d}}\Theta'; R'$
and $\mathscr{D} :: {}^{\mathsf{d}}\Xi, \Xi \vdash \exists^{\mathsf{d}}\Theta'.R' \wedge {}^{\mathsf{d}}\Theta' \leq^{+} \exists^{\mathsf{d}}\Theta.R \wedge {}^{\mathsf{d}}\Theta;$

moreover, if $G = \hat{I}$

then ${}^{\mathsf{d}}\Xi, \overset{\mathsf{d}\cdot}{\div}\Xi, \overline{{}^{\mathsf{d}}\Theta} \vdash {}^{\mathsf{d}}\Theta' - \overline{{}^{\mathsf{d}}\Theta'} \equiv {}^{\mathsf{d}}\Theta - \overline{{}^{\mathsf{d}}\Theta} : \mathbb{B}$

and every subderivation ${}^{\mathsf{d}}\Xi, \overset{\mathsf{d}\cdot}{\div}\Xi, \overline{{}^{\mathsf{d}}\Theta} \vdash \mathscr{M}_1' \geq \mathscr{M}_1$ *that is a premise of* $\leq^+\mu$ *in* $\mathscr{D}$

satisfies ${}^{\mathsf{d}}\Xi, \overset{\mathsf{d}\cdot}{\div}\Xi, \overline{{}^{\mathsf{d}}\Theta} \vdash \mathscr{M}_1' \geq_{\equiv} \mathscr{M}_1$,

and the $\leq^+\exists$R*-witness* ${}^{\mathsf{d}}\Xi, \overset{\mathsf{d}\cdot}{\div}\Xi, \overline{{}^{\mathsf{d}}\Theta} \vdash \sigma : \overline{{}^{\mathsf{d}}\Theta}$ *of* $\mathscr{D}$

is the identity substitution on $\overline{{}^{\mathsf{d}}\Theta}$ *and* $dom({}^{\mathsf{d}}\Theta) = dom({}^{\mathsf{d}}\Theta')$.

Proof. By induction on the height of the given unrolling derivation. For each case, the types in ${}^{\mathsf{d}}\Xi, \Xi \vdash \exists^{\mathsf{d}}\Theta'.R' \wedge {}^{\mathsf{d}}\Theta' \leq^+ \exists^{\mathsf{d}}\Theta.R \wedge {}^{\mathsf{d}}\Theta$ are well-formed by Lemma C.55 (Unrolling Output WF).

Similar to Lemma C.75 (Unroll to Supertype).

Case analyze the given unrolling derivation and consider ${}^{\mathsf{d}}\Xi, \Xi \vdash \overrightarrow{\beta'}; G' \leq_\tau \overrightarrow{\beta}; G$ subcases. Use Lemma C.57 (Unrolling Syntactic Substitution) in the ⦇Const⦈ case. Use Lemma C.37 (Ix. App. Respects Equivalence), Lemma C.72 (Equal Ix. Equalities), Lemma C.69 (Ix. Equiv. Symmetric), Lemma C.28 (Assumption), Lemma C.71 (Ctx. Equiv. Compat), and Lemma C.36 (Ix. Equiv. Weakening) in the ⦇I⦈ case.

The ⦇Id⦈ case relies on using the "moreover" conditions. These conditions together with the definition of judgmental index equivalence (so that Id variables in judgmentally equivalent index terms must struturally be in the same positions) makes the needed $\rightsquigarrow$ premises have outputs with the needed relations to the original $\rightsquigarrow$ premises outputs: Lemma C.76. □

Lemma C.78 (Unroll Sublist)**.**

If $\Xi \vdash ⦇\overrightarrow{\beta}; G; \mathscr{M}(F)⦈ \doteq {}^{\mathsf{d}}\Theta; R$ *and* $zip(\overrightarrow{\beta'})(\mathscr{M}'(F)) \subseteq zip(\overrightarrow{\beta})(\mathscr{M}(F))$

then $\Xi \vdash ⦇\overrightarrow{\beta'}; G; \mathscr{M}'(F)⦈ \doteq {}^{\mathsf{d}}\Theta'; R'$ *and* $\Xi \vdash \exists^{\mathsf{d}}\Theta.R \wedge {}^{\mathsf{d}}\Theta \leq^+ \exists^{\mathsf{d}}\Theta'.R' \wedge {}^{\mathsf{d}}\Theta'$

and if $G = \hat{I}$ *then* ${}^{\mathsf{d}}\Theta' \subseteq {}^{\mathsf{d}}\Theta$ *and* $R' \subseteq R$ *where the latter* ($R' \subseteq R$) *is defined by*

- $1 \subseteq 1$
- *if* $R' \subseteq R$ *and* $\mathscr{M}_1'(F) \subseteq \mathscr{M}_1(F)$
  then $\{\nu : \mu F \mid \mathscr{M}_1'(F)\} \times R' \subseteq \{\nu : \mu F \mid \mathscr{M}_1(F)\} \times R$

Proof. By structural induction on $\Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta; R$.

- **Case**

$$\dfrac{\begin{array}{ll} \overrightarrow{\beta} \circ \mathsf{inj}_1 \doteq \overrightarrow{\beta_1} & \Xi \vdash \wr \overrightarrow{\beta_1}; G_1; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta_1; R_1 \\ \overrightarrow{\beta} \circ \mathsf{inj}_2 \doteq \overrightarrow{\beta_2} & \Xi \vdash \wr \overrightarrow{\beta_2}; G_2; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta_2; R_2 \end{array}}{\Xi \vdash \wr \overrightarrow{\beta}; G_1 \oplus G_2; \mathscr{M}(F) \int \doteq \cdot; (\exists^{\mathsf{d}}\Theta_1.\, (R_1 \wedge {}^{\mathsf{d}}\Theta_1)) + (\exists^{\mathsf{d}}\Theta_2.\, (R_2 \wedge {}^{\mathsf{d}}\Theta_2))}\, \wr\oplus\int$$

$\overrightarrow{\beta} \circ \mathsf{inj}_j \doteq \overrightarrow{\beta_j}$ Premises

$zip(\overrightarrow{\beta'})(\mathscr{M}'(F)) \subseteq zip(\overrightarrow{\beta})(\mathscr{M}(F))$ Given

$\overrightarrow{\beta'} \circ \mathsf{inj}_j \doteq \overrightarrow{\beta_j'}$ Straightforward

$zip(\overrightarrow{\beta_j'})(\mathscr{M}'(F)) \subseteq zip(\overrightarrow{\beta_j})(\mathscr{M}(F))$ ″

$\Xi \vdash \wr \overrightarrow{\beta_j}; G_j; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta_j; R_j$ Subderivations

$\Xi \vdash \wr \overrightarrow{\beta_j'}; G_j; \mathscr{M}'(F) \int \doteq {}^{\mathsf{d}}\Theta_j'; R_j'$ By i.h. (twice)

$\Xi \vdash \exists^{\mathsf{d}}\Theta_j.\, R_j \wedge {}^{\mathsf{d}}\Theta_j \leq^{+} \exists^{\mathsf{d}}\Theta_j'.\, R_j' \wedge {}^{\mathsf{d}}\Theta_j'$ ″

$\Xi \vdash \wr \overrightarrow{\beta'}; G_1 \oplus G_2; \mathscr{M}'(F) \int \doteq \cdot; (\exists^{\mathsf{d}}\Theta_1'.\, R_1' \wedge {}^{\mathsf{d}}\Theta_1') + (\exists^{\mathsf{d}}\Theta_2'.\, R_2' \wedge {}^{\mathsf{d}}\Theta_2')$ By $\wr\oplus\int$

$\Xi \vdash (\exists^{\mathsf{d}}\Theta_1.\, R_1 \wedge {}^{\mathsf{d}}\Theta_1) + (\exists^{\mathsf{d}}\Theta_2.\, R_2 \wedge {}^{\mathsf{d}}\Theta_2) \leq^{+} (\exists^{\mathsf{d}}\Theta_1'.\, R_1' \wedge {}^{\mathsf{d}}\Theta_1') + (\exists^{\mathsf{d}}\Theta_2'.\, R_2' \wedge {}^{\mathsf{d}}\Theta_2') \quad \leq^{+}+$

- **Case**

$$\dfrac{\overrightarrow{\beta} \rightsquigarrow \overrightarrow{\beta_0} \qquad \Xi, {}^{\mathsf{d}}\Xi_1 \vdash \wr \overrightarrow{\beta_0}; \hat{P}; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta_0; R_0}{\Xi \vdash \wr \overrightarrow{\beta}; \underline{\exists^{\mathsf{d}}\Xi_1.\, R_1 \wedge \overrightarrow{\varphi_1}} \otimes \hat{P}; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Xi_1, {}^{\mathsf{d}}\Theta_0, \overrightarrow{\varphi_1}; R_1 \times R_0}\, \wr\mathrm{Const}\int$$

$\overrightarrow{\beta} \rightsquigarrow \overrightarrow{\beta_0}$	Premise
$zip(\overrightarrow{\beta'})(\mathscr{M}'(F)) \subseteq zip(\overrightarrow{\beta})(\mathscr{M}(F))$	Given
$\overrightarrow{\beta'} \rightsquigarrow \overrightarrow{\beta'_0}$	Straightforward
$zip(\overrightarrow{\beta'_0})(\mathscr{M}'(F)) \subseteq zip(\overrightarrow{\beta_0})(\mathscr{M}(F))$	$''$
$\Xi, {}^{\mathsf{d}}\Xi_1 \vdash \wr \overrightarrow{\beta_0}; \hat{P}; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta_0; R_0$	Subderivation
$\Xi, {}^{\mathsf{d}}\Xi_1 \vdash \wr \overrightarrow{\beta'_0}; \hat{P}; \mathscr{M}'(F) \int \doteq {}^{\mathsf{d}}\Theta'_0; R'_0$	By i.h.
$\Xi, {}^{\mathsf{d}}\Xi_1 \vdash \exists^{\mathsf{d}}\Theta_0. R_0 \wedge {}^{\mathsf{d}}\Theta_0 \leq^+ \exists^{\mathsf{d}}\Theta'_0. R'_0 \wedge {}^{\mathsf{d}}\Theta'_0$	$''$

☞ $\Xi \vdash \wr \overrightarrow{\beta'}; \underline{\exists^{\mathsf{d}}\Xi_1. R_1 \wedge \overrightarrow{\varphi_1}} \otimes \hat{P}; \mathscr{M}'(F) \int \doteq {}^{\mathsf{d}}\Xi_1, {}^{\mathsf{d}}\Theta'_0, \overrightarrow{\varphi_1}; R_1 \times R'_0$ By $\wr$Const$\int$

$\Xi, {}^{\mathsf{d}}\Xi_1, \overline{{}^{\mathsf{d}}\Theta_0}, {}^{\mathsf{d}}\Theta_0 - \overline{{}^{\mathsf{d}}\Theta_0} \vdash R_0 \leq^+ \exists^{\mathsf{d}}\Theta'_0. R'_0 \wedge {}^{\mathsf{d}}\Theta'_0$	By inversion
$\Xi, {}^{\mathsf{d}}\Xi_1, \overline{{}^{\mathsf{d}}\Theta_0}, {}^{\mathsf{d}}\Theta_0 - \overline{{}^{\mathsf{d}}\Theta_0} \vdash R_0 \leq^+ [\sigma]R'_0$	By inversion
$\Xi, {}^{\mathsf{d}}\Xi_1, \overline{{}^{\mathsf{d}}\Theta_0}, {}^{\mathsf{d}}\Theta_0 - \overline{{}^{\mathsf{d}}\Theta_0} \vdash \sigma : \overline{{}^{\mathsf{d}}\Theta'_0}$	$''$
$\Xi, {}^{\mathsf{d}}\Xi_1, \overline{{}^{\mathsf{d}}\Theta_0}, {}^{\mathsf{d}}\Theta_0 - \overline{{}^{\mathsf{d}}\Theta_0} \vdash [\sigma]({}^{\mathsf{d}}\Theta'_0 - \overline{{}^{\mathsf{d}}\Theta'_0})\ \mathsf{true}$	$''$

$\Xi, {}^{\mathsf{d}}\Xi_1, \overline{{}^{\mathsf{d}}\Theta_0}, \overrightarrow{\varphi_1}, {}^{\mathsf{d}}\Theta_0 - \overline{{}^{\mathsf{d}}\Theta_0} \vdash R_0 \leq^+ [\sigma]R'_0$	By weakening
$\Xi, {}^{\mathsf{d}}\Xi_1, \overline{{}^{\mathsf{d}}\Theta_0}, \overrightarrow{\varphi_1}, {}^{\mathsf{d}}\Theta_0 - \overline{{}^{\mathsf{d}}\Theta_0} \vdash \sigma : \overline{{}^{\mathsf{d}}\Theta'_0}$	By weakening
$\Xi, {}^{\mathsf{d}}\Xi_1, \overline{{}^{\mathsf{d}}\Theta_0}, \overrightarrow{\varphi_1}, {}^{\mathsf{d}}\Theta_0 - \overline{{}^{\mathsf{d}}\Theta_0} \vdash {}^{\mathsf{d}}\Xi_1/{}^{\mathsf{d}}\Xi_1, \sigma : {}^{\mathsf{d}}\Xi_1, \overline{{}^{\mathsf{d}}\Theta'_0}$	Straightforward
$\Xi, {}^{\mathsf{d}}\Xi_1, \overline{{}^{\mathsf{d}}\Theta_0}, \overrightarrow{\varphi_1}, {}^{\mathsf{d}}\Theta_0 - \overline{{}^{\mathsf{d}}\Theta_0} \vdash [\sigma]({}^{\mathsf{d}}\Theta'_0 - \overline{{}^{\mathsf{d}}\Theta'_0})\ \mathsf{true}$	By weakening
$\Xi, {}^{\mathsf{d}}\Xi_1, \overline{{}^{\mathsf{d}}\Theta_0}, \overrightarrow{\varphi_1}, {}^{\mathsf{d}}\Theta_0 - \overline{{}^{\mathsf{d}}\Theta_0} \vdash R_1 \leq^+ R_1$	By Lemma C.62
$\Xi, {}^{\mathsf{d}}\Xi_1, \overline{{}^{\mathsf{d}}\Theta_0}, \overrightarrow{\varphi_1}, {}^{\mathsf{d}}\Theta_0 - \overline{{}^{\mathsf{d}}\Theta_0} \vdash R_1 \leq^+ [\sigma]R_1$	$FV(R_1) \cap dom(\sigma) = \emptyset$
$\Xi, {}^{\mathsf{d}}\Xi_1, \overline{{}^{\mathsf{d}}\Theta_0}, \overrightarrow{\varphi_1}, {}^{\mathsf{d}}\Theta_0 - \overline{{}^{\mathsf{d}}\Theta_0} \vdash R_1 \times R_0 \leq^+ [\sigma]R_1 \times [\sigma]R_0$	By $\leq^+\times$
$\Xi, {}^{\mathsf{d}}\Xi_1, \overline{{}^{\mathsf{d}}\Theta_0}, \overrightarrow{\varphi_1}, {}^{\mathsf{d}}\Theta_0 - \overline{{}^{\mathsf{d}}\Theta_0} \vdash [\sigma]({}^{\mathsf{d}}\Theta'_0 - \overline{{}^{\mathsf{d}}\Theta'_0}), \overrightarrow{\varphi_1}\ \mathsf{true}$	By Lemma C.29 (Consequence)
$\Xi, {}^{\mathsf{d}}\Xi_1, \overline{{}^{\mathsf{d}}\Theta_0}, \overrightarrow{\varphi_1}, {}^{\mathsf{d}}\Theta_0 - \overline{{}^{\mathsf{d}}\Theta_0} \vdash [\sigma]({}^{\mathsf{d}}\Theta'_0 - \overline{{}^{\mathsf{d}}\Theta'_0}), [\sigma]\overrightarrow{\varphi_1}\ \mathsf{true}$	$FV(\overrightarrow{\varphi_1}) \cap dom(\sigma) = \emptyset$
$\Xi, {}^{\mathsf{d}}\Xi_1, \overline{{}^{\mathsf{d}}\Theta_0}, \overrightarrow{\varphi_1}, {}^{\mathsf{d}}\Theta_0 - \overline{{}^{\mathsf{d}}\Theta_0} \vdash [\sigma]({}^{\mathsf{d}}\Theta'_0 - \overline{{}^{\mathsf{d}}\Theta'_0}, \overrightarrow{\varphi_1})\ \mathsf{true}$	Property of subst.

$\Xi, {}^{\mathsf{d}}\Xi_1, \overline{{}^{\mathsf{d}}\Theta_0}, \overrightarrow{\varphi_1}, {}^{\mathsf{d}}\Theta_0 - \overline{{}^{\mathsf{d}}\Theta_0} \vdash R_1 \times R_0 \leq^+ [\sigma]R_1 \times [\sigma]R_0 \wedge [\sigma]({}^{\mathsf{d}}\Theta'_0 - \overline{{}^{\mathsf{d}}\Theta'_0}, \overrightarrow{\varphi_1})$	$\leq^+\wedge\mathrm{R}$
$\Xi, {}^{\mathsf{d}}\Xi_1, \overline{{}^{\mathsf{d}}\Theta_0}, \overrightarrow{\varphi_1}, {}^{\mathsf{d}}\Theta_0 - \overline{{}^{\mathsf{d}}\Theta_0} \vdash R_1 \times R_0 \leq^+ \exists {}^{\mathsf{d}}\Xi_1, \overline{{}^{\mathsf{d}}\Theta'_0}. R_1 \times R_0 \wedge ({}^{\mathsf{d}}\Theta'_0 - \overline{{}^{\mathsf{d}}\Theta'_0}, \overrightarrow{\varphi_1})$	$\leq^+\exists\mathrm{R}$

By $\leq^+\wedge\mathrm{L}$ and $\leq^+\exists\mathrm{L}$ and exchange,

$$\Xi \vdash \exists {}^{\mathsf{d}}\Xi_1, {}^{\mathsf{d}}\Theta_0, \overrightarrow{\varphi_1}. R_1 \times R_0 \wedge ({}^{\mathsf{d}}\Xi_1, {}^{\mathsf{d}}\Theta_0, \overrightarrow{\varphi_1}) \leq^+ \exists {}^{\mathsf{d}}\Xi_1, \overline{{}^{\mathsf{d}}\Theta'_0}. R_1 \times R_0 \wedge ({}^{\mathsf{d}}\Theta'_0 - \overline{{}^{\mathsf{d}}\Theta'_0}, \overrightarrow{\varphi_1})$$

By definitions,

$$\Xi \vdash \exists {}^{\mathsf{d}}\Xi_1, {}^{\mathsf{d}}\Theta_0, \overrightarrow{\varphi_1}. R_1 \times R_0 \wedge ({}^{\mathsf{d}}\Xi_1, {}^{\mathsf{d}}\Theta_0, \overrightarrow{\varphi_1}) \leq^+ \exists {}^{\mathsf{d}}\Xi_1, {}^{\mathsf{d}}\Theta'_0, \overrightarrow{\varphi_1}. R_1 \times R'_0 \wedge ({}^{\mathsf{d}}\Xi_1, {}^{\mathsf{d}}\Theta'_0, \overrightarrow{\varphi_1})$$

Let ${}^{\mathsf{d}}\Theta' = {}^{\mathsf{d}}\Xi_1, {}^{\mathsf{d}}\Theta'_0, \overrightarrow{\varphi_1}$.

Let $R' = R_1 \times R'_0$.

☞ $\Xi \vdash \exists^{\mathsf{d}}\Theta.\, R \wedge {}^{\mathsf{d}}\Theta \leq^{+} \exists^{\mathsf{d}}\Theta'.\, R' \wedge {}^{\mathsf{d}}\Theta'$ By equalities

- **Case**

$$\frac{\begin{array}{c}\overrightarrow{a \overset{\mathsf{d}\cdot}{\div} \tau} = \overrightarrow{a} \overset{\mathsf{d}\cdot}{\div} \mathscr{M}(F) \\ \Xi, \overrightarrow{a \overset{\mathsf{d}\cdot}{\div} \tau}, a\,\mathrm{Id} \vdash \wr \overrightarrow{q \Rightarrow u}; \hat{I}; \mathscr{M}(F) \wr \mathrel{\mathring{=}} \Xi_1, \overrightarrow{\psi_1}; R_1 \\ \Xi; \Xi_1; zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash \overrightarrow{\psi_1} \rightsquigarrow \check{\Xi}_1; \mathscr{M}_1(F); \overrightarrow{\psi_0} \\ \Xi; \Xi_1; zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash R_1 \rightsquigarrow \check{\Xi}_2; \mathscr{M}_2(F); R_0 \\ \check{\Xi} = \check{\Xi}_1 \cup \check{\Xi}_2 \qquad \mathscr{M}'(F) = \mathscr{M}_1(F) \cup \mathscr{M}_2(F) \\ dom(\Xi') \cap dom(\Xi, \overrightarrow{a \overset{\mathsf{d}\cdot}{\div} \tau}, \Xi_1, \check{\Xi}) = \emptyset \qquad \rho = \Xi'/\check{\Xi} \text{ is a variable renaming}\end{array}}{\Xi \vdash \wr \overrightarrow{(a,q) \Rightarrow u}; \mathrm{Id} \otimes \hat{I}; \mathscr{M}(F) \wr \mathrel{\mathring{=}} \Xi', \Xi_1, [\rho]\overrightarrow{\psi_0}; \{\nu : \mu F \mid [\rho]\mathscr{M}'(F)\} \times [\rho]R_0}\ \wr \mathrm{Id} \wr$$

$$\overrightarrow{a \stackrel{\mathsf{d}\bullet}{\div} \tau} = \overrightarrow{a} \stackrel{\mathsf{d}\bullet}{\div} \mathscr{M}(F) \qquad \text{Premise}$$

$$zip(\overrightarrow{\beta'})(\mathscr{M}'(F)) \subseteq zip(\overrightarrow{(a,q) \Rightarrow u})(\mathscr{M}(F)) \qquad \text{Given}$$

$$zip(\overrightarrow{\beta'})(\mathscr{M}'(F)) = zip(\overrightarrow{(a',q') \Rightarrow u'})(\mathscr{M}'(F)) \qquad \text{Follows from line above}$$

$$\overrightarrow{a'} \stackrel{\mathsf{d}\bullet}{\div} \mathscr{M}'(F) \subseteq \overrightarrow{a} \stackrel{\mathsf{d}\bullet}{\div} \mathscr{M}(F) \qquad ''$$

$$\overrightarrow{q' \Rightarrow u'} \subseteq \overrightarrow{q \Rightarrow u} \qquad ''$$

$$zip(\overrightarrow{a'})(\mathscr{M}'(F)) \subseteq zip(\overrightarrow{a})(\mathscr{M}(F)) \qquad ''$$

$$\Xi, \overrightarrow{a \stackrel{\mathsf{d}\bullet}{\div} \tau, a\,\mathrm{Id}} \vdash \wr \overrightarrow{q \Rightarrow u}; \hat{I}; \mathscr{M}(F) \int \mathrel{\hat{=}} \Xi_1, \overrightarrow{\psi_1}; R_1 \qquad \text{Subderivation}$$

$$\Xi, \overrightarrow{a \stackrel{\mathsf{d}\bullet}{\div} \tau, a\,\mathrm{Id}} \vdash \wr \overrightarrow{q' \Rightarrow u'}; \hat{I}; \mathscr{M}'(F) \int \mathrel{\hat{=}} {}^{\mathsf{d}}\Theta_1'; R_1' \qquad \text{By i.h.}$$

$${}^{\mathsf{d}}\Theta_1' \subseteq \Xi_1, \overrightarrow{\psi_1} \qquad ''$$

$$R_1' \subseteq R_1 \qquad ''$$

$$\text{Let } \overrightarrow{a' \stackrel{\mathsf{d}\bullet}{\div} \tau'} = \overrightarrow{a'} \stackrel{\mathsf{d}\bullet}{\div} \mathscr{M}'(F).$$

$$\Xi, \overrightarrow{a' \stackrel{\mathsf{d}\bullet}{\div} \tau', a'\,\mathrm{Id}} \vdash \wr \overrightarrow{q' \Rightarrow u'}; \hat{I}; \mathscr{M}'(F) \int \mathrel{\hat{=}} {}^{\mathsf{d}}\Theta_1'; R_1' \qquad \text{By strengthening}$$

$${}^{\mathsf{d}}\Theta_1' = \Xi_1', \overrightarrow{\psi_1'} \qquad \text{By inversion}$$

$$\Xi_1' \subseteq \Xi_1 \qquad ''$$

$$\overrightarrow{\psi_1'} \subseteq \overrightarrow{\psi_1} \qquad ''$$

$$\Xi, \overrightarrow{a' \stackrel{\mathsf{d}\bullet}{\div} \tau', a'\,\mathrm{Id}} \vdash \wr \overrightarrow{q' \Rightarrow u'}; \hat{I}; \mathscr{M}'(F) \int \mathrel{\hat{=}} \Xi_1', \overrightarrow{\psi_1'}; R_1' \qquad \text{By equality}$$

$$\Xi;\Xi_1;zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash \overrightarrow{\psi_1} \rightsquigarrow \breve{\Xi}_1;\mathscr{M}_1(F);\overrightarrow{\psi_0} \qquad \text{Premise}$$

$$\Xi;\Xi_1;zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash R_1 \rightsquigarrow \breve{\Xi}_2;\mathscr{M}_2(F);R_0 \qquad \text{Premise}$$

$$\Xi;\Xi_1;zip(\overrightarrow{a'})(\mathscr{M}'(F)) \vdash \overrightarrow{\psi'_1} \rightsquigarrow \breve{\Xi}'_1;\mathscr{M}'_1(F);\overrightarrow{\psi'_0} \qquad \text{Straightforward}$$

$$\breve{\Xi}'_1 \subseteq \breve{\Xi}_1 \qquad ''$$

$$\mathscr{M}'_1(F) \subseteq \mathscr{M}_1(F) \qquad ''$$

$$\overrightarrow{\psi'_0} \subseteq \overrightarrow{\psi_0} \qquad ''$$

$$\Xi;\Xi_1;zip(\overrightarrow{a'})(\mathscr{M}'(F)) \vdash R'_1 \rightsquigarrow \breve{\Xi}'_2;\mathscr{M}'_2(F);R'_0 \qquad \text{Straightforward}$$

$$\breve{\Xi}'_2 \subseteq \breve{\Xi}_2 \qquad ''$$

$$\mathscr{M}'_2(F) \subseteq \mathscr{M}_2(F) \qquad ''$$

$$R'_0 \subseteq R_0 \qquad ''$$

$$\text{Let } \breve{\Xi}' = \breve{\Xi}'_1 \cup \breve{\Xi}'_2.$$

$$\text{Let } \mathscr{M}''(F) = \mathscr{M}'_1(F) \cup \mathscr{M}'_2(F).$$

$$\text{Let } \rho' = \rho\restriction_{\breve{\Xi}'}.$$

$$\text{Let } \Xi'' = \Xi'\restriction_{cod(\rho')}.$$

☞ $\Xi'', \Xi'_1, [\rho']\overrightarrow{\psi'_0} \subseteq \Xi', \Xi_1, [\rho]\overrightarrow{\psi_0}$ Follows from above

☞ $\{\nu : \mu F \mid [\rho']\mathscr{M}''(F)\} \times [\rho']R'_0 \subseteq \{\nu : \mu F \mid [\rho]\mathscr{M}'(F)\} \times [\rho]R_0$ Follows from above

☞ $\Xi \vdash \wr\overrightarrow{(a,q) \Rightarrow u};\mathrm{Id} \otimes \hat{I};\mathscr{M}'(F)\wr \mathrel{\hat{=}} \Xi'', \Xi'_1, [\rho']\overrightarrow{\psi'_0};\{\nu : \mu F \mid [\rho']\mathscr{M}''(F)\} \times [\rho']R'_0 \quad \wr\mathrm{Id}\wr$

It is straightforward to prove by induction, and by using Lemma C.62 (Subtyping Reflexive), Lemma C.34 (Ix. Equiv. Reflexive), and Lemma C.27 (Prop. Truth Equiv.

Relation), that if ${}^{\mathsf{d}}\Theta' \subseteq {}^{\mathsf{d}}\Theta$ and $R' \subseteq R$ (as defined in the statement) and these are all well-formed under Ξ then $\Xi \vdash \exists^{\mathsf{d}}\Theta.\, R \wedge {}^{\mathsf{d}}\Theta \leq^{+} \exists^{\mathsf{d}}\Theta'.\, R' \wedge {}^{\mathsf{d}}\Theta'$.

- **Case**

$$\frac{\overrightarrow{t'} \mathbin{@} \mathscr{M}(F) \mathrel{\dot{=}} \overrightarrow{\varphi}}{\Xi \vdash \wr \overrightarrow{() \Rightarrow t'}; I; \mathscr{M}(F) \wr \mathrel{\dot{=}} \underbrace{\overrightarrow{\varphi}}_{{}^{\mathsf{d}}\Theta}; \underbrace{1}_{R}} \wr I \wr$$

$\overrightarrow{t'} \mathbin{@} \mathscr{M}(F) \mathrel{\dot{=}} \overrightarrow{\varphi}$ — Premise

$zip(\overrightarrow{\beta'})(\mathscr{M}'(F)) \subseteq zip(\overrightarrow{() \Rightarrow t'})(\mathscr{M}(F))$ — Given

$zip(\overrightarrow{\beta'})(\mathscr{M}'(F)) = zip(\overrightarrow{() \Rightarrow u})(\mathscr{M}'(F))$ — By inversion

$\overrightarrow{u} \mathbin{@} \mathscr{M}'(F) \mathrel{\dot{=}} \overrightarrow{\varphi'}$ — Straightforward

☞ $\overrightarrow{\varphi'} \subseteq \overrightarrow{\varphi}$ — ″

$\Xi \vdash \wr \overrightarrow{() \Rightarrow u}; I; \mathscr{M}(F) \wr \mathrel{\dot{=}} \overrightarrow{\varphi'}; 1$ — By $\wr I \wr$

☞ $\Xi \vdash \wr \overrightarrow{\beta'}; I; \mathscr{M}(F) \wr \mathrel{\dot{=}} \overrightarrow{\varphi'}; 1$ — By equality

$\Xi, \overrightarrow{\varphi} \vdash 1 \leq^{+} 1$ — By $\leq^{+}1$

$\Xi, \overrightarrow{\varphi} \vdash \overrightarrow{\varphi'}\,\mathsf{true}$ — By Lemma C.29 (Consequence) (repeated)

$\Xi, \overrightarrow{\varphi} \vdash 1 \leq^{+} 1 \wedge \overrightarrow{\varphi'}$ — By $\leq^{+}\wedge$R

$\Xi \vdash 1 \wedge \overrightarrow{\varphi} \leq^{+} 1 \wedge \overrightarrow{\varphi'}$ — By $\leq^{+}\wedge$L (and permute)

Let ${}^{\mathsf{d}}\Theta' = \overrightarrow{\varphi'}$.

Let $R' = 1$.

☞ $\Xi \vdash \exists^{\mathsf{d}}\Theta.\, R \wedge {}^{\mathsf{d}}\Theta \leq^{+} \exists^{\mathsf{d}}\Theta'.\, R' \wedge {}^{\mathsf{d}}\Theta'$ — By equalities, defs.

□

Lemma C.79 (Unroll Superlist)**.**
If $\Xi \vdash \wr \overrightarrow{\beta'}; G; \mathscr{M}'(F) \wr \triangleq {}^{\mathsf{d}}\Theta'; R'$ *and* $zip(\overrightarrow{\beta'})(\mathscr{M}'(F)) \subseteq zip(\overrightarrow{\beta})(\mathscr{M}(F))$
then $\Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \wr \triangleq {}^{\mathsf{d}}\Theta; R$ *and* $\Xi \vdash \exists^{\mathsf{d}}\Theta.\, R \wedge {}^{\mathsf{d}}\Theta \leq^{+} \exists^{\mathsf{d}}\Theta'.\, R' \wedge {}^{\mathsf{d}}\Theta'$
and if $G = \hat{I}$ *then* ${}^{\mathsf{d}}\Theta' \subseteq {}^{\mathsf{d}}\Theta$ *and* $R' \subseteq R$ *where the latter* ($R' \subseteq R$) *is defined by*

- $1 \subseteq 1$
- *if* $R' \subseteq R$ *and* $\mathscr{M}_1'(F) \subseteq \mathscr{M}_1(F)$
  then $\{\nu : \mu F \mid \mathscr{M}_1'(F)\} \times R' \subseteq \{\nu : \mu F \mid \mathscr{M}_1(F)\} \times R$

Proof. By structural induction on $\Xi \vdash \wr \overrightarrow{\beta'}; G; \mathscr{M}'(F) \wr \triangleq {}^{\mathsf{d}}\Theta'; R'$. Similar to Lemma C.78 (Unroll Sublist). □

Lemma C.80 (Subsumption Admissibility)**.** *Assume* $\Theta \vdash \Gamma' \leq^{+} \Gamma$. *Then:*

(1) If $\Theta; \Gamma \vdash h \Rightarrow P$
then there exists P' *such that* $\Theta \vdash P' \leq^{+} P$ *and* $\Theta; \Gamma' \vdash h \Rightarrow P'$.
Moreover, either (a) $P' = P$ *or (b)* $P' = R$ *for some* R.

(2) If $\Theta; \Gamma \vdash g \Rightarrow \uparrow P$
then there exists P' *such that* $\Theta \vdash \uparrow P' \leq^{-} \uparrow P$ *and* $\Theta; \Gamma' \vdash g \Rightarrow \uparrow P'$.

(3) If $\Theta; \Gamma \vdash v \Leftarrow P$ *and* $\Theta \vdash P \leq^{+} P'$ *then* $\Theta; \Gamma' \vdash v \Leftarrow P'$.

(4) If $\Theta; \Gamma \vdash e \Leftarrow N$ *and* $\Theta \vdash N \leq^{-} N'$ *then* $\Theta; \Gamma' \vdash e \Leftarrow N'$.

(5) If $\Theta; \Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$ *and* $\Theta \vdash N \leq^{-} N'$ *and* $\Theta \vdash P' \leq^{+} P$
then $\Theta; \Gamma'; [P'] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N'$.

(6) If $\Theta; \Gamma; [N] \vdash s \Rightarrow \uparrow P$ *and* $\Theta \vdash N' \leq^{-} N$
then there exists P' *such that* $\Theta \vdash \uparrow P' \leq^{-} \uparrow P$ *and* $\Theta; \Gamma'; [N'] \vdash s \Rightarrow \uparrow P'$.

Proof. By lexicographic induction, first, on the height of the given typing derivation; second, on the structure of the given principal subtyping judgment, that is, $\Theta \vdash P \leq^+ P'$ in part (3), $\Theta \vdash N \leq^- N'$ in part (4), $\Theta \vdash P' \leq^+ P$ in part (5), and $\Theta \vdash N' \leq^- N$ in part (6).

(1) • **Case**

$$\dfrac{(x : R) \in \Gamma}{\Theta; \Gamma \vdash x \Rightarrow R}\ \text{Decl}{\Rightarrow}\text{Var}$$

	$(x : R) \in \Gamma$	Premise
	$\Theta \vdash \Gamma' \leq^+ \Gamma$	Given
	$(x : R') \in \Gamma'$	By inversion
☞	$\Theta \vdash R' \leq^+ R$	$''$
☞	$\Theta; \Gamma' \vdash x \Rightarrow R'$	By Decl⇒Var

• **Case**

$$\dfrac{\overline{\Theta} \vdash P\ \mathsf{type}[\xi] \qquad \Theta; \Gamma \vdash v \Leftarrow P}{\Theta; \Gamma \vdash (v : P) \Rightarrow P}\ \text{Decl}{\Rightarrow}\text{ValAnnot}$$

	$\Theta \vdash \Gamma' \leq^+ \Gamma$	Given
☞	$\Theta \vdash P \leq^+ P$	By Lemma C.62 (Subtyping Reflexive)
	$\Theta; \Gamma \vdash v \Leftarrow P$	Subderivation
	$\Theta; \Gamma' \vdash v \Leftarrow P$	By i.h. (smaller typing height)
☞	$\Theta; \Gamma' \vdash (v : P) \Rightarrow P$	By Decl⇒ValAnnot

(2) • **Case**

$$\dfrac{\Theta; \Gamma \vdash h \Rightarrow {\downarrow}N \qquad \Theta; \Gamma; [N] \vdash s \Rightarrow {\uparrow}P}{\Theta; \Gamma \vdash h(s) \Rightarrow {\uparrow}P}\ \text{Decl}{\Rightarrow}\text{App}$$

$\Theta \vdash \Gamma' \leq^+ \Gamma$ Given

$\Theta;\Gamma \vdash h \Rightarrow \downarrow N$ Subderivation

By i.h., either (a) $\Theta;\Gamma' \vdash h \Rightarrow \downarrow N$; or

(b) there exists R such that $\Theta \vdash R \leq^+ \downarrow N$ and $\Theta;\Gamma' \vdash h \Rightarrow R$.

Consider subcases (a) and (b):

– **Case** (a):

$\Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P$ Subderivation

$\Theta \vdash N \leq^- N$ By Lemma C.62 (Subtyping Reflexive)

$\Theta;\Gamma';[N] \vdash s \Rightarrow \uparrow P'$ By i.h. (smaller typing height)

☞ $\Theta \vdash \uparrow P' \leq^- \uparrow P$ ″

☞ $\Theta;\Gamma' \vdash h(s) \Rightarrow \uparrow P'$ By Decl⇒App

– **Case** (b):

$R = \downarrow N'$ By inversion on $\leq^+\downarrow$

$\Theta \vdash \downarrow N' \leq^+ \downarrow N$ Rewrite above

$\Theta \vdash N' \leq^- N$ By inversion on $\leq^+\downarrow$

$\Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P$ Subderivation

$\Theta;\Gamma';[N'] \vdash s \Rightarrow \uparrow P'$ By i.h. (smaller typing height)

☞ $\Theta \vdash \uparrow P' \leq^- \uparrow P$ ″

☞ $\Theta;\Gamma' \vdash h(s) \Rightarrow \uparrow P'$ By Decl⇒App

- **Case**

$$\dfrac{\overline{\Theta} \vdash P\ \mathsf{type}[\xi] \qquad \Theta;\Gamma \vdash e \Leftarrow \uparrow P}{\Theta;\Gamma \vdash (e : \uparrow P) \Rightarrow \uparrow P}\ \text{Decl}{\Rightarrow}\text{ExpAnnot}$$

	$\Theta \vdash \Gamma' \leq^{+} \Gamma$	Given
☞	$\Theta \vdash \uparrow P \leq^{-} \uparrow P$	By Lemma C.62 (Subtyping Reflexive)
	$\Theta;\Gamma \vdash e \Leftarrow \uparrow P$	Subderivation
	$\Theta;\Gamma' \vdash e \Leftarrow \uparrow P$	By i.h. (smaller typing height)
☞	$\Theta;\Gamma' \vdash (e : \uparrow P) \Rightarrow \uparrow P$	By Decl⇒ExpAnnot

(3) We first consider the $\leq^{+}\wedge$R and $\leq^{+}\exists$R cases of the given subtyping, which are independent of typing derivation structure. Then we case analyze the typing derivation.

- **Case**

$$\dfrac{\Theta \vdash R \leq^{+} R' \qquad \Theta \vdash \overrightarrow{\varphi}\ \mathsf{true}}{\Theta \vdash \underbrace{R}_{P} \leq^{+} \underbrace{R' \wedge \overrightarrow{\varphi}}_{P'}}\ \leq^{+}\wedge\text{R}$$

$\Theta \vdash \Gamma' \leq^{+} \Gamma$	Given
$\Theta;\Gamma \vdash v \Leftarrow R$	Given
$\Theta \vdash P \leq^{+} R'$	Subderivation
$\Theta;\Gamma' \vdash v \Leftarrow R'$	By i.h. (same typing height; smaller subtyping)
$\Theta \vdash \overrightarrow{\varphi}\ \mathsf{true}$	Premise
$\Theta;\Gamma' \vdash v \Leftarrow R' \wedge \overrightarrow{\varphi}$	By Decl⇐∧

- **Case**

$$\dfrac{\overset{\mathsf{d}}{\cdot}\Theta \vdash \sigma : {}^{\mathsf{d}}\Xi \qquad \Theta \vdash R \leq^{+} [\sigma]Q'}{\Theta \vdash R \leq^{+} \exists^{\mathsf{d}}\Xi.\, Q'}\ \leq^{+}\exists\text{R}$$

$\Theta \vdash \Gamma' \leq^+ \Gamma$ Given

$\Theta;\Gamma \vdash v \Leftarrow R$ Given

$\Theta \vdash R \leq^+ [\sigma]Q'$ Premise

$\Theta;\Gamma' \vdash v \Leftarrow [\sigma]Q'$ By i.h. (same typing height; smaller subtyping)

$\dot{\div}\Theta \vdash \sigma : {}^{\mathsf{d}}\Xi$ Premise

$\Theta;\Gamma' \vdash v \Leftarrow \exists^{\mathsf{d}}\Xi.\, Q'$ By Decl$\Leftarrow\exists$

- **Case**

$$\frac{(x : R') \in \Gamma \qquad \Theta \vdash R' \leq^+ R}{\Theta;\Gamma \vdash x \Leftarrow R}\ \text{Decl}\Leftarrow\text{Var}$$

$\Theta \vdash R' \leq^+ R$ Premise

$\Theta \vdash R \leq^+ P'$ Given

$\Theta \vdash R' \leq^+ P'$ By Lemma C.63 (Subtyping Transitive)

$(x : R') \in \Gamma$ Premise

$\Theta \vdash \Gamma' \leq^+ \Gamma$ Given

$(x : R'') \in \Gamma'$ By inversion

$\Theta \vdash R'' \leq^+ R'$ ″

$\Theta \vdash R'' \leq^+ P'$ By Lemma C.63 (Subtyping Transitive)

☞ $\Theta;\Gamma' \vdash x \Leftarrow P'$ By Decl$\Leftarrow$Var

- **Case**

$$\frac{\dot{\div}\Theta \vdash \sigma : {}^{\mathsf{d}}\Xi \qquad \Theta;\Gamma \vdash v \Leftarrow [\sigma]Q}{\Theta;\Gamma \vdash v \Leftarrow (\exists^{\mathsf{d}}\Xi.\, Q)}\ \text{Decl}\Leftarrow\exists$$

– **Case**

$$\frac{\Theta, {}^{\mathsf{d}}\Xi \vdash Q \leq^{+} P'}{\Theta \vdash \exists^{\mathsf{d}}\Xi.\, Q \leq^{+} P'} \leq^{+}\exists\mathrm{L}$$

${}^{\mathsf{d}}\!\div\Theta \vdash \sigma : {}^{\mathsf{d}}\Xi$	Premise
$\Theta, {}^{\mathsf{d}}\Xi \vdash Q \leq^{+} P'$	Subderivation
$\Theta \vdash [\sigma]Q \leq^{+} [\sigma]P'$	By Lemma C.59 (Sub. Syn. Subs)
$\Theta \vdash [\sigma]Q \leq^{+} P'$	$FV(P') \cap dom({}^{\mathsf{d}}\Xi) = \emptyset$
$\Theta; \Gamma \vdash v \Leftarrow [\sigma]Q$	Subderivation
$\Theta; \Gamma' \vdash v \Leftarrow P'$	By i.h. (smaller typing height)

- **Case**

$$\frac{\Theta \vdash \overrightarrow{\varphi}\,\mathsf{true} \qquad \Theta; \Gamma \vdash v \Leftarrow R}{\Theta; \Gamma \vdash v \Leftarrow R \wedge \overrightarrow{\varphi}}\ \mathrm{Decl}{\Leftarrow}\wedge$$

– **Case**

$$\frac{\Theta, \overrightarrow{\varphi} \vdash R \leq^{+} P'}{\Theta \vdash R \wedge \overrightarrow{\varphi} \leq^{+} P'} \leq^{+}\wedge\mathrm{L}$$

$\Theta; \Gamma \vdash v \Leftarrow R$	Subderivation
$\Theta, \overrightarrow{\varphi}; \Gamma \vdash v \Leftarrow R$	By Lemma C.41 (Ix.-Level Weakening)
(same height)	$''$
$\Theta, \overrightarrow{\varphi} \vdash R \leq^{+} P'$	Subderivation
$\Theta, \overrightarrow{\varphi}; \Gamma' \vdash v \Leftarrow P'$	By i.h. (smaller typing height)
$\Theta \vdash \overrightarrow{\varphi}\,\mathsf{true}$	Premise
$\Theta; \Gamma' \vdash v \Leftarrow P'$	By repeated Lemma C.67 (Typing Consequence)

- **Case**

$$\frac{\nexists x.\, v' = \overrightarrow{\mathrm{inj}_{k_i}}^{i}\left(\overrightarrow{\langle __j, - \rangle}^{j} x\right) \qquad \mathscr{M}(F) \rightsquigarrow \overrightarrow{\alpha}; \overrightarrow{\tau} \qquad \overset{\mathsf{d}}{\div}\Theta \vdash \wr\, \overrightarrow{\alpha}; F; \mathscr{M}(F) \,\wr \doteq {}^{\mathsf{d}}\Theta; R \qquad \Theta; \Gamma \vdash v' \Leftarrow \exists^{\mathsf{d}}\Theta.\,(R \wedge {}^{\mathsf{d}}\Theta)}{\Theta; \Gamma \vdash \mathtt{into}(v') \Leftarrow \{\nu : \mu F \mid \mathscr{M}(F)\}}\ \mathrm{Decl}{\Leftarrow}\mu$$

  - **Case**

$$\frac{\Theta \vdash \mathscr{M}(F) \geq \mathscr{M}'(F')}{\Theta \vdash \{\nu : \mu F \mid \mathscr{M}(F)\} \leq^{+} \left\{\nu : \mu F' \;\middle|\; \mathscr{M}'(F')\right\}}\ {\leq^{+}}\mu$$

    From $\Theta \vdash \mathscr{M}(F) \geq \mathscr{M}'(F')$

    we extract the $\mathscr{M}_0(F) \subseteq \mathscr{M}(F)$ such that $\Theta \vdash \mathscr{M}_0(F) \geq_{\equiv} \mathscr{M}'(F')$.

    We extract $\mathscr{M}_0(F) \rightsquigarrow \overrightarrow{\alpha_0}; \overrightarrow{\tau_0}$ and $\mathscr{M}'(F') \rightsquigarrow \overrightarrow{\alpha'}; \overrightarrow{\tau_0}$

    using the side judgment for Decl$\Leftarrow\mu$ (defined where the rule is).

$\overset{\mathsf{d}}{\div}\Theta \vdash \wr\, \overrightarrow{\alpha}; F; \mathscr{M}(F) \,\wr \doteq {}^{\mathsf{d}}\Theta; R$	Premise
$\overset{\mathsf{d}}{\div}\Theta \vdash \wr\, \overrightarrow{\alpha_0}; F; \mathscr{M}_0(F) \,\wr \doteq {}^{\mathsf{d}}\Theta_0; R_0$	By Lemma C.78 (Unroll Sublist)
$\Theta \vdash \exists^{\mathsf{d}}\Theta.\, R \wedge {}^{\mathsf{d}}\Theta \leq^{+} \exists^{\mathsf{d}}\Theta_0.\, R_0 \wedge {}^{\mathsf{d}}\Theta_0$	$''$
$\cdot \vdash \overrightarrow{\alpha_0}; F \leq_{\overrightarrow{\tau_0}} \overrightarrow{\alpha'}; F'$	By inversion
$\overset{\mathsf{d}}{\div}\Theta \vdash \wr\, \overrightarrow{\alpha'}; F'; \mathscr{M}'(F') \,\wr \doteq {}^{\mathsf{d}}\Theta'; R'$	By Lemma C.75 (Unroll to Supertype)
$\Theta \vdash \exists^{\mathsf{d}}\Theta_0.\, R_0 \wedge {}^{\mathsf{d}}\Theta_0 \leq^{+} \exists^{\mathsf{d}}\Theta'.\, R' \wedge {}^{\mathsf{d}}\Theta'$	$''$
$\Theta \vdash \exists^{\mathsf{d}}\Theta.\, R \wedge {}^{\mathsf{d}}\Theta \leq^{+} \exists^{\mathsf{d}}\Theta'.\, R' \wedge {}^{\mathsf{d}}\Theta'$	By Lemma C.63 (Subtyping Transitive)
$\Theta; \Gamma \vdash v' \Leftarrow \exists^{\mathsf{d}}\Theta.\, R \wedge {}^{\mathsf{d}}\Theta$	Subderivation
$\Theta; \Gamma' \vdash v' \Leftarrow \exists^{\mathsf{d}}\Theta'.\, R' \wedge {}^{\mathsf{d}}\Theta'$	By i.h. (smaller typing hgt.)
$\Theta; \Gamma' \vdash \mathtt{into}(v') \Leftarrow \{\nu : \mu F' \mid \mathscr{M}'(F')\}$	By Decl$\Leftarrow\mu$

- **Case**

$$\frac{\Theta;\Gamma \vdash e \Leftarrow N}{\Theta;\Gamma \vdash \{e\} \Leftarrow \downarrow N}\ \mathrm{Decl}{\Leftarrow}{\downarrow}$$

  - **Case**

$$\frac{\Theta \vdash N \leq^{-} N'}{\Theta \vdash \downarrow N \leq^{+} \downarrow N'}\ {\leq^{+}}{\downarrow}$$

$\Theta \vdash \Gamma' \leq^{+} \Gamma$ Given

$\Theta \vdash N \leq^{-} N'$ Premise

$\Theta;\Gamma \vdash e \Leftarrow N$ Subderivation

$\Theta;\Gamma' \vdash e \Leftarrow N'$ By i.h. (smaller typing hgt.)

$\Theta;\Gamma' \vdash \{e\} \Leftarrow \downarrow N'$ By Decl$\Leftarrow\downarrow$

- **Case**

$$\frac{\Theta;\Gamma \vdash v' \Leftarrow P_k}{\Theta;\Gamma \vdash \mathrm{inj}_k\, v' \Leftarrow P_1 + P_2}\ \mathrm{Decl}{\Leftarrow}{+_k}$$

  - **Case**

$$\frac{\Theta \vdash P_1 \leq^{+} P_1' \qquad \Theta \vdash P_2 \leq^{+} P_2'}{\Theta \vdash P_1 + P_2 \leq^{+} P_1' + P_2'}\ {\leq^{+}}{+}$$

$\Theta \vdash \Gamma' \leq^{+} \Gamma$ Given

$\Theta \vdash P_k \leq^{+} P_k'$ Premise

$\Theta;\Gamma \vdash v' \Leftarrow P_k$ Subderivation

$\Theta;\Gamma' \vdash v' \Leftarrow P_k'$ By i.h. (smaller typing hgt.)

$\Theta;\Gamma' \vdash \mathrm{inj}_k\, v' \Leftarrow P_1' + P_2'$ By Decl$\Leftarrow+_k$

- **Case** Decl$\Leftarrow\times$: Straightforward. Use i.h. (smaller typing height) for each corresponding value typing and subtyping subderivation/premise, then reapply Decl$\Leftarrow\times$.
- **Case** Decl$\Leftarrow$1:

$P' = 1$ By inversion on $\leq^{+}1$

$\Theta;\Gamma' \vdash \langle\rangle \Leftarrow 1$ By Decl$\Leftarrow$1

(4) We first consider the cases of rule concluding the subtyping derivation that is independent of the program typing derivation, namely, $\leq^{-}\forall$R and $\leq^{-}\supset$R. We then analyze the expression typing derivation; each such case has exactly one corresponding subcase for the concluding rule of the subtyping derivation, because the $\leq^{-}\forall$R and $\leq^{-}\supset$R subcases are already independently covered.

- **Case**

$$\frac{\Theta, \overrightarrow{\varphi'} \vdash N \leq^{-} L'}{\Theta \vdash N \leq^{-} \overrightarrow{\varphi'} \supset L'} \leq^{-}\supset\text{R}$$

$\Theta;\Gamma \vdash e \Leftarrow N$ Given

$\Theta, \overrightarrow{\varphi'};\Gamma \vdash e \Leftarrow N$ By Lemma C.41 (Ix.-Level Weakening)

(same height) $''$

$\Theta, \overrightarrow{\varphi'} \vdash N \leq^{-} L'$ Subderivation

$\Theta, \overrightarrow{\varphi'};\Gamma' \vdash e \Leftarrow L'$ By i.h. (same typing height; smaller subtyping)

$\Theta;\Gamma' \vdash e \Leftarrow \overrightarrow{\varphi'} \supset L'$ By Decl$\Leftarrow\supset$

- **Case** $\leq^{-}\forall$R: Similar to $\leq^{-}\supset$R case.
- **Case** Decl$\Leftarrow$Unreachable: By Decl$\Leftarrow$Unreachable.

We have now covered all the cases where $N' \neq L'$ (the two right rules for the given subtyping). For the remaining cases we may assume $N' = L'$.

- **Case**

$$\dfrac{\Theta;\Gamma \vdash g \Rightarrow \uparrow(\exists^{\mathsf{d}}\Xi.R \wedge \overrightarrow{\psi}) \qquad \Theta,{}^{\mathsf{d}}\Xi,\overrightarrow{\psi};\Gamma,x:R \vdash e' \Leftarrow L}{\mathscr{D} :: \Theta;\Gamma \vdash \mathsf{let}\ x = g;\ e' \Leftarrow L}\ \mathrm{Decl}{\Leftarrow}\mathsf{let}$$

$\Theta \vdash \Gamma' \leq^{+} \Gamma$	Given
$\Theta;\Gamma \vdash g \Rightarrow \uparrow(\exists^{\mathsf{d}}\Xi.R \wedge \overrightarrow{\psi})$	Subderivation
$\Theta;\Gamma' \vdash g \Rightarrow \uparrow(\exists^{\mathsf{d}}\Xi'.R' \wedge \overrightarrow{\psi'})$	By i.h. (smaller typing height)
$\Theta \vdash \uparrow(\exists^{\mathsf{d}}\Xi'.R' \wedge \overrightarrow{\psi'}) \leq^{-} \uparrow(\exists^{\mathsf{d}}\Xi.R \wedge \overrightarrow{\psi})$	$''$
$\Theta \vdash \exists^{\mathsf{d}}\Xi'.R' \wedge \overrightarrow{\psi'} \leq^{+} \exists^{\mathsf{d}}\Xi.R \wedge \overrightarrow{\psi}$	By inversion
$\Theta,{}^{\mathsf{d}}\Xi',\overrightarrow{\psi'} \vdash R' \leq^{+} \exists^{\mathsf{d}}\Xi.R \wedge \overrightarrow{\psi}$	By inversion
$\Theta,{}^{\mathsf{d}}\Xi',\overrightarrow{\psi'} \vdash R' \leq^{+} [\overrightarrow{t}/{}^{\mathsf{d}}\Xi]R$	By inversion
${}^{\mathsf{d}}\div\Theta,{}^{\mathsf{d}}\Xi' \vdash \overrightarrow{t}/{}^{\mathsf{d}}\Xi : {}^{\mathsf{d}}\Xi$	$''$
$\Theta,{}^{\mathsf{d}}\Xi',\overrightarrow{\psi'} \vdash [\overrightarrow{t}/{}^{\mathsf{d}}\Xi]\overrightarrow{\psi}\,\mathsf{true}$	$''$
Let $\sigma = \overrightarrow{t}/{}^{\mathsf{d}}\Xi$.	
$\Theta,{}^{\mathsf{d}}\Xi',\overrightarrow{\psi'} \vdash \sigma : {}^{\mathsf{d}}\Xi$	By weakening
$\Theta,{}^{\mathsf{d}}\Xi',\overrightarrow{\psi'} \vdash \Gamma' \leq^{+} \Gamma$	By weakening
$\Theta,{}^{\mathsf{d}}\Xi',\overrightarrow{\psi'} \vdash \Gamma',x:R' \leq^{+} \Gamma,x:[\sigma]R$	Add entry

$$\begin{array}{rll}
\mathscr{D}_2 :: \ \Theta, {}^{\mathsf{d}}\Xi, \overrightarrow{\psi}; \Gamma, x : R \vdash e' \Leftarrow L & & \text{Subderivation} \\
\mathscr{D}_2' :: \ \Theta, {}^{\mathsf{d}}\Xi', \overrightarrow{\psi'}; [\sigma](\Gamma, x : R') \vdash [\sigma]e' \Leftarrow [\sigma]L & & \text{By cor. of Lemma C.68} \\
hgt(\mathscr{D}_2') \leq hgt(\mathscr{D}_2) & & '' \\
< hgt(\mathscr{D}) & & \text{By def. of } hgt(-) \\
\mathscr{D}_2' :: \ \Theta, {}^{\mathsf{d}}\Xi', \overrightarrow{\psi'}; [\sigma]\Gamma, x : [\sigma]R' \vdash [\sigma]e' \Leftarrow [\sigma]L & & \text{By def. of } [-]- \\
\mathscr{D}_2' :: \ \Theta, {}^{\mathsf{d}}\Xi', \overrightarrow{\psi'}; \Gamma, x : [\sigma]R' \vdash e' \Leftarrow L & & dom(\Theta) \cap dom({}^{\mathsf{d}}\Xi) = \emptyset
\end{array}$$

$$\begin{array}{rll}
\Theta \vdash L \leq^{-} N' & & \text{Given} \\
\Theta \vdash L \leq^{-} L' & & \text{By equality} \\
\Theta, {}^{\mathsf{d}}\Xi', \overrightarrow{\psi'} \vdash L \leq^{-} L' & & \text{By weakening} \\
\Theta, {}^{\mathsf{d}}\Xi', \overrightarrow{\psi'}; \Gamma', x : R' \vdash e' \Leftarrow L' & & \text{By i.h. (smaller typing height)} \\
\Theta; \Gamma' \vdash \mathsf{let}\ x = g;\ e' \Leftarrow L' & & \text{By Decl}\Leftarrow\text{let}
\end{array}$$

- **Case** Decl⇐`match`: Straightforward (use i.h. at smaller typing height).
- **Case**

$$\frac{\Theta \vdash \forall a \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.M \leq^{-} L \qquad \Theta, a \div \mathbb{N}; \Gamma, x : \downarrow \forall a' \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.a' < a \supset [a'/a]M \vdash e_0 \Leftarrow \forall^{\mathsf{d}}\Xi.M}{\Theta; \Gamma \vdash \mathsf{rec}\, x : (\forall a \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.M).\ e_0 \Leftarrow L}\ \text{Decl}\Leftarrow\text{rec}$$

$\Theta \vdash \Gamma' \leq^{+} \Gamma$ Given

$\Theta, a \div \mathbb{N} \vdash \Gamma' \leq^{+} \Gamma$ By weakening

By Lemma C.62 (Subtyping Reflexive),

$$\Theta, a \div \mathbb{N} \vdash \downarrow \forall a' \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.a' < a \supset [a'/a]M \leq^{+} \downarrow \forall a' \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.a' < a \supset [a'/a]M$$

Add entry:

$\Theta, a \div \mathbb{N} \vdash \Gamma', x : \downarrow \forall a' \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, a' < a \supset [a'/a]M$

$\leq^{+} \Gamma, x : \downarrow \forall a' \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, a' < a \supset [a'/a]M$

$\Theta, a \div \mathbb{N} \vdash \forall^{\mathsf{d}}\Xi.\, M \leq^{-} \forall^{\mathsf{d}}\Xi.\, M$ By Lemma C.62 (Subtyping Reflexive)

$\Theta, a \div \mathbb{N}; \Gamma, x : \downarrow \forall a' \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, a' < a \supset [a'/a]M \vdash e_0 \Leftarrow \forall^{\mathsf{d}}\Xi.\, M$ Subderivation

$\Theta, a \div \mathbb{N}; \Gamma', x : \downarrow \forall a' \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, a' < a \supset [a'/a]M \vdash e_0 \Leftarrow \forall^{\mathsf{d}}\Xi.\, M$ By i.h.

(smaller typing hgt.)

$\Theta \vdash \forall a \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, M \leq^{-} L$ Premise

$\Theta \vdash L \leq^{-} N'$ Given

$\Theta \vdash \forall a \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, M \leq^{-} N'$ By Lemma C.63 (Subtyping Transitive)

$\Theta; \Gamma' \vdash \mathtt{rec}\, x : (\forall a \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, M).\, e_0 \Leftarrow L'$ By Decl⇐rec

- **Case**

$$\frac{\Theta; \Gamma, x : R \vdash e_0 \Leftarrow L}{\Theta; \Gamma \vdash \lambda x.\, e_0 \Leftarrow R \to L}\ \text{Decl}{\Leftarrow}\lambda$$

  - **Case**

$$\frac{\Theta \vdash R' \leq^{+} R \qquad \Theta \vdash L \leq^{-} L'}{\Theta \vdash R \to L \leq^{-} R' \to L'}\ \leq^{-}{\to}$$

$\Theta \vdash R' \leq^+ R$ Premise

$\Theta \vdash \Gamma' \leq^+ \Gamma$ Given

$\Theta \vdash \Gamma', x : R' \leq^+ \Gamma, x : R$ Add entry

$\Theta \vdash L \leq^- L'$ Premise

$\Theta; \Gamma, x : R \vdash e_0 \Leftarrow L$ Subderivation

$\Theta; \Gamma', x : R' \vdash e_0 \Leftarrow L'$ By i.h.

(smaller typing height)

$\Theta; \Gamma' \vdash \lambda x.\, e_0 \Leftarrow R' \to L'$ By Decl$\Leftarrow\lambda$

- **Case**

$$\frac{\Theta, {}^{\mathsf{d}}\Xi; \Gamma \vdash e \Leftarrow M}{\Theta; \Gamma \vdash e \Leftarrow \forall^{\mathsf{d}}\Xi.\, M}\ \text{Decl}\Leftarrow\forall$$

$\Theta \vdash \forall^{\mathsf{d}}\Xi.\, M \leq^- L'$ Given

$\Theta \vdash [\overrightarrow{t}/{}^{\mathsf{d}}\Xi]M \leq^- L'$ By inversion

$\overset{\mathsf{d}}{\div}\Theta \vdash \overrightarrow{t}/{}^{\mathsf{d}}\Xi : {}^{\mathsf{d}}\Xi$ ″

$\Theta \vdash \overrightarrow{t}/{}^{\mathsf{d}}\Xi : {}^{\mathsf{d}}\Xi$ By weakening

$\Theta, {}^{\mathsf{d}}\Xi; \Gamma \vdash e \Leftarrow M$ Subderivation

$\Theta; [\overrightarrow{t}/{}^{\mathsf{d}}\Xi]\Gamma \vdash [\overrightarrow{t}/{}^{\mathsf{d}}\Xi]e \Leftarrow [\overrightarrow{t}/{}^{\mathsf{d}}\Xi]M$ By cor. of Lemma C.68

hgt. does not increase ″

$\Theta; \Gamma \vdash e \Leftarrow [\overrightarrow{t}/{}^{\mathsf{d}}\Xi]M$ $dom(\Theta) \cap dom({}^{\mathsf{d}}\Xi) = \emptyset$

$\Theta \vdash \Gamma' \leq^+ \Gamma$ Given

$\Theta; \Gamma' \vdash e \Leftarrow L'$ By i.h. (smaller typing height)

- **Case**

$$\frac{\Theta, \overrightarrow{\varphi}; \Gamma \vdash e \Leftarrow L}{\Theta; \Gamma \vdash e \Leftarrow \overrightarrow{\varphi} \supset L}\ \mathrm{Decl}{\Leftarrow}{\supset}$$

$\Theta \vdash \overrightarrow{\varphi} \supset L \leq^{-} L'$	Given
$\Theta \vdash L \leq^{-} L'$	By inversion
$\Theta \vdash \overrightarrow{\varphi}\,\mathsf{true}$	$''$
$\Theta, \overrightarrow{\varphi}; \Gamma \vdash e \Leftarrow L$	Subderivation
$\Theta, \overrightarrow{\varphi} \vdash L \leq^{-} L'$	By weakening
$\Theta, \overrightarrow{\varphi}; \Gamma \vdash e \Leftarrow L'$	By i.h. (smaller typing height)
$\Theta; \Gamma \vdash e \Leftarrow L'$	By Lemma C.67 (Typing Consequence)

- **Case**

$$\frac{\Theta; \Gamma \vdash v \Leftarrow P}{\Theta; \Gamma \vdash \mathtt{return}\, v \Leftarrow \uparrow P}\ \mathrm{Decl}{\Leftarrow}{\uparrow}$$

  - **Case**

$$\frac{\Theta \vdash P \leq^{+} P'}{\Theta \vdash \uparrow P \leq^{-} \uparrow P'}\ \leq^{-}\uparrow$$

$\Theta \vdash \Gamma' \leq^{+} \Gamma$	Given
$\Theta \vdash P \leq^{+} P'$	Premise
$\Theta; \Gamma \vdash v \Leftarrow P$	Subderivation
$\Theta; \Gamma' \vdash v \Leftarrow P'$	By i.h. (smaller typing height)
$\Theta; \Gamma' \vdash \mathtt{return}\, v \Leftarrow \uparrow P'$	By Decl$\Leftarrow\uparrow$

(5) We first consider cases for the concluding rule of the positive subtyping derivation that is independent of the structure of P, i.e. $\leq^{+}\wedge\mathrm{L}$ and $\leq^{+}\exists\mathrm{L}$. Then we consider

cases for the final rule of the typing derivation; each such case has exactly one corresponding subcase for the final rule of the subtyping derivation that's determined by the structure of P for the given case (the $\leq^+\wedge$L and $\leq^+\exists$L cases already being covered).

- **Case**
$$\frac{\Theta, \overrightarrow{\varphi} \vdash R \leq^+ P}{\Theta \vdash R \wedge \overrightarrow{\varphi} \leq^+ P} \leq^+\wedge\text{L}$$

$\Theta, \overrightarrow{\varphi} \vdash R \leq^+ P$	Subderivation
$\Theta;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$	Given
$\Theta, \overrightarrow{\varphi};\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$	By Lemma C.41 (Ix.-Level Weakening)
(same height)	$''$
$\Theta, \overrightarrow{\varphi};\Gamma';[R] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N'$	By i.h. (same typing height; smaller subtyping)
$\Theta;\Gamma';[R \wedge \overrightarrow{\varphi}] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N'$	By DeclMatch$\wedge$

- **Case** $\leq^+\exists$L: Similar to $\leq^+\wedge$L case.

- **Case**
$$\frac{\Theta, {}^{\mathsf{d}}\Xi;\Gamma;[Q] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}{\Theta;\Gamma;[\exists^{\mathsf{d}}\Xi.\, Q] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N}\ \text{DeclMatch}\exists$$

  - **Case**
$$\frac{\Theta \vdash R \leq^+ [\overrightarrow{t}/^{\mathsf{d}}\Xi]Q \qquad {}^{\mathsf{d}}\div\Theta \vdash \overrightarrow{t}/^{\mathsf{d}}\Xi : {}^{\mathsf{d}}\Xi}{\Theta \vdash R \leq^+ \exists^{\mathsf{d}}\Xi.\, Q} \leq^+\exists\text{R}$$

$$\Theta, {}^{\mathsf{d}}\Xi; \Gamma; [Q] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N \quad \text{Subderivation}$$
$$\overset{\mathsf{d}\cdot}{\cdot}\Theta \vdash \overrightarrow{t}/{}^{\mathsf{d}}\Xi : {}^{\mathsf{d}}\Xi \quad \text{Premise}$$
$$\Theta \vdash \overrightarrow{t}/{}^{\mathsf{d}}\Xi : {}^{\mathsf{d}}\Xi \quad \text{By weakening}$$

$$\Theta; [\overrightarrow{t}/{}^{\mathsf{d}}\Xi]\Gamma; [[\overrightarrow{t}/{}^{\mathsf{d}}\Xi]Q] \vdash [\overrightarrow{t}/{}^{\mathsf{d}}\Xi]\{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow [\overrightarrow{t}/{}^{\mathsf{d}}\Xi]N \quad \text{By Lemma C.68 cor.}$$
$$\text{(equal or lesser height)} \quad ''$$

$$\Theta; \Gamma; [[\overrightarrow{t}/{}^{\mathsf{d}}\Xi]Q] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N \quad dom(\Theta) \cap dom({}^{\mathsf{d}}\Xi) = \emptyset$$
$$\Theta \vdash \Gamma' \leq^{+} \Gamma \quad \text{Given}$$
$$\Theta \vdash N \leq^{-} N' \quad \text{Given}$$
$$\Theta \vdash R \leq^{+} [\overrightarrow{t}/{}^{\mathsf{d}}\Xi]Q \quad \text{Subderivation}$$
$$\Theta; \Gamma'; [R] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N' \quad \text{By i.h. (smaller typing hgt.)}$$

- **Case**
$$\frac{\Theta, \overrightarrow{\varphi}; \Gamma; [R] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N}{\Theta; \Gamma; [R \wedge \overrightarrow{\varphi}] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N}\ \text{DeclMatch}\wedge$$

  - **Case**
$$\frac{\Theta \vdash R \leq^{+} R' \qquad \Theta \vdash \overrightarrow{\varphi}\,\mathsf{true}}{\Theta \vdash R \leq^{+} R' \wedge \overrightarrow{\varphi}}\ \leq^{+}\wedge\text{R}$$
    Similar to $\leq^{+}\exists$R subcase of DeclMatch$\exists$ case, but using Lemma C.67 (Typing Consequence) rather than Lemma C.68 (Index-Level Substitution).

- **Case**
$$\frac{\Theta; \Gamma \vdash e \Leftarrow N}{\Theta; \Gamma; [1] \vdash \{\langle\rangle \Rightarrow e\} \Leftarrow N}\ \text{DeclMatch1}$$

– **Case**

$$\frac{}{\Theta \vdash 1 \leq^{+} 1} \leq^{+}1$$

$\Theta \vdash \Gamma' \leq^{+} \Gamma$	Given
$\Theta \vdash N \leq^{-} N'$	Given
$\Theta;\Gamma \vdash e \Leftarrow N$	Subderivation
$\Theta;\Gamma' \vdash e \Leftarrow N'$	By i.h. (smaller typing hgt.)
$\Theta;\Gamma';[1] \vdash \{\langle\rangle \Rightarrow e\} \Leftarrow N'$	By DeclMatch1

- **Case**

$$\frac{\Theta;\Gamma,x_1 : R_1,x_2 : R_2 \vdash e_0 \Leftarrow N}{\Theta;\Gamma;[R_1 \times R_2] \vdash \{\langle x_1,x_2\rangle \Rightarrow e_0\} \Leftarrow N}\ \text{DeclMatch}\times$$

– **Case**

$$\frac{\Theta \vdash R_1' \leq^{+} R_1 \qquad \Theta \vdash R_2' \leq^{+} R_2}{\Theta \vdash R_1' \times R_2' \leq^{+} R_1 \times R_2} \leq^{+}\times$$

$\Theta \vdash \Gamma' \leq^{+} \Gamma$	Given
$\Theta \vdash R_1' \leq^{+} R_1$	Subderivation
$\Theta \vdash R_2' \leq^{+} R_2$	Subderivation
$\Theta \vdash \Gamma',x_1 : R_1',x_2 : R_2' \leq^{+} \Gamma,x_1 : R_1,x_2 : R_2$	Add entries
$\Theta;\Gamma,x_1 : R_1,x_2 : R_2 \vdash e_0 \Leftarrow N$	Subderivation
$\Theta;\Gamma',x_1 : R_1',x_2 : R_2' \vdash e_0 \Leftarrow N'$	By i.h. (smaller typing height)
$\Theta;\Gamma';[R_1' \times R_2'] \vdash \{\langle x_1,x_2\rangle \Rightarrow e_0\} \Leftarrow N'$	By DeclMatch$\times$

- **Case** DeclMatch$+$: Similar to case DeclMatch$\times$ case.

- **Case**

$$\frac{}{\Theta;\Gamma;[0] \vdash \{\} \Leftarrow N}\ \text{DeclMatch0}$$

  - **Case**

$$\frac{}{\Theta \vdash 0 \leq^{+} 0}\ \leq^{+}0$$

$\Theta;\Gamma';[0] \vdash \{\} \Leftarrow N'$ By DeclMatch0

- **Case**

$$\frac{\begin{array}{c}\mathscr{M}(F) \rightsquigarrow \overrightarrow{\alpha};\overrightarrow{\tau} \\ \overset{\mathsf{d}\div}{\div}\Theta \vdash \wr \overrightarrow{\alpha};F;\mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta;R \\ \Theta,{}^{\mathsf{d}}\Theta;\Gamma,x:R \vdash e \Leftarrow N\end{array}}{\Theta;\Gamma;[\{\nu:\mu F \mid \mathscr{M}(F)\}] \vdash \{\mathtt{into}(x) \Rightarrow e\} \Leftarrow N}\ \text{DeclMatch}\mu$$

  - **Case**

$$\frac{\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)}{\Theta \vdash \left\{\nu:\mu F' \;\middle|\; \mathscr{M}'(F')\right\} \leq^{+} \{\nu:\mu F \mid \mathscr{M}(F)\}}\ \leq^{+}\mu$$

From $\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)$

we extract the $\mathscr{M}'_0(F') \subseteq \mathscr{M}'(F)'$ such that $\Theta \vdash \mathscr{M}'_0(F') \geq_{\equiv} \mathscr{M}(F)$.

We extract $\mathscr{M}'_0(F') \rightsquigarrow \overrightarrow{\alpha'_0};\overrightarrow{\tau'_0}$

using the side judgment for Decl$\Leftarrow\mu$ (defined where the rule is).

$\overset{\mathsf{d}}{\div}\Theta \vdash \wr \overrightarrow{\alpha}; F; \mathscr{M}(F) \int \mathrel{\hat{=}} {}^{\mathsf{d}}\Theta; R$	Premise
$\cdot \vdash \overrightarrow{\alpha'_0}; F' \leq_{\overrightarrow{\tau}} \overrightarrow{\alpha}; F$	By inversion
$\overset{\mathsf{d}}{\div}\Theta \vdash \wr \overrightarrow{\alpha'_0}; F'; \mathscr{M}'_0(F') \int \mathrel{\hat{=}} {}^{\mathsf{d}}\Theta'_0; R'_0$	By Lemma C.77 (Unroll to Subtype)
$\Theta \vdash \exists^{\mathsf{d}}\Theta'_0. R'_0 \wedge {}^{\mathsf{d}}\Theta'_0 \leq \exists^{\mathsf{d}}\Theta. R \wedge {}^{\mathsf{d}}\Theta$	$''$
$\overset{\mathsf{d}}{\div}\Theta \vdash \wr \overrightarrow{\alpha'}; F'; \mathscr{M}'(F') \int \mathrel{\hat{=}} {}^{\mathsf{d}}\Theta'; R'$	By Lemma C.79 (Unroll Superlist)
$\Theta \vdash \exists^{\mathsf{d}}\Theta'. R' \wedge {}^{\mathsf{d}}\Theta' \leq \exists^{\mathsf{d}}\Theta'_0. R'_0 \wedge {}^{\mathsf{d}}\Theta'_0$	$''$
$\Theta \vdash \exists^{\mathsf{d}}\Theta'. R' \wedge {}^{\mathsf{d}}\Theta' \leq \exists^{\mathsf{d}}\Theta. R \wedge {}^{\mathsf{d}}\Theta$	By Lemma C.63 (Subtyping Transitive)

$$\text{Let } \overrightarrow{\varphi} = {}^{\mathsf{d}}\Theta - dom({}^{\mathsf{d}}\Theta).$$

$$\text{Let } \overrightarrow{\varphi'} = {}^{\mathsf{d}}\Theta' - dom({}^{\mathsf{d}}\Theta').$$

$$\Theta, \overline{{}^{\mathsf{d}}\Theta'}, \overrightarrow{\varphi'} \vdash R' \leq \exists {}^{\mathsf{d}}\Theta.\, R \wedge {}^{\mathsf{d}}\Theta \qquad \text{By inversion}$$

$$\Theta, \overline{{}^{\mathsf{d}}\Theta'}, \overrightarrow{\varphi'} \vdash R' \leq [\overrightarrow{t}/\overline{{}^{\mathsf{d}}\Theta}]R \qquad \text{By inversion}$$

$$\dot{-}\Theta, \overline{{}^{\mathsf{d}}\Theta'} \vdash \overrightarrow{t}/\overline{{}^{\mathsf{d}}\Theta} : \overline{{}^{\mathsf{d}}\Theta} \qquad ''$$

$$\Theta, \overline{{}^{\mathsf{d}}\Theta'}, \overrightarrow{\varphi'} \vdash [\overrightarrow{t}/\overline{{}^{\mathsf{d}}\Theta}]\overrightarrow{\varphi}\,\mathsf{true} \qquad ''$$

$$\Theta \vdash \Gamma' \leq^{+} \Gamma \qquad \text{Given}$$

$$\Theta, \overline{{}^{\mathsf{d}}\Theta'}, \overrightarrow{\varphi'} \vdash \Gamma' \leq^{+} \Gamma \qquad \text{By weakening}$$

$$\Theta, \overline{{}^{\mathsf{d}}\Theta'}, \overrightarrow{\varphi'} \vdash \Gamma', x : R' \leq^{+} \Gamma, x : [\overrightarrow{t}/\overline{{}^{\mathsf{d}}\Theta}]R \qquad \text{Add entry}$$

$$\text{Let } \sigma = \overrightarrow{t}/\overline{{}^{\mathsf{d}}\Theta}.$$

$$\Theta, \overline{{}^{\mathsf{d}}\Theta'}, \overrightarrow{\varphi'} \vdash \overrightarrow{t} : \overline{{}^{\mathsf{d}}\Theta} \qquad \text{By weakening}$$

$$\Theta, \overline{{}^{\mathsf{d}}\Theta'}, \overrightarrow{\varphi'} \vdash \sigma : \overline{{}^{\mathsf{d}}\Theta}, \overrightarrow{\varphi} \qquad \text{By repeated SubstProp}$$

$${}^{\mathsf{d}}\Theta = \overline{{}^{\mathsf{d}}\Theta}, \overrightarrow{\varphi} \qquad \text{By inspection of unrolling rules}$$

$$\Theta, \overline{{}^{\mathsf{d}}\Theta'}, \overrightarrow{\varphi'} \vdash \sigma : {}^{\mathsf{d}}\Theta \qquad \text{By equality}$$

$$\Theta, {}^{\mathsf{d}}\Theta; \Gamma, x : R \vdash e \Leftarrow N \qquad \text{Subderivation}$$

$$\Theta, \overline{{}^{\mathsf{d}}\Theta'}, \overrightarrow{\varphi'}; \Gamma, x : [\sigma]R \vdash e \Leftarrow N \qquad \text{By cor. of Lemma C.68}$$

$$\text{(lesser or equal hgt.)} \qquad ''$$

$$\Theta, \overline{{}^{\mathsf{d}}\Theta'}, \overrightarrow{\varphi'}; \Gamma', x : R' \vdash e \Leftarrow N' \qquad \text{By i.h. (smaller typing height)}$$

$$\Theta, {}^{\mathsf{d}}\Theta'; \Gamma', x : R' \vdash e \Leftarrow N' \qquad \text{By inspection of unrolling rules}$$

$$\Theta;\Gamma';[\{\nu : \mu F' \mid \mathscr{M}'(F')\}] \vdash \{\mathtt{into}(x) \Rightarrow e\} \Leftarrow N' \quad \text{By DeclMatch}\mu$$

(6) We case analyze the rule concluding the subtyping derivation.

- **Case**
$$\frac{\Theta \vdash P' \leq^{+} P}{\Theta \vdash \uparrow P' \leq^{-} \uparrow P} \leq^{-}\uparrow$$

  - **Case**
$$\frac{}{\Theta;\Gamma;[\uparrow P] \vdash \cdot \Rightarrow \uparrow P}\ \text{DeclSpineNil}$$

    ☞ $\Theta \vdash \uparrow P' \leq^{-} \uparrow P$ Given

    $\Theta \vdash \uparrow P'\ \mathsf{type}[\xi]$ Presupposed derivation

    ☞ $\Theta;\Gamma';[\uparrow P'] \vdash \cdot \Rightarrow \uparrow P'$ By DeclSpineNil

- **Case**
$$\frac{\Theta \vdash L' \leq^{-} N \qquad \Theta \vdash \overrightarrow{\varphi'}\ \mathsf{true}}{\Theta \vdash \overrightarrow{\varphi'} \supset L' \leq^{-} N} \leq^{-}\supset\text{L}$$

  $\Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P$ Given

  $\Theta \vdash \Gamma' \leq^{+} \Gamma$ Given

  $\Theta \vdash L' \leq^{-} N$ Subderivation

  $\Theta;\Gamma';[L'] \vdash s \Rightarrow \uparrow P'$ By i.h. (same typing height; smaller subtyping)

  ☞ $\Theta \vdash \uparrow P' \leq^{-} \uparrow P$ ″

  $\Theta \vdash \overrightarrow{\varphi'}\ \mathsf{true}$ Premise

  ☞ $\Theta;\Gamma';[\overrightarrow{\varphi'} \supset L'] \vdash s \Rightarrow \uparrow P'$ By DeclSpine$\supset$

- **Case** $\leq^-\forall$L: Similar to $\leq^-\supset$L case.

- **Case**
$$\frac{\Theta, \overrightarrow{\varphi} \vdash N' \leq^- L}{\Theta \vdash N' \leq^- \overrightarrow{\varphi} \supset L} \leq^-\supset\text{R}$$

  - **Case**
$$\frac{\Theta \vdash \overrightarrow{\varphi}\ \mathsf{true} \qquad \Theta;\Gamma;[L] \vdash s \Rightarrow \uparrow P}{\Theta;\Gamma;[\overrightarrow{\varphi} \supset L] \vdash s \Rightarrow \uparrow P}\ \text{DeclSpine}\supset$$

    $\Theta, \overrightarrow{\varphi} \vdash N' \leq^- L$ Subderivation

    $\Theta \vdash \Gamma' \leq^+ \Gamma$ Given

    $\Theta, \overrightarrow{\varphi} \vdash \Gamma' \leq^+ \Gamma$ By weakening

    $\Theta;\Gamma;[L] \vdash s \Rightarrow \uparrow P$ Subderivation

    $\Theta, \overrightarrow{\varphi};\Gamma;[L] \vdash s \Rightarrow \uparrow P$ By Lemma C.41 (Ix.-Level Weakening)

    (same height) $''$

    $\Theta, \overrightarrow{\varphi};\Gamma';[N'] \vdash s \Rightarrow \uparrow P'$ By i.h. (smaller typing hgt.)

    $\Theta, \overrightarrow{\varphi} \vdash \uparrow P' \leq^- \uparrow P$ $''$

    $\Theta \vdash \overrightarrow{\varphi}\ \mathsf{true}$ Premise

    $\Theta;\Gamma';[N'] \vdash s \Rightarrow \uparrow P'$ By Lemma C.67 (Typing Consequence)

    $\Theta \vdash \uparrow P' \leq^- \uparrow P$ By Lemma C.61 (Subtyping Consequence)

- **Case**
$$\frac{\Theta, {}^{\mathsf{d}}\Xi \vdash N' \leq^- M}{\Theta \vdash N' \leq^- \forall^{\mathsf{d}}\Xi.M} \leq^-\forall\text{R}$$

– **Case**
$$\dfrac{\overset{\mathsf{d}}{\div}\Theta \vdash \overrightarrow{t}/^{\mathsf{d}}\Xi : {}^{\mathsf{d}}\Xi \qquad \Theta;\Gamma;[[\overrightarrow{t}/^{\mathsf{d}}\Xi]M] \vdash s \Rightarrow \uparrow P}{\Theta;\Gamma;[\forall^{\mathsf{d}}\Xi.M] \vdash s \Rightarrow \uparrow P}\ \text{DeclSpine}\forall$$
Similar to DeclSpine$\supset$ subcase of $\leq^-\supset$R case, but using Lemma C.68 (Index-Level Substitution) rather than the consequence lemmas.

• **Case**
$$\dfrac{\Theta \vdash R \leq^+ R' \qquad \Theta \vdash L' \leq^- L}{\Theta \vdash R' \to L' \leq^- R \to L}\ \leq^-\!\to$$

– **Case**
$$\dfrac{\Theta;\Gamma \vdash v \Leftarrow R \qquad \Theta;\Gamma;[L] \vdash s_0 \Rightarrow \uparrow P}{\Theta;\Gamma;[R \to L] \vdash v, s_0 \Rightarrow \uparrow P}\ \text{DeclSpineApp}$$

$\Theta \vdash \Gamma' \leq^+ \Gamma$	Given
$\Theta;\Gamma \vdash v \Leftarrow R$	Subderivation
$\Theta \vdash R \leq^+ R'$	Premise
$\Theta;\Gamma' \vdash v \Leftarrow R'$	By i.h. (smaller typing height)
$\Theta;\Gamma;[L] \vdash s_0 \Rightarrow \uparrow P$	Subderivation
$\Theta \vdash L' \leq^- L$	Premise
$\Theta;\Gamma';[L'] \vdash s_0 \Rightarrow \uparrow P'$	By i.h. (smaller typing height)
$\Theta \vdash \uparrow P' \leq^- \uparrow P$	$''$
$\Theta;\Gamma';[R' \to L'] \vdash v, s_0 \Rightarrow \uparrow P'$	By DeclSpineApp

□

Lemma C.81 (Id. Subst.)**.**

If $\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma$ *and* $\mathscr{O}$ *is the subject of derivation* $\mathscr{D}$ *under* $\Theta;\Gamma$

then for any Θ' *and* Γ' *we have* $[\sigma]\mathscr{O} = [\sigma - id_{\Theta';\Gamma'}]\mathscr{O}$. *In particular,* $[\Theta;\Gamma/\Theta;\Gamma]\mathscr{O} = \mathscr{O}$.

Proof. By structural induction on $\mathscr{D}$. Straightforward. □

Lemma C.82 (Syntactic Substitution)**.** *Assume* $\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma$.

(1) If $\Theta;\Gamma \vdash h \Rightarrow P$

then there exists P' *such that* $\Theta_0 \vdash P' \leq^+ [\lfloor\sigma\rfloor]P$ *and* $\Theta_0;\Gamma_0 \vdash [\sigma]^{\mathsf{h}}h \Rightarrow P'$.

Moreover, either (a) $P' = [\lfloor\sigma\rfloor]P$ *or (b)* $P' = R$ *for some* R.

(2) If $\Theta;\Gamma \vdash g \Rightarrow \uparrow P$

then there exists P' *such that* $\Theta_0 \vdash \uparrow P' \leq^- [\lfloor\sigma\rfloor]\uparrow P$ *and* $\Theta_0;\Gamma_0 \vdash [\sigma]g \Rightarrow \uparrow P'$.

(3) If $\Theta;\Gamma \vdash v \Leftarrow P$ *then* $\Theta_0;\Gamma_0 \vdash [\sigma]v \Leftarrow [\lfloor\sigma\rfloor]P$.

(4) If $\Theta;\Gamma \vdash e \Leftarrow N$ *then* $\Theta_0;\Gamma_0 \vdash [\sigma]e \Leftarrow [\lfloor\sigma\rfloor]N$.

(5) If $\Theta;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$

then $\Theta_0;\Gamma_0;[[\lfloor\sigma\rfloor]P] \vdash [\sigma]\{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow [\lfloor\sigma\rfloor]N$.

(6) If $\Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P$ *then* $\Theta_0;\Gamma_0;[[\lfloor\sigma\rfloor]N] \vdash [\sigma]s \Rightarrow [\lfloor\sigma\rfloor]\uparrow P$.

Proof. By mutual induction on the structure of the given program typing derivation.

(1) • **Case**

$$\frac{(x : R) \in \Gamma}{\Theta;\Gamma \vdash x \Rightarrow R}\ \text{Decl}{\Rightarrow}\text{Var}$$

$(x : R) \in \Gamma$	By inversion on Decl⇒Var
$\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma$	Given
$\Theta = \Theta_1, \Theta_2$	By inversion
$\Gamma = \Gamma_1, x : R, \Gamma_2$	$''$
$\sigma = \sigma_1, v_1 : [\lfloor \sigma_1 \rfloor]R/x, \sigma_2$	$''$
$\Theta_0; \Gamma_0 \vdash \sigma_1 : \Theta_1; \Gamma_1$	$''$
$\Theta_0; \Gamma_0 \vdash \sigma_1, v_1 : [\lfloor \sigma_1 \rfloor]R/x : \Theta_1; \Gamma_1, x : R$	$''$
$\Theta_0; \Gamma_0 \vdash v_1 \Leftarrow [\lfloor \sigma_1 \rfloor]R$	$''$
$\Theta_1 \vdash \Gamma_1, x : R\ \mathsf{ctx}$	Presupposed derivation
$\overline{\Theta_1} \vdash R\ \mathsf{type}[_]$	By inversion
$FV(R) \subseteq dom(\Theta_1)$	Straightforward
$(\Theta_1, \Theta_2)\ \mathsf{ctx}$	Presupposed derivation

$dom(\Theta_1) \cap dom(\Theta_2) = \emptyset$	By inversion
$FV(R) \cap dom(\Theta_2) = \emptyset$	Follows from above
$[\lfloor \sigma_1 \rfloor]R = [\lfloor \sigma \rfloor]R$	Because $FV(R) \cap dom(\Theta_2) = \emptyset$

– **Case** $v_1 = x$:

$\Theta_0; \Gamma_0 \vdash x \Leftarrow [\sigma_1]R$ Rewrite above

$\Theta_0 \vdash R' \leq [\lfloor \sigma_1 \rfloor]R$ By inversion on Decl⇐Var

$(x : R') \in \Gamma_0$ ″

☞ $\Theta_0 \vdash R' \leq [\lfloor \sigma \rfloor]R$ By equality

$\overline{\Theta_0} \vdash \Gamma_0$ ctx Presupposed derivation

$\Theta_0; \Gamma_0 \vdash x \Rightarrow R'$ By Decl⇒Var

$\Theta_0; \Gamma_0 \vdash v_1 \Rightarrow R'$ By equality

☞ $\Theta_0; \Gamma_0 \vdash [\sigma]^{\mathsf{h}} x \Rightarrow R'$ By def. of $[-]^{\mathsf{h}}-$

– **Case** $v_1 \neq x$:

$\Theta_0; \Gamma_0 \vdash v_1 \Leftarrow [\lfloor \sigma_1 \rfloor]R$ Above

$\Theta_0; \Gamma_0 \vdash (v_1 : [\lfloor \sigma_1 \rfloor]R) \Rightarrow [\lfloor \sigma_1 \rfloor]R$ By Decl⇒ValAnnot

$\Theta_0; \Gamma_0 \vdash [\sigma]^{\mathsf{h}} x \Rightarrow [\lfloor \sigma_1 \rfloor]R$ By def. of $[-]^{\mathsf{h}}-$ $(v_1 \neq x)$

$\Theta_0; \Gamma_0 \vdash [\sigma]^{\mathsf{h}} x \Rightarrow [\lfloor \sigma \rfloor]R$ By equality

• **Case**

$$\frac{\overline{\Theta} \vdash P\ \mathsf{type}[\xi] \qquad \Theta; \Gamma \vdash v \Leftarrow P}{\Theta; \Gamma \vdash (v : P) \Rightarrow P}\ \text{Decl⇒ValAnnot}$$

$\Theta_0; \Gamma_0 \vdash [\sigma]v \Leftarrow [\lfloor \sigma \rfloor]P$ By i.h.

$\Theta_0; \Gamma_0 \vdash ([\sigma]v : [\lfloor \sigma \rfloor]P) \Rightarrow [\lfloor \sigma \rfloor]P$ By Decl⇒ValAnnot

$\overline{\Theta_0} \vdash [\lfloor \sigma \rfloor]P\ \mathsf{type}[_]$ Presupposed derivation

☞ $\Theta_0; \Gamma_0 \vdash [\sigma](v : P) \Rightarrow [\lfloor \sigma \rfloor]P$ By def. of $[-]-$

☞ $\Theta_0 \vdash [\lfloor \sigma \rfloor]P \leq [\lfloor \sigma \rfloor]P$ By Lemma C.62 (Subtyping Reflexive)

(2) • **Case**

$$\frac{\Theta;\Gamma \vdash h \Rightarrow \downarrow N \qquad \Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P}{\Theta;\Gamma \vdash h(s) \Rightarrow \uparrow P}\ \text{Decl}{\Rightarrow}\text{App}$$

$\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma$	Given
$\Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P$	Subderivation
$\Theta_0;\Gamma_0;[[\lfloor\sigma\rfloor]N] \vdash [\sigma]s \Rightarrow [\lfloor\sigma\rfloor]\uparrow P$	By i.h.
$\Theta_0;\Gamma_0;[[\lfloor\sigma\rfloor]N] \vdash [\sigma]s \Rightarrow \uparrow[\lfloor\sigma\rfloor]P$	By def. of subst.

$\Theta;\Gamma \vdash h \Rightarrow \downarrow N$	Subderivation

By i.h., either (a) $\Theta_0;\Gamma_0 \vdash [\sigma]^{\mathsf{h}}h \Rightarrow [\lfloor\sigma\rfloor]\downarrow N$; or (b) there exists R such that $\Theta_0 \vdash R \leq^{+} [\lfloor\sigma\rfloor]\downarrow N$ and $\Theta_0;\Gamma_0 \vdash [\sigma]h \Rightarrow R$.

Consider subcases (a) and (b):

– **Case** (a):

	$\Theta_0;\Gamma_0 \vdash [\sigma]^{\mathsf{h}}h \Rightarrow \downarrow[\lfloor\sigma\rfloor]N$	By def. of subst.
	$\Theta_0;\Gamma_0 \vdash ([\sigma]^{\mathsf{h}}h)([\sigma]s) \Rightarrow \uparrow[\lfloor\sigma\rfloor]P$	By Decl⇒App
☞	$\Theta_0;\Gamma_0 \vdash [\sigma](h(s)) \Rightarrow [\lfloor\sigma\rfloor]\uparrow P$	By def. of subst.
☞	$\Theta_0 \vdash [\lfloor\sigma\rfloor]\uparrow P \leq [\lfloor\sigma\rfloor]\uparrow P$	By Lemma C.62 (Subtyping Reflexive)

– **Case** (b):

$\Theta_0 \vdash R \leq^+ \downarrow[\lfloor\sigma\rfloor]N$ By def. of subst.

$R = \downarrow N'$ By inversion

$\Theta_0 \vdash N' \leq^+ [\lfloor\sigma\rfloor]N$ ″

$\Theta_0 \vdash \Gamma_0 \leq^+ \Gamma_0$ By repeated Lemma C.62

$\Theta_0; \Gamma_0; [N'] \vdash [\sigma]s \Rightarrow \uparrow P'$ By Lemma C.80

☞ $\Theta_0 \vdash \uparrow P' \leq^+ \uparrow[\lfloor\sigma\rfloor]P$ ″

☞ $\Theta_0; \Gamma_0 \vdash [\sigma](h(s)) \Rightarrow \uparrow P'$ By Decl⇒App and def. of subst.

- **Case**

$$\frac{\overline{\Theta} \vdash P \text{ type}[\xi] \qquad \Theta; \Gamma \vdash e \Leftarrow \uparrow P}{\Theta; \Gamma \vdash (e : \uparrow P) \Rightarrow \uparrow P}\ \text{Decl}\Rightarrow\text{ExpAnnot}$$

$\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma$ Given

$\Theta; \Gamma \vdash e \Leftarrow \uparrow P$ Subderivation

$\Theta_0; \Gamma_0 \vdash [\sigma]e \Leftarrow [\lfloor\sigma\rfloor]\uparrow P$ By i.h.

☞ $\Theta_0; \Gamma_0 \vdash [\sigma](e : \uparrow P) \Rightarrow [\lfloor\sigma\rfloor]\uparrow P$ By Decl⇒ExpAnnot and def. of subst.

☞ $\Theta_0 \vdash [\lfloor\sigma\rfloor]\uparrow P \leq [\lfloor\sigma\rfloor]\uparrow P$ By Lemma C.62 (Subtyping Reflexive) and def. of $[\lfloor - \rfloor]-$

(3)

- **Case**

$$\frac{(x : R') \in \Gamma \qquad \Theta \vdash R' \leq^+ R}{\Theta; \Gamma \vdash x \Leftarrow R}\ \text{Decl}\Leftarrow\text{Var}$$

$(x : R') \in \Gamma$	Premise
$\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma$	Given
$\Theta = \Theta_1, \Theta_2$	By inversion
$\Gamma = \Gamma_1, x : R', \Gamma_2$	$''$
$\sigma = \sigma_1, v_1 : [\lfloor \sigma_1 \rfloor] R'/x, \sigma_2$	$''$
$\Theta_0; \Gamma_0 \vdash \sigma_1 : \Theta_1; \Gamma_1$	$''$
$\Theta_0; \Gamma_0 \vdash v_1 \Leftarrow [\lfloor \sigma_1 \rfloor] R'$	$''$
$[\lfloor \sigma_1 \rfloor] R' = [\lfloor \sigma \rfloor] R'$	Similarly to case Decl$\Rightarrow$Var of part (1)
$\Theta_0; \Gamma_0 \vdash v_1 \Leftarrow [\lfloor \sigma \rfloor] R'$	By equality
$\Theta \vdash R' \leq^+ R$	Premise
$\Theta_0 \vdash [\lfloor \sigma \rfloor] R' \leq^+ [\lfloor \sigma \rfloor] R$	By Lemma C.59 (Sub. Syn. Subs)
$\Theta_0 \vdash \Gamma_0 \leq^+ \Gamma_0$	By Lemma C.62 (Subtyping Reflexive)
$\Theta_0; \Gamma_0 \vdash v_1 \Leftarrow [\lfloor \sigma \rfloor] R$	By Lemma C.80 (Subsumption Admissibility)
$\Theta_0; \Gamma_0 \vdash [\sigma] x \Leftarrow [\lfloor \sigma \rfloor] R$	By def. of $[-]-$ (here, x is a value)

- **Cases** Decl$\Leftarrow$1, Decl$\Leftarrow\times$, Decl$\Leftarrow+_k$: Straightforward.
- **Case**

$$\frac{\overset{\mathsf{d}}{\div}\Theta \vdash \overrightarrow{t}/^{\mathsf{d}}\Xi : {}^{\mathsf{d}}\Xi \qquad \Theta; \Gamma \vdash v \Leftarrow [\overrightarrow{t}/^{\mathsf{d}}\Xi] Q}{\Theta; \Gamma \vdash v \Leftarrow (\exists^{\mathsf{d}}\Xi . Q)}\ \text{Decl}\Leftarrow\exists$$

$\overset{\mathsf{d}}{\div}\Theta_0 \vdash \sigma\restriction_{\overset{\mathsf{d}}{\div}\Theta} : \overset{\mathsf{d}}{\div}\Theta$	By Lemma C.12 (Value-Det. Substitution)
$\Theta_0;\Gamma_0 \vdash [\sigma]v \Leftarrow [\lfloor\sigma\rfloor][\overrightarrow{t}/^{\mathsf{d}}\Xi]Q$	By i.h.
$\Theta_0;\Gamma_0 \vdash [\sigma]v \Leftarrow [[\lfloor\sigma\rfloor]\overrightarrow{t}/^{\mathsf{d}}\Xi]([\lfloor\sigma\rfloor]Q)$	By Lemma C.38 (Type/Functor Barendregt)
$\overset{\mathsf{d}}{\div}\Theta_0 \vdash [\lfloor\sigma\rfloor]\overrightarrow{t}/^{\mathsf{d}}\Xi : {}^{\mathsf{d}}\Xi$	By Lemma C.18 (Subst. on Substitution) and properties of restriction and $\lfloor-\rfloor$
$\Theta_0;\Gamma_0 \vdash [\sigma]v \Leftarrow [\lfloor\sigma\rfloor](\exists^{\mathsf{d}}\Xi.\, Q)$	By Decl$\Leftarrow\exists$ and def. of subst.

- **Case**

$$\frac{\Theta;\Gamma \vdash v \Leftarrow R \qquad \Theta \vdash \overrightarrow{\varphi}\,\mathsf{true}}{\Theta;\Gamma \vdash v \Leftarrow R \wedge \overrightarrow{\varphi}}\ \text{Decl}\Leftarrow\wedge$$

$\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma$	Given
$\Theta;\Gamma \vdash v \Leftarrow R$	Subderivation
$\Theta_0;\Gamma_0 \vdash [\sigma]v \Leftarrow [\lfloor\sigma\rfloor]R$	By i.h.
$\Theta \vdash \overrightarrow{\varphi}\,\mathsf{true}$	Premise
$\Theta_0 \vdash [\lfloor\sigma\rfloor]\overrightarrow{\varphi}\,\mathsf{true}$	By repeated Lemma C.30 (Prop. Truth Syn. Subs)
$\Theta_0;\Gamma_0 \vdash [\sigma]v \Leftarrow [\lfloor\sigma\rfloor](R \wedge \overrightarrow{\varphi})$	By Decl$\Leftarrow\wedge$ and def. of subst.

- **Case**

$$\frac{\nexists x.\, v' = \overrightarrow{\mathsf{inj}_{k_i}}^{i}\left(\overrightarrow{\langle __j, - \rangle}^{j} x\right) \qquad \mathscr{M}(F) \rightsquigarrow \overrightarrow{\alpha};\overrightarrow{\tau} \qquad \overset{\mathsf{d}}{\div}\Theta \vdash \wr\, \overrightarrow{\alpha};F;\mathscr{M}(F) \,\wr \doteq {}^{\mathsf{d}}\Theta;R \qquad \Theta;\Gamma \vdash v' \Leftarrow \exists^{\mathsf{d}}\Theta.\,(R \wedge {}^{\mathsf{d}}\Theta)}{\Theta;\Gamma \vdash \mathsf{into}(v') \Leftarrow \{\nu : \mu F \mid \mathscr{M}(F)\}}\ \text{Decl}\Leftarrow\mu$$

$[\lfloor\sigma\rfloor](\mathscr{M}(F)) \rightsquigarrow [\lfloor\sigma\rfloor]\overrightarrow{\alpha};\overrightarrow{\tau}$ Straightforward

$\dot{\div}^{\mathsf{d}}\Theta \vdash \wr[\lfloor\sigma\rfloor]\overrightarrow{\alpha};[\lfloor\sigma\rfloor]F;[\lfloor\sigma\rfloor].\mathscr{M}([\lfloor\sigma\rfloor]F)\wr \doteq [\lfloor\sigma\rfloor]^{\mathsf{d}}\Theta;[\lfloor\sigma\rfloor]R$ By Lemma C.57 with Lemma C.12 and subst. property

$\Theta_0;\Gamma_0 \vdash [\sigma]v' \Leftarrow [\lfloor\sigma\rfloor](\exists^{\mathsf{d}}\Theta.(R \wedge {}^{\mathsf{d}}\Theta))$ By i.h.

$\Theta_0;\Gamma_0 \vdash [\sigma]v' \Leftarrow \exists^{\mathsf{d}}\Theta.([\lfloor\sigma\rfloor]R \wedge [\lfloor\sigma\rfloor]^{\mathsf{d}}\Theta)$ By def. of subst.

$\Theta_0;\Gamma_0 \vdash [\sigma]v' \Leftarrow \exists[\lfloor\sigma\rfloor]^{\mathsf{d}}\Theta.([\lfloor\sigma\rfloor]R \wedge [\lfloor\sigma\rfloor]^{\mathsf{d}}\Theta)$ By def. of $\exists\Theta.-$

$\Theta_0;\Gamma_0 \vdash \mathtt{into}([\sigma]v') \Leftarrow \{v : \mu[\lfloor\sigma\rfloor]F \mid [\lfloor\sigma\rfloor]\mathscr{M}([\lfloor\sigma\rfloor]F)\}$ By Decl$\Leftarrow\mu$

$\Theta_0;\Gamma_0 \vdash [\sigma]\mathtt{into}(v') \Leftarrow [\lfloor\sigma\rfloor]\{v : \mu F \mid \mathscr{M}(F)\}$ By def. of subst.

- **Case** Decl$\Leftarrow\downarrow$: Straightforward.

(4)

- **Case** Decl$\Leftarrow\uparrow$: : Straightforward.

- **Case**

$$\frac{\Theta;\Gamma \vdash g \Rightarrow \uparrow(\exists^{\mathsf{d}}\Xi.R \wedge \overrightarrow{\psi}) \qquad \Theta, {}^{\mathsf{d}}\Xi, \overrightarrow{\psi};\Gamma, x : R \vdash e' \Leftarrow L}{\Theta;\Gamma \vdash \mathsf{let}\ x = g;\ e' \Leftarrow L}\ \mathrm{Decl}\Leftarrow\mathsf{let}$$

$\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma$ Given

$\Theta;\Gamma \vdash g \Rightarrow \uparrow(\exists^{\mathsf{d}}\Xi.R \wedge \overrightarrow{\psi})$ Subderivation

$\Theta_0;\Gamma_0 \vdash [\sigma]g \Rightarrow \uparrow P'$ By i.h.

$\Theta_0 \vdash \uparrow P' \leq^{-} [\lfloor\sigma\rfloor]\uparrow(\exists^{\mathsf{d}}\Xi.R \wedge \overrightarrow{\psi})$ ″

$\Theta_0 \vdash \uparrow P' \leq^{-} \uparrow\exists^{\mathsf{d}}\Xi.[\lfloor\sigma\rfloor]R \wedge [\lfloor\sigma\rfloor]\overrightarrow{\psi}$ By def. of subst.

$\Theta_0, {}^{\mathsf{d}}\Xi, [\lfloor\sigma\rfloor]\overrightarrow{\psi};\Gamma_0 \vdash \sigma, {}^{\mathsf{d}}\Xi/{}^{\mathsf{d}}\Xi : \Theta, {}^{\mathsf{d}}\Xi, \overrightarrow{\psi};\Gamma$ Lem. C.45

$\Theta_0, {}^{\mathsf{d}}\Xi, [\lfloor\sigma\rfloor]\overrightarrow{\psi};\Gamma_0, x : [\lfloor\sigma\rfloor]R \vdash \sigma, {}^{\mathsf{d}}\Xi/{}^{\mathsf{d}}\Xi, x : [\lfloor\sigma\rfloor]R/x : \Theta, {}^{\mathsf{d}}\Xi, \overrightarrow{\psi};\Gamma, x : R$ Lem. C.66

$\Theta, {}^{\mathsf{d}}\Xi, \overrightarrow{\psi}; \Gamma, x : R \vdash e' \Leftarrow L$ Subderivation

$\Theta_0, {}^{\mathsf{d}}\Xi, [\lfloor\sigma\rfloor]\overrightarrow{\psi}; \Gamma_0, x : [\lfloor\sigma\rfloor]R \vdash [\sigma]e' \Leftarrow [\lfloor\sigma\rfloor]L$ By i.h.

(and identity subst.)

$\Theta_0 \vdash P' \leq^+ \exists^{\mathsf{d}}\Xi.\, [\lfloor\sigma\rfloor]R \wedge [\lfloor\sigma\rfloor]\overrightarrow{\psi}$ By inversion

$P' = \exists^{\mathsf{d}}\Xi'.\, R' \wedge \overrightarrow{\psi'}$ Canonical form of P'

$\Theta_0, {}^{\mathsf{d}}\Xi', \overrightarrow{\psi'} \vdash R' \leq^+ [\overrightarrow{t}/{}^{\mathsf{d}}\Xi][\lfloor\sigma\rfloor]R$ By inversion

$\stackrel{\mathsf{d}}{\div}\Theta_0, {}^{\mathsf{d}}\Xi' \vdash \overrightarrow{t}/{}^{\mathsf{d}}\Xi : {}^{\mathsf{d}}\Xi$ $''$

$\Theta_0, {}^{\mathsf{d}}\Xi', \overrightarrow{\psi'} \vdash [\overrightarrow{t}/{}^{\mathsf{d}}\Xi][\lfloor\sigma\rfloor]\overrightarrow{\psi}\,\mathsf{true}$ $''$

Let $\sigma' = \overrightarrow{t}/{}^{\mathsf{d}}\Xi$.

$\Theta_0, {}^{\mathsf{d}}\Xi', \overrightarrow{\psi'} \vdash \sigma' : {}^{\mathsf{d}}\Xi, [\lfloor\sigma\rfloor]\overrightarrow{\psi}$ By rules and weakening

$\Theta_0, {}^{\mathsf{d}}\Xi', \overrightarrow{\psi'}; [\sigma']\Gamma_0, x : [\sigma']([\lfloor\sigma\rfloor]R) \vdash [\sigma']([\sigma]e') \Leftarrow [\sigma']([\lfloor\sigma\rfloor]L)$ By corollary of Lemma C.68 and def. of subst

$\Theta_0, {}^{\mathsf{d}}\Xi', \overrightarrow{\psi'}; \Gamma_0, x : [\sigma']([\lfloor\sigma\rfloor]R) \vdash [\sigma]e' \Leftarrow [\lfloor\sigma\rfloor]L \quad \emptyset = dom(\Theta_0) \cap dom({}^{\mathsf{d}}\Xi)$

$\Theta_0 \vdash \Gamma_0 \leq^+ \Gamma_0$ By repeated Lemma C.62

$\Theta_0, {}^{\mathsf{d}}\Xi', \overrightarrow{\psi'} \vdash \Gamma_0 \leq^+ \Gamma_0$ By repeated Lemma C.41

$\Theta_0, {}^{\mathsf{d}}\Xi', \overrightarrow{\psi'} \vdash \Gamma_0, x : R' \leq^+ \Gamma_0, x : [\sigma'][\lfloor\sigma\rfloor]R$ Add entry

$\Theta_0 \vdash [\lfloor\sigma\rfloor]L \leq^- [\lfloor\sigma\rfloor]L$ By Lemma C.62

$\Theta_0, {}^{\mathsf{d}}\Xi', \overrightarrow{\psi'} \vdash [\lfloor\sigma\rfloor]L \leq^- [\lfloor\sigma\rfloor]L$ By weakening

$\Theta_0, {}^{\mathsf{d}}\Xi', \overrightarrow{\psi'}; \Gamma_0, x : R' \vdash [\sigma]e' \Leftarrow [\lfloor\sigma\rfloor]L$ By Lemma C.80 (Subsumption Admissibility)

$\Theta_0; \Gamma_0 \vdash [\sigma](\mathsf{let}\ x = g;\ e') \Leftarrow [\lfloor\sigma\rfloor]L$ By Decl⇐let and def. of subst.

- **Case**

$$\dfrac{\Theta; \Gamma \vdash h \Rightarrow P \qquad \Theta; \Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow L}{\Theta; \Gamma \vdash \mathtt{match}\ h\ \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow L}\ \text{Decl}\Leftarrow\mathtt{match}$$

$\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma$ Given

$\Theta; \Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow L$ Subderivation

$\Theta_0; \Gamma_0; [[\lfloor\sigma\rfloor]P] \vdash [\sigma]\{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow [\lfloor\sigma\rfloor]L$ By i.h.

$\Theta; \Gamma \vdash h \Rightarrow P$ Subderivation

By i.h., either (a) $\Theta_0; \Gamma_0 \vdash [\sigma]^{\mathsf{h}} h \Rightarrow [\lfloor\sigma\rfloor]P$; or (b) there exists P' such that $\Theta_0 \vdash P' \leq^{+} [\lfloor\sigma\rfloor]P$ and $\Theta_0; \Gamma_0 \vdash [\sigma]^{\mathsf{h}} h \Rightarrow P'$.

Consider subcases (a) and (b):

  - **Case** (a): Apply Decl⇐match, and use def. of substitution.
  - **Case** (b):

$\Theta_0 \vdash \Gamma_0 \leq^{+} \Gamma_0$ By Lemma C.62

$\Theta_0 \vdash [\lfloor\sigma\rfloor]L \leq^{+} [\lfloor\sigma\rfloor]L$ By Lemma C.62

$\Theta_0; \Gamma_0; [P'] \vdash [\sigma]\{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow [\lfloor\sigma\rfloor]L$ By Lemma C.80

$\Theta_0; \Gamma_0 \vdash [\sigma](\mathtt{match}\ h\ \{r_i \Rightarrow e_i\}_{i \in I}) \Leftarrow [\lfloor\sigma\rfloor]L$ By Decl⇐match and def. of subst.

- **Case** Decl⇐λ: Similar to Decl⇐let case, but simpler.

- **Case**

$$\dfrac{\Theta \vdash \forall a \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, M \leq^{-} L \qquad \overbrace{\Theta, a \div \mathbb{N}; \Gamma, x : \downarrow \forall a' \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, a' < a \supset [a'/a]M}^{P'} \vdash e_0 \Leftarrow \forall^{\mathsf{d}}\Xi.\, M}{\Theta; \Gamma \vdash \mathtt{rec}\, x : (\forall a \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, M).\, e_0 \Leftarrow L}\ \mathrm{Decl}{\Leftarrow}\mathtt{rec}$$

$\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma$ Given

$\Theta \vdash \forall a \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, M \leq^{-} L$ Premise

$\Theta_0 \vdash \forall a \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, [\lfloor\sigma\rfloor]M \leq^{-} [\lfloor\sigma\rfloor]L$ By Lemma C.59 (Sub. Syn. Subs) and def. of subst

$\Theta, a \div \mathbb{N}; \Gamma, x : P' \vdash e_0 \Leftarrow \forall^{\mathsf{d}}\Xi.\, M$ Subderivation

By Lemma C.66 (Id. Subst. Extension) and Lemma C.43 (Index Id. Subs. Extension),

$$\Theta_0, a \div \mathbb{N}; \Gamma_0, x : [\lfloor\sigma\rfloor, a/a]P' \vdash \sigma, a/a, x : [\lfloor\sigma\rfloor, a/a]P'/x : \Theta, a \div \mathbb{N}; \Gamma, x : P'$$

By def. of $[-]-$ and P' and identity substitution,

$$\Theta_0, a \div \mathbb{N}; \Gamma_0, x : \downarrow \forall a' \overset{\mathsf{d}\cdot}{\div} \mathbb{N}.\, a' < a \supset [\lfloor\sigma\rfloor]([a'/a]M) \vdash \sigma, a/a, x : [\lfloor\sigma\rfloor]P'/x : \Theta, a \div \mathbb{N}; \Gamma, x : P'$$

By Lemma C.38 (Type/Functor Barendregt) and because $a' \notin dom(\Theta)$,

$$\Theta_0, a \div \mathbb{N}; \Gamma_0, x : \downarrow \forall a' \overset{\mathsf{d}\cdot}{\div} \mathbb{N}.\, a' < a \supset [a'/a]([\lfloor\sigma\rfloor]M) \vdash \sigma, a/a, x : [\lfloor\sigma\rfloor]P'/x : \Theta, a \div \mathbb{N}; \Gamma, x : P'$$

$\Theta_0, a \div \mathbb{N}; \Gamma_0, x : \downarrow \forall a' \overset{\mathsf{d}\div}{\div} \mathbb{N}.\, a' < a \supset [a'/a]([\lfloor\sigma\rfloor]M) \vdash [\sigma]e_0 \Leftarrow [\lfloor\sigma\rfloor]\forall^{\mathsf{d}}\Xi.\, M$ By i.h.

. . . and id. subst.

$\Theta_0; \Gamma_0 \vdash \mathtt{rec}\, x : (\forall a \overset{\mathsf{d}\div}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, [\lfloor\sigma\rfloor]M).\, [\sigma]e_0 \Leftarrow [\lfloor\sigma\rfloor]L$ By Decl⇐rec

$\Theta_0; \Gamma_0 \vdash [\sigma](\mathtt{rec}\, x : (\forall a \overset{\mathsf{d}\div}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, M).\, e_0) \Leftarrow [\lfloor\sigma\rfloor]L$ By def. of subst.

- **Cases** Decl⇐∀, DeclChkExpImp: Similar to Decl⇐λ case.
- **Case** Decl⇐Unreachable: Use Lemma C.1 (Filter Out Prog. Vars. Syn) and Lemma C.31 (Subst. Inconsistent).

(5)

- **Case** DeclMatch∃: Straightforward. Use Lemma C.45 (Ix.-Level Id. Subs. Extension).
- **Case** DeclMatch∧: Similar to DeclMatch∃ case.
- **Case** DeclMatch1: Straightforward.
- **Case** DeclMatch×: Straightforward. Use Lemma C.81 (Id. Subst), Lemma C.66 (Id. Subst. Extension) (twice), and the fact that program variables x cannot occur in types.
- **Case** DeclMatch+: Similar to DeclMatch× case.
- **Case** DeclMatch0: Straightforward.
- **Case**

$$\frac{\begin{array}{c}\mathscr{M}(F) \rightsquigarrow \overrightarrow{\alpha}; \overrightarrow{\tau} \\ \overset{\mathsf{d}\div}{\div}\Theta \vdash \wr \overrightarrow{\alpha}; F; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta; R \\ \Theta, {}^{\mathsf{d}}\Theta; \Gamma, x : R \vdash e \Leftarrow N\end{array}}{\Theta; \Gamma; [\{\nu : \mu F \mid \mathscr{M}(F)\}] \vdash \{\mathtt{into}(x) \Rightarrow e\} \Leftarrow N}\ \text{DeclMatch}\mu$$

We are given $\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma$. By inversion and Lemma C.12 (Value-Det. Substitution), $\overset{\mathsf{d}}{\div}\Theta_0;\cdot \vdash \lfloor\sigma\rfloor\restriction_{\overset{\mathsf{d}}{\div}\Theta} : \overset{\mathsf{d}}{\div}\Theta;\cdot$. That $[\lfloor\sigma\rfloor]\mathscr{M}([\lfloor\sigma\rfloor]F) \rightsquigarrow [\sigma]\overrightarrow{\alpha};\overrightarrow{\tau}$ follows from premise $\mathscr{M}(F) \rightsquigarrow \overrightarrow{\alpha};\overrightarrow{\tau}$ and property of substitution/restriction.

Consider the premise:

$$\overset{\mathsf{d}}{\div}\Theta \vdash \wr\, \overrightarrow{\alpha};F;\mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Theta;R$$

By Lemma C.57 (Unrolling Syntactic Substitution) and property of substitution/restriction,

$$\overset{\mathsf{d}}{\div}\Theta_0 \vdash \wr\, [\lfloor\sigma\rfloor]\overrightarrow{\alpha};[\lfloor\sigma\rfloor]F;[\lfloor\sigma\rfloor]\mathscr{M}([\lfloor\sigma\rfloor]F) \wr \doteq [\lfloor\sigma\rfloor]{}^{\mathsf{d}}\Theta;[\lfloor\sigma\rfloor]R$$

By Lemma C.45 (Ix.-Level Id. Subs. Extension), Lemma C.66 (Id. Subst. Extension), and Lemma C.81 (Id. Subst),

$$\Theta_0,[\lfloor\sigma\rfloor]{}^{\mathsf{d}}\Theta;\Gamma_0,x : [\lfloor\sigma\rfloor]R \vdash \sigma,{}^{\mathsf{d}}\Theta/{}^{\mathsf{d}}\Theta,x : [\lfloor\sigma\rfloor]R/x : \Theta,{}^{\mathsf{d}}\Theta;\Gamma,x : R$$

Consider the typing subderivation $\Theta,{}^{\mathsf{d}}\Theta;\Gamma,x : R \vdash e \Leftarrow N$. By the induction hypothesis and Lemma C.81,

$$\Theta,[\lfloor\sigma\rfloor]{}^{\mathsf{d}}\Theta;\Gamma_0,x : [\lfloor\sigma\rfloor]R \vdash [\sigma]e \Leftarrow [\lfloor\sigma\rfloor]N$$

By DeclMatchμ,

$$\Theta_0;\Gamma_0;[\{\nu : \mu[\lfloor\sigma\rfloor]F \mid [\lfloor\sigma\rfloor]\mathscr{M}([\lfloor\sigma\rfloor]F)\}] \vdash \{\mathtt{into}(x) \Rightarrow [\sigma]e\} \Leftarrow [\lfloor\sigma\rfloor]N$$

By definition of substitution,

$$\Theta_0; \Gamma_0; [[\lfloor \sigma \rfloor](\mathscr{M}(F))] \vdash [\sigma]\,\{\mathtt{into}(x) \Rightarrow e\} \Leftarrow [\lfloor \sigma \rfloor]N$$

(6)

- **Case** DeclSpine$\forall$: Straightforward. Use Lemma C.38 (Type/Functor Barendregt) and Lemma C.17 (Ix. Syntactic Substitution).
- **Case** DeclSpine$\supset$: Straightforward. Use Lemma C.30 (Prop. Truth Syn. Subs).
- **Cases** DeclSpineApp, DeclSpineNil: Straightforward. □

Appendix D

Unrefined System and Erasure

Lemma D.1 (Unrefined Syntactic Substitution)**.** *Assume* $\Gamma_0 \vdash \sigma : \Gamma$.

(1) If $\Gamma \vdash h \Rightarrow P$ *then* $\Gamma_0 \vdash [\sigma]h \Rightarrow P$.

(2) If $\Gamma \vdash g \Rightarrow \uparrow P$ *then* $\Gamma_0 \vdash [\sigma]g \Rightarrow \uparrow P$.

(3) If $\Gamma \vdash v \Leftarrow P$ *then* $\Gamma_0 \vdash [\sigma]v \Leftarrow P$.

(4) If $\Gamma \vdash e \Leftarrow N$ *then* $\Gamma_0 \vdash [\sigma]e \Leftarrow N$.

(5) If $\Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$ *then* $\Gamma_0; [P] \vdash [\sigma]\{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$.

(6) If $\Gamma; [N] \vdash s \Rightarrow \uparrow P$ *then* $\Gamma_0; [N] \vdash [\sigma]s \Rightarrow \uparrow P$.

Proof. Similar to Lemma C.82 (Syntactic Substitution), but simpler. □

It is straightforward to prove by structural induction on unrefined functor well-formedness that if $\vdash F$ functor then $[\![F]\!]$ is an endofunctor on the category of sets and $\mu\,[\![F]\!]$ is a set (as is any number of applications of $[\![F]\!]$ to a set). We leave these lemmas implicit. We later prove $[\![F]\!]$ is an endofunctor on the category of cpos and $\mu\,[\![F]\!]$ is a cpo.

Lemma D.2 (Functor Apps. Commute)**.** *If $F : \mathbf{C} \to \mathbf{D}$ is a functor and $n \in \mathbb{N}$ and $m \in \mathbb{N}$ then $F^n(F^m(X)) = F^m(F^n(X))$ for any object $X \in \mathbf{C}$; and similarly for any morphism f from Y to Z in $\mathbf{C}$.*

Proof. By induction on n. □

Lemma D.3 (Functor Monotone)**.** *If $\vdash F$ functor and $X, Y \in \mathbf{Set}$ and $X \subseteq Y$ then $[\![F]\!]\, X \subseteq [\![F]\!]\, Y$.*

Proof. By structural induction on F. □

Lemma D.4. *If $\vdash F$ functor and $n \in \mathbb{N}$ then $[\![F]\!]^n\, \emptyset \subseteq [\![F]\!]^{n+1}\, \emptyset$.*

Proof. By induction on n, using Lemma D.3 (Functor Monotone) in the inductive step. □

Lemma D.5 (Repeated Monotonicity)**.** *If $m, n \in \mathbb{N}$ and $m \leq n$ and $\vdash F$ functor then $[\![F]\!]^m\, \emptyset \subseteq [\![F]\!]^n\, \emptyset$.*

Proof. Follows from Lemma D.4 (and reflexivity of $\subseteq$). □

Lemma D.6 (Mu Superset)**.** *If $\vdash F$ functor and $n \in \mathbb{N}$ then $[\![F]\!]^n\, \emptyset \subseteq \mu\, [\![F]\!]$.*

Proof.

$$
\begin{aligned}
[\![F]\!]^n\, \emptyset &\subseteq \cup_{k \in \mathbb{N}}\, [\![F]\!]^k\, \emptyset && \text{Set theory} \\
&= \mu\, [\![F]\!] && \text{By def.}
\end{aligned}
$$

□

Lemma D.7 (Reverse Mu Superset)**.**

If $\vdash \mathscr{F}$ functor and $\vdash F$ functor and $V \in [\![\mathscr{F}]\!]\, (\mu\, [\![F]\!])$

then there exists $n \in \mathbb{N}$ such that $V \in [\![\mathscr{F}]\!]\, ([\![F]\!]^n\, \emptyset)$.

Proof.

- **Case** $\mathscr{F} = I$:

  By def., $[\![I]\!]\,(\mu\,[\![F]\!]) = \{\bullet\}$, so $V = \bullet$, which is in $\{\bullet\} = [\![I]\!]\,([\![F]\!]^n\,\emptyset)$ for all $n \in \mathbb{N}$.

- **Case** $\mathscr{F} = \underline{P}$:

  By def., $[\![\underline{P}]\!]\,(\mu\,[\![F]\!]) = [\![P]\!]$, so $V \in [\![P]\!]$. But $[\![P]\!] = [\![\underline{P}]\!]\,([\![F]\!]^n\,\emptyset)$ for all $n \in \mathbb{N}$.

- **Case** $\mathscr{F} = \mathrm{Id}$:

  By def., $[\![\mathrm{Id}]\!]\,(\mu\,[\![F]\!]) = \mu\,[\![F]\!]$, so $V \in \mu\,[\![F]\!]$. By def., $\mu\,[\![F]\!] = \cup_{k\in\mathbb{N}}\,[\![F]\!]\,\emptyset$, so there exists $n \in \mathbb{N}$ such that $V \in [\![F]\!]^n\,\emptyset = [\![\mathrm{Id}]\!]\,([\![F]\!]^n\,\emptyset)$.

- **Case** $\mathscr{F} = \hat{B} \otimes \hat{P}$:

  By def., $\left[\!\left[\hat{B} \otimes \hat{P}\right]\!\right]\,(\mu\,[\![F]\!]) = \left[\!\left[\hat{B}\right]\!\right]\,(\mu\,[\![F]\!]) \times \left[\!\left[\hat{P}\right]\!\right]\,(\mu\,[\![F]\!])$, so there exist $V_1 \in \left[\!\left[\hat{B}\right]\!\right]\,(\mu\,[\![F]\!])$ and $V_2 \in \left[\!\left[\hat{P}\right]\!\right]\,(\mu\,[\![F]\!])$ such that $V = (V_1, V_2)$. By i.h., there exists n_1 such that $V_1 \in \left[\!\left[\hat{B}\right]\!\right]\,([\![F]\!]^{n_1}\,\emptyset)$. By i.h., there exists n_2 such that $V_2 \in \left[\!\left[\hat{P}\right]\!\right]\,([\![F]\!]^{n_2}\,\emptyset)$. Let $n = \max\{n_1, n_2\}$. By Lemma D.5 (Repeated Monotonicity), $V_1 \in \left[\!\left[\hat{B}\right]\!\right]\,([\![F]\!]^n\,\emptyset)$ and $V_2 \in \left[\!\left[\hat{P}\right]\!\right]\,([\![F]\!]^n\,\emptyset)$. Therefore, $V = (V_1, V_2) \in \left[\!\left[\hat{B}\right]\!\right]\,([\![F]\!]^n\,\emptyset) \times \left[\!\left[\hat{P}\right]\!\right]\,([\![F]\!]^n\,\emptyset) = \left[\!\left[\hat{B} \otimes \hat{P}\right]\!\right]\,([\![F]\!]^n\,\emptyset)$, as desired.

- **Case** $\mathscr{F} = F_1 \oplus F_2$:

  By def., $[\![F_1 \oplus F_2]\!]\,(\mu\,[\![F]\!]) = [\![F_1]\!]\,(\mu\,[\![F]\!]) \uplus [\![F_2]\!]\,(\mu\,[\![F]\!])$, so there exist $j \in \{1,2\}$ and $V' \in \left[\!\left[F_j\right]\!\right]\,(\mu\,[\![F]\!])$ such that $V = (j, V')$. By i.h., there exists n such that $V' \in \left[\!\left[F_j\right]\!\right]\,([\![F]\!]^n\,\emptyset)$. Therefore $V = (j, V') \in [\![F_1]\!]\,([\![F]\!]^n\,\emptyset) \uplus [\![F_2]\!]\,([\![F]\!]^n\,\emptyset) = [\![F_1 \oplus F_2]\!]\,([\![F]\!]^n\,\emptyset)$.

□

Lemma D.8 (Mu is Fixed Point)**.** *If* $\vdash F$ functor *then* $\mu\,[\![F]\!] = [\![F]\!]\,(\mu\,[\![F]\!])$.

Proof.

- "$\subseteq$"

  Suppose $V \in \mu\,[\![F]\!]$. Then there exists $n > 0$ such that $V \in [\![F]\!]^n\,\emptyset$. By Lemma D.6 (Mu Superset), $[\![F]\!]^{n-1}\,\emptyset \subseteq \mu\,[\![F]\!]$. Now,

  $$\begin{aligned}
  [\![F]\!]^n\,\emptyset &= [\![F]\!]^{n-1}\,([\![F]\!]\,\emptyset) && \text{By def.} \\
  &= [\![F]\!]\,([\![F]\!]^{n-1}\,\emptyset) && \text{By Lemma D.2 (Functor Apps. Commute)} \\
  &\subseteq [\![F]\!]\,(\mu\,[\![F]\!]) && \text{By Lemma D.3 (Functor Monotone)}
  \end{aligned}$$

- "$\supseteq$"

  Suppose $V \in [\![F]\!]\,\mu\,[\![F]\!]$. By Lemma D.7 (Reverse Mu Superset), there exists $n \in \mathbb{N}$ such that $V \in [\![F]\!]\,([\![F]\!]^n\,\emptyset)$. But

  $$\begin{aligned}
  [\![F]\!]\,([\![F]\!]^n\,\emptyset) &= [\![F]\!]^n\,([\![F]\!]\,\emptyset) && \text{By Lemma D.2 (Functor Apps. Commute)} \\
  &= [\![F]\!]^{n+1}\,\emptyset && \text{By def.} \\
  &\subseteq \mu\,[\![F]\!] && \text{By Lemma D.6 (Mu Superset)}
  \end{aligned}$$

  □

Definition D.1 (Predomain)**.** A *complete partial order (cpo)* or *predomain* is a poset $(D, \sqsubseteq)$ that is chain-complete, i.e., such that every chain $d_0 \sqsubseteq d_1 \sqsubseteq \cdots$ in D has a least upper bound (*lub*) $\sqcup_{k\in\mathbb{N}} d_k \in D$. We often write $\sqsubseteq_D$ for the partial order of a predomain with set D.

Definition D.2 (Domain)**.** A *complete pointed partial order (cppo)* or *domain* is a triple $(D, \sqsubseteq, \bot)$ where $(D, \sqsubseteq)$ is a predomain, $\bot \in D$, and $\bot \sqsubseteq d$ for all $d \in D$. We often write $\bot_D$ for the bottom element $\bot$ of a domain with set D.

Definition D.3 (Continuous)**.** A function $f : D \to E$ of predomains (or domains) is *continuous* if f is monotone (that is, if $d \sqsubseteq_D d'$, then $f(d) \sqsubseteq_E f(d')$) and preserves least upper bounds (that is, if $d_0 \sqsubseteq d_1 \sqsubseteq \cdots$ is a chain in D, then $\sqcup_{k\in\mathbb{N}} f(d_k) = f(\sqcup_{k\in\mathbb{N}} d_k)$).

Lemma D.9 (Predomains and Domains are Cats)**.**

(1) Predomains and continuous functions form a category **Cpo**.

(2) Domains and continuous functions form a category **Cppo**.

Proof. The composition of continuous functions is continuous, the identity function is continuous, and function composition is associative. □

Definition D.4 (Cpo)**.** Define the category of predomains **Cpo** to have cpos as objects and continuous functions as morphisms.

Definition D.5 (Cppo)**.** Define the category of domains **Cppo** to have cppos as objects and continuous functions as morphisms.

Lemma D.10 (Predomain Constructions)**.**

(1) The empty set $\emptyset$ *with empty ordering* $\emptyset$ *is a predomain.*

(2) A singleton $\{d\}$ *with ordering* $d \sqsubseteq d$ *is a predomain.*

(3) If D *and* E *are predomains*
then $D \times E$ *with component-wise ordering* $\sqsubseteq_{D\times E}$ *is a predomain.*

(4) If D *and* E *are predomains*
then $D \uplus E$ *with injection-wise ordering* $\sqsubseteq_{D\uplus E}$ *is a predomain.*

(5) If D *and* E *are predomains*
then $D \Rightarrow E = \{f : D \to E \mid f \text{ is continuous}\}$
with pointwise ordering $\sqsubseteq_{D\Rightarrow E}$ *is a predomain.*

Proof. Each part is straightforward. □

Lemma D.11 (Least Fixed Point)**.** *If* $D \in$ **Cppo** *and* $f : D \to D$ *is continuous*
then f *has a least fixed point* $\sqcup_{k\in\mathbb{N}} f^k(\bot_D)$ *in* D.

Proof. See proof of Theorem 4.12 in the textbook of Gunter [1993]. □

Lemma D.12. *If* $\vdash \mathscr{F}$ functor *and* $\vdash F$ functor *and* $d \sqsubseteq_{[\![\mathscr{F}]\!]([\![F]\!]^m\emptyset)} d'$
then for all $n > m$*, we have* $d \sqsubseteq_{[\![\mathscr{F}]\!]([\![F]\!]^n\emptyset)} d'$.

Proof. By lexicographic induction, first, on m and, second, on the structure of $\mathscr{F}$. □

Lemma D.13 (Mu Chain-Complete)**.** *If* $\vdash \mathscr{F}$ functor *and* $\vdash F$ functor
and $[\![\mathscr{F}]\!] : \mathbf{Cpo} \to \mathbf{Cpo}$ *and* $[\![F]\!] : \mathbf{Cpo} \to \mathbf{Cpo}$
and $d_0 \sqsubseteq_{[\![\mathscr{F}]\!]([\![F]\!]^{n_0}\emptyset)} d_1 \sqsubseteq_{[\![\mathscr{F}]\!]([\![F]\!]^{n_1}\emptyset)} \cdots d_k \sqsubset_{[\![\mathscr{F}]\!]([\![F]\!]^{n_k}\emptyset)} \cdots$
then there exists a least upper bound $\sqcup_{k\in\mathbb{N}} d_k$ *in* $[\![\mathscr{F}]\!]\,(\mu\,[\![F]\!])$.

Proof. By lexicographic induction, first, on $\min\{n_k \mid k \in \mathbb{N}\}$ and, second, on $\mathscr{F}$ structure.
For all k, let $D_k = [\![F]\!]^{n_k}\emptyset$.

- **Case** $\mathscr{F} = I$:
  By def., for all $k \in \mathbb{N}$, we have $[\![I]\!]\,D_k = \{\bullet\}$; so $d_k = \bullet$ for all k. then $\sqcup_k d_k = \bullet$.

- **Case** $\mathscr{F} = \underline{Q}$:
  For all k, we have $[\![\underline{Q}]\!]\,D_k = [\![Q]\!]$; so $d_k \in [\![Q]\!]$ for all k. By Lemma D.10 (Predomain Constructions), $\emptyset \in \mathbf{Cpo}$. Therefore, by def. of $[\![-]\!]$ and because $[\![\mathscr{F}]\!] : \mathbf{Cpo} \to \mathbf{Cpo}$, we have $[\![Q]\!] = [\![\underline{Q}]\!]\,\emptyset \in \mathbf{Cpo}$. Therefore, the chain $d_0 \sqsubseteq \cdots$ has a least upper bound $\sqcup_k d_k$ in $[\![Q]\!] = [\![\underline{Q}]\!]\,(\mu\,[\![F]\!])$.

- **Case** $\mathscr{F} = \mathrm{Id}$:
  By def., for all k, we have $[\![\mathrm{Id}]\!]\,D_k = D_k = [\![F]\!]^{n_k}\emptyset = [\![F]\!]\,([\![F]\!]^{n_k-1}\emptyset)$. For all k, we have $n_k > 0$ (otherwise d_k would not exist), so $n_k - 1 \in \mathbb{N}$ for all k. Now, $\min\{n_k - 1 \mid k \in \mathbb{N}\} < \min\{n_k \mid k \in \mathbb{N}\}$, and we are given $[\![F]\!] : \mathbf{Cpo} \to \mathbf{Cpo}$, so by

the i.h., the chain $d_0 \sqsubseteq \cdots$ has a least upper bound $\sqcup_k d_k$ in $[\![F]\!]\,(\mu\,[\![F]\!])$; and the latter equals $\mu\,[\![F]\!]$ by Lemma D.8 (Mu is Fixed Point), which equals $[\![\mathrm{Id}]\!]\,(\mu\,[\![F]\!])$ by definition.

- **Case** $\mathscr{F} = \hat{B} \otimes \hat{P}$:

  By def., for all k, we have $[\![\hat{B} \otimes \hat{P}]\!]\,D_k = [\![\hat{B}]\!]\,D_k \times [\![\hat{P}]\!]\,D_k$; so for all k, there exist $d_{k1} \in [\![\hat{B}]\!]\,D_k$ and $d_{k2} \in [\![\hat{P}]\!]\,D_k$ such that $d_k = (d_{k1}, d_{k2})$. By i.h., chain $d_{01} \sqsubseteq \cdots$ has a least upper bound $\sqcup_k d_{k1}$ in $[\![\hat{B}]\!]\,(\mu\,[\![F]\!])$. By i.h., chain $d_{02} \sqsubseteq \cdots$ has a least upper bound $\sqcup_k d_{k2}$ in $[\![\hat{P}]\!]\,(\mu\,[\![F]\!])$. Then $\sqcup_k d_k = (\sqcup_k d_{k1}, \sqcup_k d_{k2})$ is the least upper bound for chain $d_k \sqsubseteq \cdots$ in $[\![\hat{B}]\!]\,(\mu\,[\![F]\!]) \times [\![\hat{P}]\!]\,(\mu\,[\![F]\!]) = [\![B \otimes \hat{P}]\!]\,(\mu\,[\![F]\!])$.

- **Case** $\mathscr{F} = F_1 \oplus F_2$:

  By def., for all k, we have $[\![F_1 \oplus F_2]\!]\,D_k = [\![F_1]\!]\,D_k \uplus [\![F_2]\!]\,D_k$; so for all k, there exists $d'_k \in [\![F_j]\!]\,D_k$ such that $d_k = (j, d'_k)$. By i.h., the chain $d'_0 \sqsubseteq \cdots$ has a least upper bound $\sqcup_k d'_k$ in $[\![F_j]\!]\,(\mu\,[\![F]\!])$. Then $\sqcup_k d_k = (j, \sqcup_k d'_k)$ is the least upper bound for chain $d_k \sqsubseteq \cdots$ in $[\![F_1]\!]\,(\mu\,[\![F]\!]) \uplus [\![F_2]\!]\,(\mu\,[\![F]\!]) = [\![F_1 \oplus F_2]\!]\,(\mu\,[\![F]\!])$. □

Lemma D.14 (Repeated Cpo Functor)**.** *If* F : **Cpo** $\to$ **Cpo** *and* $X \in$ **Cpo**
then for all $k \in \mathbb{N}$*, we have* $F^k(X) \in$ **Cpo**.

Proof. By induction on k. □

Lemma D.15 (Unref. Type Denotations)**.**

(1) If $\vdash P$ type *then* $[\![P]\!] \in$ **Cpo**.

(2) If $\vdash N$ type *then* $[\![N]\!] \in$ **Cppo**.

(3) If $\vdash \mathscr{F}$ functor *then* $[\![\mathscr{F}]\!]$ *is a* **Cpo** *endofunctor.*

Proof. By mutual induction on the structure of P, N or $\mathscr{F}$.

(1)

- **Case** $P = 0$:

$$\begin{aligned} [\![0]\!] &= \emptyset && \text{By def.} \\ &\in \mathbf{Cpo} && \text{By Lemma D.10 (Predomain Constructions)} \end{aligned}$$

- **Case** $P = P_1 + P_2$:

$$\begin{aligned} [\![P_1]\!] &\in \mathbf{Cpo} && \text{By i.h.} \\ [\![P_2]\!] &\in \mathbf{Cpo} && \text{By i.h.} \\ [\![P_1 + P_2]\!] &= ([\![P_1]\!] \uplus [\![P_2]\!], \sqsubseteq_{[\![P_1]\!] \uplus [\![P_2]\!]}) && \text{By def.} \\ &\in \mathbf{Cpo} && \text{By Lemma D.10 (Predomain Constructions)} \end{aligned}$$

- **Case** $P = 1$:

  Similar to $P = 0$ case.

- **Case** $P = P_1 \times P_2$:

  Similar to $P = P_1 + P_2$ case.

- **Case** $P = \downarrow N$:

$$\begin{aligned} [\![\downarrow N]\!] &= ([\![N]\!], \sqsubseteq_{[\![N]\!]}) && \text{By def.} \\ &\in \mathbf{Cpo} && \text{By i.h.} \end{aligned}$$

- **Case** $P = \mu F$:

  We first show that $([\![\mu F]\!], \sqsubseteq_{[\![\mu F]\!]})$ is a poset.

  Suppose $V \in [\![\mu F]\!]$. Then there exists $l \in \mathbb{N}$ such that $V \in [\![F]\!]^l \emptyset$. By Lemma D.10 (Predomain Constructions), $\emptyset \in \mathbf{Cpo}$. By i.h. (part (3)), $[\![F]\!] : \mathbf{Cpo} \to \mathbf{Cpo}$. By Lemma D.14 (Repeated Cpo Functor), $[\![F]\!]^l \emptyset \in \mathbf{Cpo}$; therefore (by reflexivity), $V \sqsubseteq_{[\![F]\!]^l \emptyset} V$. By def., $V \sqsubseteq_{[\![\mu F]\!]} V$, so $\sqsubseteq_{[\![\mu F]\!]}$ is reflexive.

Suppose $V_1 \sqsubseteq_{[\![\mu F]\!]} V_2$ and $V_2 \sqsubseteq_{[\![\mu F]\!]} V_3$. Then there exist l_1 and l_2 such that $V_1 \sqsubseteq_{[\![F]\!]^{l_1}\emptyset} V_2$ and $V_2 \sqsubseteq_{[\![F]\!]^{l_2}\emptyset} V_3$. Let $l = \max(l_1, l_2)$. By Lemma D.12, $V_1 \sqsubseteq_{[\![F]\!]^{l}\emptyset} V_2$ and $V_2 \sqsubseteq_{[\![F]\!]^{l}\emptyset} V_3$. By Lemma D.10 (Predomain Constructions), $\emptyset \in \mathbf{Cpo}$. By i.h. (part (3)), $[\![F]\!] : \mathbf{Cpo} \to \mathbf{Cpo}$. By Lemma D.14 (Repeated Cpo Functor), $[\![F]\!]^l\,\emptyset \in \mathbf{Cpo}$; therefore (by transitivity of $\sqsubseteq_{[\![F]\!]^{l}\emptyset}$), $V_1 \sqsubseteq_{[\![F]\!]^{l}\emptyset} V_3$. By def., $V_1 \sqsubseteq_{[\![\mu F]\!]} V_3$, so $\sqsubseteq_{[\![\mu F]\!]}$ is transitive.

Similarly, $\sqsubseteq_{[\![\mu F]\!]}$ is antisymmetric, so $[\![\mu F]\!]$ is a poset.

To conclude, we show that $[\![\mu F]\!]$ is chain-complete. Suppose $d_0 \sqsubseteq \cdots$ is a chain in $[\![\mu F]\!]$. By def., for all k, there exists n_k such that $d_k \sqsubseteq_{[\![F]\!]^{n_k+1}\emptyset} d_{k+1}$. Above, we have $[\![F]\!] : \mathbf{Cpo} \to \mathbf{Cpo}$. By Lemma D.13 (Mu Chain-Complete), the chain $d_0 \sqsubseteq \cdots$ has a least fixed point $\sqcup_k d_k$ in $[\![F]\!]\,(\mu\,[\![F]\!])$; and the latter equals $\mu\,[\![F]\!]$ by Lemma D.8 (Mu is Fixed Point), which equals $[\![\mu F]\!]$ by definition.

(2) • **Case** $N = \uparrow P$:

By def.,

$$[\![\uparrow P]\!] = ([\![P]\!] \uplus \{\bot_\uparrow\}, \left\{ ((1,d),(1,d')) \;\middle|\; d \sqsubseteq_{[\![P]\!]} d' \right\} \cup \left\{ ((2,\bot_\uparrow),d) \;\middle|\; d \in [\![\uparrow P]\!] \right\}, (2,\bot_\uparrow))$$

By i.h. (part (1)), $[\![P]\!] \in \mathbf{Cpo}$. This together with the definition of $\sqsubseteq_{[\![\uparrow P]\!]}$ above, that is,

$$\left\{ ((1,d),(1,d')) \;\middle|\; d \sqsubseteq_{[\![P]\!]} d' \right\} \cup \left\{ ((2,\bot_\uparrow),d) \;\middle|\; d \in [\![\uparrow P]\!] \right\}$$

implies that $[\![\uparrow P]\!]$ is a poset. The definition also implies that $\bot_{[\![\uparrow P]\!]} = (2,\bot_\uparrow)$ is the bottom element of $[\![\uparrow P]\!]$.

To conclude this case, we show that $[\![\uparrow P]\!]$ is chain-complete. Suppose $d_0 \sqsubseteq \cdots$

is a chain in $[\![\uparrow P]\!]$. If for all k, $d_k = (2, \bot_\uparrow)$, then $\sqcup_k d_k = (2, \bot_\uparrow)$. Else, if there exist j and d'_j such that $d_j = (1, d'_j)$, then, by definition of $\sqsubseteq_{[\![\uparrow P]\!]}$ (above), for all $m > j$, there exists $d'_m \in [\![P]\!]$ such that $d_m = (1, d'_m)$ and $d'_m \sqsubseteq_{[\![P]\!]} d'_{m+1}$; by the i.h., $[\![P]\!] \in \mathbf{Cpo}$, hence chain-complete, so the chain $d'_j \sqsubseteq \cdots$ has a least upper bound $\sqcup_{k \geq j} d'_k$ in $[\![P]\!]$; then $\sqcup_k d_k = (1, \sqcup_{k \geq j} d'_k)$ is the least upper bound of chain $d_0 \sqsubseteq \cdots$ in $[\![P]\!] \uplus \{\bot_\uparrow\} = [\![\uparrow P]\!]$.

- **Case** $N = P \to N_0$:

  By def., $[\![P \to N_0]\!]$ is the set $[\![P]\!] \Rightarrow [\![N_0]\!]$ of continuous functions from $[\![P]\!]$ to $[\![N_0]\!]$. By i.h., $[\![P]\!] \in \mathbf{Cpo}$. By i.h., $[\![N_0]\!] \in \mathbf{Cppo}$; therefore $[\![N_0]\!] \in \mathbf{Cpo}$. By Lemma D.10 (Predomain Constructions), $[\![P]\!] \Rightarrow [\![N_0]\!]$ (with pointwise ordering) is a **Cpo**.

  To conclude this case, we show $\bot_{[\![P \to N_0]\!]} \sqsubseteq f$ for all $f \in [\![P \to N_0]\!]$. To this end, suppose $f \in [\![P \to N_0]\!]$ and $x \in [\![P]\!]$. Then:

  $$\begin{aligned} \bot_{[\![P \to N_0]\!]}\, x &= \bot_{[\![N_0]\!]} && \text{By def.} \\ &\sqsubseteq f(x) && \text{By i.h., } [\![N_0]\!] \in \mathbf{Cppo} \end{aligned}$$

(3) For each case below, X is an arbitrary predomain (**Cpo**), and $f \in \mathrm{Hom}_{\mathbf{Cpo}}(X, Y)$ is an arbitrary continuous function.

- **Case** $\mathscr{F} = I$:

  $$\begin{aligned} [\![I]\!]\, X &= \{\bullet\} && \text{By def.} \\ &\in \mathbf{Cpo} && \text{By Lemma D.10 (Predomain Constructions)} \end{aligned}$$

  Further, $[\![I]\!]\, f = id_{\{\bullet\}}$ is continuous because identity functions are continuous.

- **Case** $\mathscr{F} = \underline{Q}$:

$$\begin{aligned}
[\![\underline{Q}]\!]\, X &= [\![Q]\!] && \text{By def.}\\
&\in \mathbf{Cpo} && \text{By i.h.}
\end{aligned}$$

Further, $[\![\underline{Q}]\!]\, f = id_{[\![Q]\!]}$ is continuous because identity functions are continuous.

- **Case** $\mathscr{F} = \mathrm{Id}$:

$$\begin{aligned}
[\![\mathrm{Id}]\!]\, X &= X && \text{By def.}\\
&\in \mathbf{Cpo} && \text{Given}
\end{aligned}$$

Further, $[\![\mathrm{Id}]\!]\, f = f$ is continuous.

- **Case** $\mathscr{F} = \hat{B} \otimes \hat{P}$:

$$\begin{aligned}
X &\in \mathbf{Cpo} && \text{Given}\\
[\![\hat{B}]\!]\, X &\in \mathbf{Cpo} && \text{By i.h.}\\
[\![\hat{P}]\!]\, X &\in \mathbf{Cpo} && \text{By i.h.}\\
[\![\hat{B} \otimes \hat{P}]\!]\, X &= [\![\hat{B}]\!]\, X \times [\![\hat{P}]\!]\, X && \text{By def.}\\
&\in \mathbf{Cpo} && \text{By Lemma D.10 (Predomain Constructions)}
\end{aligned}$$

Further, $[\![\hat{B} \otimes \hat{P}]\!]\, f = (([\![\hat{B}]\!]\, f) \circ \pi_1, ([\![\hat{P}]\!]\, f) \circ \pi_2)$ is continuous by two uses of the i.h., by the fact that projections π_k are continuous, by the fact that the composition of continuous functions is continuous, and by the fact that the universal pair (g_1, g_2) of continuous functions g_1 and g_2 is continuous.

- **Case** $\mathscr{F} = F_1 \oplus F_2$:

$$
\begin{aligned}
X &\in \mathbf{Cpo} && \text{Given}\\
[\![F_1]\!]\,X &\in \mathbf{Cpo} && \text{By i.h.}\\
[\![F_2]\!]\,X &\in \mathbf{Cpo} && \text{By i.h.}\\
[\![F_1 \oplus F_2]\!]\,X = [\![F_1]\!]\,X \uplus [\![F_2]\!]\,X &&& \text{By def.}\\
&\in \mathbf{Cpo} && \text{By Lemma D.10 (Predomain Constructions)}
\end{aligned}
$$

Further, $[\![F_1 \oplus F_2]\!]\,f = [inj_1 \circ ([\![F_1]\!]\,f), inj_2 \circ ([\![F_2]\!]\,f)]$ is continuous by two uses of the i.h., by the fact that injections inj_k are continuous, by the fact that the composition of continuous functions is continuous, and by the fact that the universal copair $[g_1, g_2]$ of continuous functions g_1 and g_2 is continuous. □

Lemma D.16 (Unref. Unroll Sound)**.** *If* $\vdash G[\mu F] \stackrel{\circ}{=} P$ *then* $[\![G]\!]\,(\mu\,[\![F]\!]) = [\![P]\!]$.

Proof. By structural induction on the derivation of $\vdash G[\mu F] \stackrel{\circ}{=} P$. □

Definition D.6 (Directed Poset)**.** A subset $D \subseteq P$ of a poset P is *directed* if every finite subset $E \subseteq D$ of it has an upper bound in D.

Lemma D.17 (Exchange)**.**

If D_1 *and* D_2 *are directed posets,* E *is a predomain,*

and $f : D_1 \times D_2 \to E$ *is a monotone function*

then $\sqcup_{x_1 \in D_1} \sqcup_{x_2 \in D_2} f(x_1, x_2) = \sqcup_{x_2 \in D_2} \sqcup_{x_1 \in D_1} f(x_1, x_2)$

Proof. See Lemma 4.9 (Exchange) in the textbook of Gunter [1993]. □

Lemma D.18 (Diagonal)**.** *If* D *is a directed poset and* E *is a predomain*

and $f : D \times D \to E$ *is a monotone function*

then $\sqcup_{x \in D} \sqcup_{y \in D} f(x,y) = \sqcup_{x \in D} f(x,x)$.

Proof. This is Lemma 4.16 (Diagonal) in the textbook of Gunter [1993]. □

Lemma D.19 (Application Diagonal)**.** *If D and E are predomains and $f_0 \sqsubseteq f_1 \sqsubseteq \cdots$ is a chain in $D \Rightarrow E$ and $x_0 \sqsubseteq x_1 \sqsubseteq \cdots$ is a chain in D then $\sqcup_{k\in\mathbb{N}} \sqcup_{j\in\mathbb{N}} f_k x_j = \sqcup_{k\in\mathbb{N}} f_k x_k$.*

Proof. Define the function $g : \mathbb{N}\times\mathbb{N} \to E$ by $(k,j) \mapsto f_k x_j$. The poset $\mathbb{N}\times\mathbb{N}$ (with componentwise ordering where $\mathbb{N}$ has usual order $\leq$) is directed and the function g is monotone, so the goal follows by Lemma D.18 (Diagonal). □

Lemma D.20 (App. lub Distributes)**.** *If D and E are predomains and $f_0 \sqsubseteq f_1 \sqsubseteq \cdots$ is a chain in $D \Rightarrow E$ and $x_0 \sqsubseteq x_1 \sqsubseteq \cdots$ is a chain in D then $\sqcup_{k\in\mathbb{N}} f_k x_k = (\sqcup_{k\in\mathbb{N}} f_k)(\sqcup_{k\in\mathbb{N}} x_k)$.*

Proof.

$$\begin{aligned}
\sqcup_{k\in\mathbb{N}} f_k x_k &= \sqcup_{k\in\mathbb{N}} \sqcup_{j\in\mathbb{N}} f_k x_j && \text{By Lemma D.19 (Application Diagonal)}\\
&= \sqcup_{k\in\mathbb{N}} f_k (\sqcup_{j\in\mathbb{N}} x_j) && \text{For all } k\text{, we have } f_k \text{ continuous}\\
&= (\sqcup_{k\in\mathbb{N}} f_k)(\sqcup_{k\in\mathbb{N}} x_k) && \text{By def.}
\end{aligned}$$

□

Lemma D.21 (Cut lub)**.** *If D is a predomain and $d_0 \sqsubseteq d_1 \sqsubseteq \cdots$ is a chain in D then for all $m \in \mathbb{N}$, we have $\sqcup_{k\in\mathbb{N}} d_k = \sqcup_{k\geq m} d_k$.*

Proof. Straightforward. □

The following two lemmas (Lemma D.22 (Continuous Maps) and Lemma D.23 (Unrefined Typing Soundness)) are mutually recursive.

Lemma D.22 (Continuous Maps)**.** *Suppose $\vdash \delta_1 : \Gamma_1$ and $\vdash \delta_2 : \Gamma_2$ and $(\Gamma_1, y : Q, \Gamma_2)$* ctx*.*

(1) If $\Gamma_1, y : Q, \Gamma_2 \vdash h \Rightarrow P$

then the function $[\![Q]\!] \to [\![P]\!]$ defined by $d \mapsto [\![h]\!]_{\delta_1, d/y, \delta_2}$ is continuous.

(2) *If* $\Gamma_1, y : Q, \Gamma_2 \vdash g \Rightarrow \uparrow P$

then the function $[\![Q]\!] \to [\![\uparrow P]\!]$ *defined by* $d \mapsto [\![g]\!]_{\delta_1, d/y, \delta_2}$ *is continuous.*

(3) *If* $\Gamma_1, y : Q, \Gamma_2 \vdash v \Leftarrow P$

then the function $[\![Q]\!] \to [\![P]\!]$ *defined by* $d \mapsto [\![v]\!]_{\delta_1, d/y, \delta_2}$ *is continuous.*

(4) *If* $\Gamma_1, y : Q, \Gamma_2 \vdash e \Leftarrow N$

then the function $[\![Q]\!] \to [\![N]\!]$ *defined by* $d \mapsto [\![e]\!]_{\delta_1, d/y, \delta_2}$ *is continuous.*

(5) *If* $\Gamma_1, y : Q, \Gamma_2; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$

then the function $[\![Q]\!] \to [\![N]\!]$

defined by $d \mapsto [\![\{r_i \Rightarrow e_i\}_{i \in I}]\!]_{\delta_1, d/y, \delta_2}$ *is continuous.*

(6) *If* $\Gamma_1, y : Q, \Gamma_2; [N] \vdash s \Rightarrow \uparrow P$

then the function $[\![Q]\!] \to [\![N]\!] \to [\![\uparrow P]\!]$ *defined by* $d \mapsto [\![s]\!]_{\delta_1, d/y, \delta_2}$ *is continuous.*

Proof. By mutual induction on structure of the given typing derivation. Note that we implicitly use Lemma C.7 (Ix. $[\![-]\!]$ Weak. Invariant) when extending semantic substitutions.

We use the mutually recursive Lemma D.23 (Unrefined Typing Soundness) below to obtain the fact that subderivations (program subterms) denote continuous and hence monotone functions (e.g., in the Unref⇒App of part (2)).

(1) Straightforward.

(2) Use Lemma D.20 (App. lub Distributes) in the Unref⇒App case. Otherwise straightforward.

(3) All cases are straightforward.

(4) • **Case**
$$\frac{\Gamma_1, y : Q, \Gamma_2 \vdash g \Rightarrow \uparrow P \qquad \Gamma_1, y : Q, \Gamma_2, x : P \vdash e_0 \Leftarrow N}{\Gamma_1, y : Q, \Gamma_2 \vdash \mathsf{let}\ x = g;\ e_0 \Leftarrow N}\ \text{Unref}\Leftarrow\mathsf{let}$$
Consider cases of whether or not $\sqcup_{k\in\mathbb{N}} [\![g]\!]_{\delta_1, d_k/y, \delta_2} = (2, \bot_\uparrow)$. The former case is easy. In the latter case, there exist a minimal m and $V_m \in [\![P]\!]_{\delta_1, d_m/y, \delta_2}$ such that $[\![g]\!]_{\delta_1, d_m/y, \delta_2} = (1, V_m)$ and, for all $j \geq m$, there are $V_j \in [\![P]\!]_{\delta_1, d_j/y, \delta_2}$ such that $[\![g]\!]_{\delta_1, d_j/y, \delta_2} = (1, V_j)$; use Lemma D.18 (Diagonal) and Lemma D.21 (Cut lub).

• **Case** Unref⇐`match`: Use Lemma D.20 (App. lub Distributes).

• **Case** Unref⇐rec: Use Lemma D.17 (Exchange).

• The remaining cases are straightforward.

(5) Straightforward.

(6) Use Lemma D.20 (App. lub Distributes) in the UnrefSpineApp case. Otherwise straightforward. □

Lemma D.23 (Unrefined Typing Soundness)**.** *Assume* $\vdash \delta : \Gamma$.

(1) If $\Gamma \vdash h \Rightarrow P$ *then* $[\![\Gamma \vdash h \Rightarrow P]\!]_\delta \in [\![P]\!]$.

(2) If $\Gamma \vdash g \Rightarrow \uparrow P$ *then* $[\![\Gamma \vdash g \Rightarrow \uparrow P]\!]_\delta \in [\![\uparrow P]\!]$.

(3) If $\Gamma \vdash v \Leftarrow P$ *then* $[\![\Gamma \vdash v \Leftarrow P]\!]_\delta \in [\![P]\!]$.

(4) If $\Gamma \vdash e \Leftarrow N$ *then* $[\![\Gamma \vdash e \Leftarrow N]\!]_\delta \in [\![N]\!]$.

(5) If $\Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$ *then* $[\![\Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N]\!]_\delta \in [\![P]\!] \Rightarrow [\![N]\!]$.

(6) If $\Gamma; [N] \vdash s \Rightarrow \uparrow P$ *then* $[\![\Gamma; [N] \vdash s \Rightarrow \uparrow P]\!]_\delta \in [\![N]\!] \Rightarrow [\![\uparrow P]\!]$.

Proof. By mutual induction on structure of the given typing derivation. Straightforward. Uses the mutually recursive Lemma D.22 (Continuous Maps) above to obtain the continuity of maps denoted by program subterms (e.g., in the Unref$\Leftarrow\lambda$ case of part (4)). Also uses Lemma D.15 (Unref. Type Denotations).

The Unref$\Leftarrow\mu$ and UnrefMatchμ cases use Lemma D.8 (Mu is Fixed Point) and Lemma D.16 (Unref. Unroll Sound).

The Unref$\Leftarrow$rec case uses Lemma D.11 (Least Fixed Point). □

Lemma D.24 (Unrefined Substitution Typing Soundness)**.**
If $\Gamma_0 \vdash \sigma : \Gamma$ *then* $\vdash [\![\sigma]\!]_\delta : \Gamma$ *for all* $\vdash \delta : \Gamma_0$.

Proof. Similar to Lemma E.28 (Substitution Typing Soundness), but simpler: there's no corresponding SubstIx or SubstProp case, and the UnrefValσ case uses Lemma D.23 (Unrefined Typing Soundness) and has no need for an unrefined version of Lemma E.18 (Type WF Substitution Soundness). □

Lemma D.25 (Unrefined Substitution Soundness)**.** *Assume* $\Gamma_0 \vdash \sigma : \Gamma$ *and* $\vdash \delta : \Gamma_0$.

(1) If $\mathscr{D} :: \Gamma \vdash h \Rightarrow P$ *then* $[\![\Gamma_0 \vdash [\sigma]^{\mathsf{h}} h \Rightarrow P]\!]_\delta = [\![\Gamma \vdash h \Rightarrow P]\!]_{[\![\sigma]\!]_\delta}$.

(2) If $\mathscr{D} :: \Gamma \vdash g \Rightarrow \uparrow P$ *then* $[\![\Gamma_0 \vdash [\sigma]g \Rightarrow \uparrow P]\!]_\delta = [\![\Gamma \vdash g \Rightarrow \uparrow P]\!]_{[\![\sigma]\!]_\delta}$.

(3) If $\mathscr{D} :: \Gamma \vdash v \Leftarrow P$ *then* $[\![\Gamma_0 \vdash [\sigma]v \Leftarrow P]\!]_\delta = [\![\Gamma \vdash v \Leftarrow P]\!]_{[\![\sigma]\!]_\delta}$.

(4) If $\mathscr{D} :: \Gamma \vdash e \Leftarrow N$ *then* $[\![\Gamma_0 \vdash [\sigma]e \Leftarrow N]\!]_\delta = [\![\Gamma \vdash e \Leftarrow N]\!]_{[\![\sigma]\!]_\delta}$.

(5) If $\mathscr{D} :: \Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$
then $[\![\Gamma_0; [P] \vdash [\sigma]\{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N]\!]_\delta = [\![\Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N]\!]_{[\![\sigma]\!]_\delta}$.

(6) If $\mathscr{D} :: \Gamma; [N] \vdash s \Rightarrow \uparrow P$ *then* $[\![\Gamma_0; [N] \vdash [\sigma]s \Rightarrow \uparrow P]\!]_\delta = [\![\Gamma; [N] \vdash s \Rightarrow \uparrow P]\!]_{[\![\sigma]\!]_\delta}$.

Proof. Note that by Lemma D.24 (Unrefined Substitution Typing Soundness), $\vdash [\![\sigma]\!]_\delta : \Gamma$.

For each part in the lemma statement, we are universally quantifying over "substituted" derivations; at the beginning of each part we assume such a derivation is given.

Proceed by mutual induction on the structure of the program term. For each part, we consider cases for the rule concluding $\mathscr{D}$. While the denotations of program terms are only defined when those terms are well-typed (i.e., have a typing derivation), the denotations themselves are defined merely on the syntax of the program terms. Keeping this in mind, whenever the i.h. is used, the necessary "substituted" subderivations are obtained by inversion on the given "substituted" derivation (the unrefined program typing rules are syntax directed).

(1) • **Case**

$$\frac{(x : P) \in \Gamma}{\Gamma \vdash x \Rightarrow P}\ \text{Unref}\Rightarrow\text{Var}$$

$(x : P) \in \Gamma$ Premise

$\Gamma_0 \vdash \sigma : \Gamma$ Given

$\Gamma_0 \vdash v \Leftarrow P$ By inversion

$\sigma = \sigma_1, v : P/x, \sigma_2$ ″

$\Gamma_0 \vdash \sigma_1 : \Gamma_1$ ″

– **Case** $v = x$

$$
\begin{aligned}
&[\![\Gamma \vdash x \Rightarrow P]\!]_{[\![\sigma]\!]_\delta} \\
&\quad = [\![\sigma]\!]_\delta(x) && \text{By def. of denotation} \\
&\quad = [\![\Gamma_0 \vdash v \Leftarrow P]\!]_\delta && \text{By def. of } [\![\sigma]\!]_\delta \\
&\quad = [\![\Gamma_0 \vdash x \Leftarrow P]\!]_\delta && v = x \\
&\quad = \delta(x) && \text{By def. of } [\![-]\!] \\
&\quad = [\![\Gamma_0 \vdash x \Rightarrow P]\!]_\delta && \text{By def. of } [\![-]\!] \\
&\quad = \left[\!\!\left[\Gamma_0 \vdash [\sigma]^{\mathsf{h}} x \Rightarrow P\right]\!\!\right]_\delta && \text{By def. of } [-]^{\mathsf{h}}-
\end{aligned}
$$

– **Case** $v \neq x$

$$
\begin{aligned}
&[\![\Gamma \vdash x \Rightarrow P]\!]_{[\![\sigma]\!]_\delta} \\
&\quad = [\![\sigma]\!]_\delta(x) && \text{By def. of denotation} \\
&\quad = [\![\Gamma_0 \vdash v \Leftarrow P]\!]_\delta && \text{By def. of } [\![\sigma]\!]_\delta \\
&\quad = [\![\Gamma_0 \vdash (v : P) \Leftarrow P]\!]_\delta && \text{By def. of } [\![-]\!] \\
&\quad = \left[\!\!\left[\Gamma_0 \vdash [\sigma]^{\mathsf{h}} x \Rightarrow P\right]\!\!\right]_\delta && \text{By def. of } [-]^{\mathsf{h}}-
\end{aligned}
$$

- **Case**

$$\frac{\Gamma \vdash v \Leftarrow P}{\Gamma \vdash (v : P) \Rightarrow P}\ \text{Unref}\Rightarrow\text{ValAnnot}$$

$$
\begin{aligned}
&[\![\Gamma \vdash (v : P) \Rightarrow P]\!]_{[\![\sigma]\!]_\delta} \\
&\quad = [\![\Gamma \vdash v \Leftarrow P]\!]_{[\![\sigma]\!]_\delta} && \text{By def. of } [\![-]\!] \\
&\quad = [\![\Gamma_0 \vdash [\sigma]v \Leftarrow P]\!]_\delta && \text{By i.h.} \\
&\quad = [\![\Gamma_0 \vdash ([\sigma]v : P) \Rightarrow P]\!]_\delta && \text{By def. of } [\![-]\!] \\
&\quad = \left[\!\!\left[\Gamma_0 \vdash [\sigma]^{\mathsf{h}}(v : P) \Rightarrow P\right]\!\!\right]_\delta && \text{By def. of } [-]^{\mathsf{h}}-
\end{aligned}
$$

(2) • **Case**

$$\frac{\Gamma \vdash h \Rightarrow \downarrow N \qquad \Gamma; [N] \vdash s \Rightarrow \uparrow P}{\Gamma \vdash h(s) \Rightarrow \uparrow P}\ \text{Unref}\Rightarrow\text{App}$$

Straightforward: Suppose $\Gamma_0 \vdash [\sigma](h(s)) \Rightarrow \uparrow P$. Then

$$
\begin{aligned}
[\![\Gamma \vdash h(s) \Rightarrow \uparrow P]\!]_{[\![\sigma]\!]_\delta} &= [\![h(s)]\!]_{[\![\sigma]\!]_\delta} && \text{By def.} \\
&= [\![s]\!]_{[\![\sigma]\!]_\delta}\, [\![h]\!]_{[\![\sigma]\!]_\delta} && \text{By def.} \\
&= [\![[\sigma]s]\!]_\delta\, [\![[\sigma]h]\!]_\delta && \text{By i.h.} \\
&= [\![([\sigma]h)([\sigma]s)]\!]_\delta && \text{By def.} \\
&= [\![[\sigma](h(s))]\!]_\delta && \text{By def.} \\
&= [\![\Gamma_0 \vdash [\sigma](h(s)) \Rightarrow \uparrow P]\!]_\delta && \text{By def.}
\end{aligned}
$$

• **Case** Unref⇒ExpAnnot: Straightforward.

(3) • **Case**

$$\frac{(x : P) \in \Gamma}{\Gamma \vdash x \Leftarrow P}\ \text{Unref}\Leftarrow\text{Var}$$

$$
\begin{array}{rl}
(x : P) \in \Gamma & \text{Premise} \\
\Gamma_0 \vdash \sigma : \Gamma & \text{Given} \\
\Gamma_0 \vdash v \Leftarrow P & \text{By inversion} \\
\sigma = \sigma_1, v : P/x, \sigma_2 & \text{''}
\end{array}
$$

$$
\begin{array}{lll}
[\![\Gamma \vdash x \Leftarrow P]\!]_{[\![\sigma]\!]_\delta} = [\![\sigma]\!]_\delta\,(x) & & \text{By def. of } [\![-]\!] \\
\qquad = [\![\Gamma_0 \vdash v \Leftarrow P]\!]_\delta & & \text{By def. of } [\![\sigma]\!]_\delta \\
\qquad = [\![\Gamma_0 \vdash [\sigma]x \Leftarrow P]\!]_\delta & & \text{By def. of } [-]-
\end{array}
$$

(Note that x in the last line is a *value* variable.)

- **Case** Unref$\Leftarrow$1: Straightforward.
- **Case** Unref$\Leftarrow\times$: Straightforward.
- **Case** Unref$\Leftarrow+_k$: Straightforward.
- **Case**

$$\frac{\vdash F[\mu F] \stackrel{\circ}{=} P \qquad \Gamma \vdash v_0 \Leftarrow P}{\Gamma \vdash \mathtt{into}(v_0) \Leftarrow \mu F}\ \text{Unref}\Leftarrow\mu$$

Straightforward:

$$\begin{aligned}
[\![\mathtt{into}(v_0)]\!]_{[\![\sigma]\!]_\delta} &= [\![v_0]\!]_{[\![\sigma]\!]_\delta} && \text{By def.}\\
&= [\![[\sigma]v_0]\!]_\delta && \text{By i.h.}\\
&= [\![\mathtt{into}([\sigma]v_0)]\!]_\delta && \text{By def.}\\
&= [\![[\sigma]\mathtt{into}(v_0)]\!]_\delta && \text{By def.}
\end{aligned}$$

- **Case** Unref$\Leftarrow\downarrow$: Straightforward.

(4)

- **Case** Unref$\Leftarrow\uparrow$: Straightforward.

- **Case**

$$\frac{\Gamma \vdash g \Rightarrow \uparrow P \qquad \Gamma, x : P \vdash e_0 \Leftarrow N}{\Gamma \vdash \mathsf{let}\ x = g;\ e_0 \Leftarrow N}\ \text{Unref}\Leftarrow\mathsf{let}$$

$\Gamma_0 \vdash \sigma : \Gamma$ Given

$\Gamma \vdash g \Rightarrow \uparrow P$ Subderivation

$\vdash \delta : \Gamma$ Given

But

$$\begin{aligned}
[\![g]\!]_{[\![\sigma]\!]_\delta} &= [\![[\sigma]g]\!]_\delta && \text{By i.h.}\\
&\in [\![\uparrow P]\!] && \text{By Lemma D.23 (Unrefined Typing Soundness)}\\
&= [\![P]\!] \uplus \{\bot_\uparrow\} && \text{By def.}
\end{aligned}$$

– **Case** $[\![g]\!]_{[\![\sigma]\!]_\delta} = (2, \bot_\uparrow)$: By def. of $[\![-]\!]$ and by $[\![g]\!]_{[\![\sigma]\!]_\delta} = [\![[\sigma]g]\!]_\delta$,

$$[\![\mathsf{let}\ x = g;\ e_0]\!]_{[\![\sigma]\!]_\delta} = \bot_{[\![N]\!]} = [\![[\sigma](\mathsf{let}\ x = g;\ e_0)]\!]_\delta$$

– **Case** $[\![g]\!]_{[\![\sigma]\!]_\delta} = (1, V)$:

$\vdash \delta, [\![[\sigma]g]\!]_\delta / x : \Gamma_0, x : P$ By above and UnrefValδ

$\vdash [\![\sigma]\!]_\delta, [\![g]\!]_{[\![\sigma]\!]_\delta} / x : \Gamma, x : P$ By above

Further,

$\Gamma_0, x : P \vdash \sigma, x : P / x : \Gamma, x : P$ By (unrefined version of) Lemma C.66

Therefore,

$$\begin{aligned}
[\![[\sigma](\mathsf{let}\ x = g;\ e_0)]\!]_\delta &= [\![\mathsf{let}\ x = [\sigma]g;\ [\sigma]e_0]\!]_\delta && \text{By def.} \\
&= [\![[\sigma]e_0]\!]_{\delta, [\![[\sigma]g]\!]_\delta / x} && \text{By def.} \\
&= [\![[\sigma]e_0]\!]_{\delta, [\![g]\!]_{[\![\sigma]\!]_\delta} / x} && \text{By i.h. (as seen above)} \\
&= [\![[\sigma, x : P/x]e_0]\!]_{\delta, [\![g]\!]_{[\![\sigma]\!]_\delta} / x} && \text{Lemma C.81 (unrefined version)} \\
&= [\![e_0]\!]_{[\![\sigma, x:P/x]\!]_{\delta, [\![g]\!]_{[\![\sigma]\!]_\delta} / x}} && \text{By i.h.} \\
&= [\![e_0]\!]_{\left([\![\sigma]\!]_{\delta, [\![g]\!]_{[\![\sigma]\!]_\delta} / x}\right), [\![g]\!]_{[\![\sigma]\!]_\delta} / x} && \text{By def.} \\
&= [\![e_0]\!]_{[\![\sigma]\!]_\delta, [\![g]\!]_{[\![\sigma]\!]_\delta} / x} && \text{Because } x \notin dom(\Gamma_0) \\
&= [\![\mathsf{let}\ x = g;\ e_0]\!]_{[\![\sigma]\!]_\delta} && \text{By def.}
\end{aligned}$$

- **Case** Unref⇐`match`: Straightforward.
- **Case** Unref⇐Diverge: Follows from Lemma D.15 (Unref. Type Denotations)

and definition of denotation.

- **Case** Unref$\Leftarrow\lambda$: Similar to case for Unref$\Leftarrow$let, but simpler.

- **Case**
$$\frac{\Gamma, x : \downarrow N \vdash e_0 \Leftarrow N}{\Gamma \vdash \mathtt{rec}\ x.\ e_0 \Leftarrow N}\ \text{Unref}\Leftarrow\text{rec}$$
By definition, $[\![\mathtt{rec}\ x.\ [\sigma]e_0]\!]_\delta = \sqcup_{k\in\mathbb{N}} d_k$ where $d_0 = \bot_{[\![N]\!]}$ and $d_{n+1} = [\![[\sigma]e_0]\!]_{\delta,d_n/x}$. By definition, $[\![\mathtt{rec}\ x.\ e_0]\!]_{[\![\sigma]\!]_\delta} = \sqcup_{k\in\mathbb{N}} d'_k$ where $d'_0 = \bot_{[\![N]\!]}$ and $d'_{n+1} = [\![e_0]\!]_{[\![\sigma]\!]_\delta,d'_n/x}$. Therefore, it suffices to show $d_k = d'_k$ for all $k \in \mathbb{N}$, which we will do by induction on k. Clearly, $d_0 = \bot = d'_0$. Suppose $d_n = d'_n$. It now suffices to show $d_{n+1} = d'_{n+1}$.

$\Gamma_0, x : \downarrow N \vdash \sigma, x : \downarrow N/x : \Gamma, x : \downarrow N$ — By Lemma C.66 (unrefined version)

$$\begin{aligned}
d_{n+1} &= [\![[\sigma]e_0]\!]_{\delta,d_n/x} && \text{By def.}\\
&= [\![[\sigma, x : \downarrow N/x]e_0]\!]_{\delta,d_n/x} && \text{Identity substitution}\\
&= [\![e_0]\!]_{[\![\sigma,x:\downarrow N/x]\!]_{\delta,d_n/x}} && \text{By i.h.}\\
&= [\![e_0]\!]_{[\![\sigma]\!]_\delta,d_n/x} && \text{By def.}\\
&= [\![e_0]\!]_{[\![\sigma]\!]_\delta,d'_n/x} && \text{Above supposition}\\
&= d'_{n+1} && \text{By def.}
\end{aligned}$$

(5)

- **Case** UnrefMatch1: Straightforward.

- **Case** UnrefMatch$\times$: Similar to case for Unref$\Leftarrow\lambda$.

- **Case** UnrefMatch$+$: Similar to case for Unref$\Leftarrow\lambda$.

- **Case** UnrefMatch0: Both sides are the empty function.

- **Case**

$$\frac{\vdash F[\mu F] \mathrel{\mathring{=}} P \qquad \Gamma, x : P \vdash e \Leftarrow N}{\Gamma; [\mu F] \vdash \{\mathtt{into}(x) \Rightarrow e\} \Leftarrow N}\ \text{UnrefMatch}\mu$$

$$\begin{aligned}
V \in \mu\,[\![F]\!] &\quad \text{Suppose}\\
\mu\,[\![F]\!] = [\![F]\!]\,(\mu\,[\![F]\!]) &\quad \text{By Lemma D.8 (Mu is Fixed Point)}\\
[\![F]\!]\,(\mu\,[\![F]\!]) = [\![P]\!] &\quad \text{By Lemma D.16 (Unref. Unroll Sound)}\\
V \in [\![P]\!] &\quad \text{Follows from above}\\
\vdash \delta : \Gamma_0 &\quad \text{Given}\\
\vdash \delta, V/x : \Gamma_0, x : P &\quad \text{By UnrefVal}\delta\\
\vdash [\![\sigma]\!]_\delta : \Gamma &\quad \text{Above}\\
\vdash [\![\sigma]\!]_\delta, V/x : \Gamma, x : P &\quad \text{By UnrefVal}\delta\\
\Gamma_0 \vdash \sigma : \Gamma &\quad \text{Given}\\
\Gamma_0, x : P \vdash \sigma, x : P/x : \Gamma, x : P &\quad \text{By (unrefined version of) Lemma C.66}
\end{aligned}$$

Therefore,

$$\begin{aligned}
&[\![\Gamma_0; [\mu F] \vdash [\sigma]\,\{\mathtt{into}(x) \Rightarrow e\} \Leftarrow N]\!]_\delta\, V\\
&\quad = [\![\Gamma_0; [\mu F] \vdash \{\mathtt{into}(x) \Rightarrow [\sigma]e\} \Leftarrow N]\!]_\delta\, V && \text{By def.}\\
&\quad = [\![\Gamma_0, x : P \vdash [\sigma]e \Leftarrow N]\!]_{\delta, V/x} && \text{By def.}\\
&\quad = [\![\Gamma_0, x : P \vdash [\sigma, x : P/x]e \Leftarrow N]\!]_{\delta, V/x} && \text{Identity substitution}\\
&\quad = [\![\Gamma, x : P \vdash e \Leftarrow N]\!]_{[\![\sigma, x:P/x]\!]_{\delta, V/x}} && \text{By i.h.}\\
&\quad = [\![\Gamma, x : P \vdash e \Leftarrow N]\!]_{[\![\sigma]\!]_\delta, V/x} && \text{By def.}\\
&\quad = [\![\Gamma; [\mu F] \vdash \{\mathtt{into}(x) \Rightarrow e\} \Leftarrow N]\!]_{[\![\sigma]\!]_\delta}\, V && \text{By def.}
\end{aligned}$$

- **Case** Unref⇐Diverge: Straightforward.

(6)

- **Case** UnrefSpineApp: Straightforward.

- **Case**

$$\frac{}{\Gamma;[\uparrow P] \vdash \cdot \Rightarrow \uparrow P}\ \text{UnrefSpineNil}$$

Both sides are $id_{[\![\uparrow P]\!]}$. □

Lemma D.26 (Refinement Subset of Erasure)**.** *Assume* $\vdash \delta : \Xi$.

(1) If $\Xi \vdash A\ \mathsf{type}[\xi]$ *then* $[\![A]\!]_\delta \subseteq [\![|A|]\!]$.

(2) If $\Xi \vdash \mathscr{F}\ \mathsf{functor}[\xi]$ *and* $X \in \mathbf{Set}$ *then* $[\![\mathscr{F}]\!]_\delta X \subseteq [\![|\mathscr{F}|]\!] X$.

Proof. By mutual induction on the structure of the given type or functor well-formedness derivation. □

Lemma D.27 (Refined and Unrefined fmap Agree)**.** *If* $\Xi \vdash \mathscr{F}\ \mathsf{functor}[\xi]$ *and* f *is a function from set* X *to set* Y *and* $\vdash \delta : \Xi$ *then* $[\![\mathscr{F}]\!]_\delta f = [\![|\mathscr{F}|]\!] f$ *on* $[\![\mathscr{F}]\!]_\delta X$.

Proof. By structural induction on the derivation of $\Xi \vdash \mathscr{F}\ \mathsf{functor}[\xi]$. □

Lemma D.28 (Erasure Subst. Invariant)**.** *Assume* $\Xi_0 \vdash \sigma : \Xi$.

(1) If $\Xi \vdash A\ \mathsf{type}[\xi]$ *then* $|[\sigma]A| = |A|$.

(2) If $\Xi \vdash \mathscr{F}\ \mathsf{functor}[\xi]$ *then* $|[\sigma]\mathscr{F}| = |\mathscr{F}|$.

Proof. By mutual induction on the structure of A and $\mathscr{F}$. □

Lemma D.29 (Subtyping Erases to Equality)**.**

(1) If $\Theta \vdash A \leq^{\pm} B$ *then* $|A| = |B|$.

(2) If $\Theta \vdash \alpha; F \leq_\tau \beta; G$ *then* $|F| = |G|$.

(3) If $\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)$ *then* $|F'| = |F|$.

Proof. By mutual induction on the structure of the given derivation, using the definition of erasure. The $\leq^{+}\exists$R, $\leq^{-}\forall$L, and Meas$\leq\exists$R cases use Lemma D.28 (Erasure Subst. Invariant). □

Lemma D.30 (Unrolling Erasure)**.**
If $\Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Theta; R$ *then* $\vdash |G|[\mu|F|] \doteq |R|$.

Proof. By structural induction on the derivation of the given refined unrolling judgement, using the definition of erasure. □

Lemma D.31 (Erasure of Typing)**.**

(1) If $\Theta; \Gamma \vdash h \Rightarrow P$ *then* $|\Gamma| \vdash |h| \Rightarrow |P|$.

(2) If $\Theta; \Gamma \vdash g \Rightarrow \uparrow P$ *then* $|\Gamma| \vdash |g| \Rightarrow |\uparrow P|$.

(3) If $\Theta; \Gamma \vdash v \Leftarrow P$ *then* $|\Gamma| \vdash |v| \Leftarrow |P|$.

(4) If $\Theta; \Gamma \vdash e \Leftarrow N$ *then* $|\Gamma| \vdash |e| \Leftarrow |N|$.

(5) If $\Theta; \Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$ *then* $|\Gamma|; [|P|] \vdash |\{r_i \Rightarrow e_i\}_{i \in I}| \Leftarrow |N|$.

(6) If $\Theta; \Gamma; [N] \vdash s \Rightarrow \uparrow P$ *then* $|\Gamma|; [|N|] \vdash |s| \Rightarrow |\uparrow P|$.

Proof. By mutual structural induction on the given refined typing derivation, considering cases for its concluding rule. The Decl$\Leftarrow$Var case of part (3) and the Decl$\Leftarrow$`rec` case of part (4) use Lemma D.29 (Subtyping Erases to Equality). The Decl$\Leftarrow$`rec` case of part (4) and the Decl$\Leftarrow\exists$ (DeclSpine$\forall$) case of part (3) (part (6)) uses Lemma D.28 (Erasure

Subst. Invariant). The Decl$\Leftarrow\mu$ (DeclMatchμ) case of part (3) (part (5)) uses Lemma D.30 (Unrolling Erasure). □

Lemma D.32 (Erasure and Substitution Commute)**.** *Assume* $\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma$.

(1) If $\Theta;\Gamma \vdash h \Rightarrow P$ *then* $|[\sigma]^{\mathsf{h}} h| = [|\sigma|]^{\mathsf{h}} |h|$.

(2) If $\Theta;\Gamma \vdash g \Rightarrow \uparrow P$ *then* $|[\sigma]g| = [|\sigma|]|g|$.

(3) If $\Theta;\Gamma \vdash v \Leftarrow P$ *then* $|[\sigma]v| = [|\sigma|]|v|$.

(4) If $\Theta;\Gamma \vdash e \Leftarrow N$ *then* $|[\sigma]e| = [|\sigma|]|e|$.

(5) If $\Theta;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$ *then* $|[\sigma]\{r_i \Rightarrow e_i\}_{i\in I}| = [|\sigma|]|\{r_i \Rightarrow e_i\}_{i\in I}|$.

(6) If $\Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P$ *then* $|[\sigma]s| = [|\sigma|]|s|$.

Proof. By structural induction on the given typing derivation. The proof is straightforward: apply the i.h. as needed, follow the definitions of substitution and erasure, and use Lemma D.28 (Erasure Subst. Invariant), Lemma C.43 (Index Id. Subs. Extension), and Lemma C.66 (Id. Subst. Extension) as needed. We only show the first part.

- **Case**
$$\frac{(x : R) \in \Gamma}{\Theta;\Gamma \vdash x \Rightarrow R}\ \text{Decl}\Rightarrow\text{Var}$$

$\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma$ Given

$(x : R) \in \Gamma$ Premise

$\Theta = \Theta_1, \Theta_2$ By inversion

$\Gamma = \Gamma_1, \Gamma_2$ $''$

$\sigma = \sigma_1, v : \lfloor \lfloor \sigma_1 \rfloor \rfloor R/x, \sigma_2$ $''$

$\Theta_0;\Gamma_0 \vdash \sigma_1 : \Theta_1;\Gamma_1$ $''$

$|\sigma| = |\sigma_1|, |v| : |\lfloor \lfloor \sigma_1 \rfloor \rfloor R|/x, |\sigma_2|$ By def. of erasure

– **Case** $v = x$

$$\begin{aligned} |\sigma| &= |\sigma_1|, |v| : |\lfloor \lfloor \sigma_1 \rfloor \rfloor R|/x, |\sigma_2| && \text{Above} \\ &= |\sigma_1|, |x| : |\lfloor \lfloor \sigma_1 \rfloor \rfloor R|/x, |\sigma_2| && \text{By equality} \\ &= |\sigma_1|, x : |\lfloor \lfloor \sigma_1 \rfloor \rfloor R|/x, |\sigma_2| && \text{By def. of erasure} \end{aligned}$$

$$\begin{aligned} |[\sigma]^{\mathsf{h}} x| &= x && \text{By def. of } [-]^{\mathsf{h}} - \\ &= [|\sigma|]^{\mathsf{h}} x && \text{By def. of } [-]^{\mathsf{h}} - \\ &= [|\sigma|]^{\mathsf{h}} |x| && \text{By def. of erasure} \end{aligned}$$

– **Case** $v \neq x$

$$
\begin{array}{lll}
|[\sigma]^{\mathsf{h}} x| = |(v : [\lfloor \sigma_1 \rfloor]R)| & & \text{By def. of substitution (at heads)} \\
= (|v| : |[\lfloor \sigma_1 \rfloor]R|) & & \text{By definition of } [-]- \\
= [|\sigma|]^{\mathsf{h}} x & & \text{By def. of } [-]^{\mathsf{h}}- \\
= [|\sigma|]^{\mathsf{h}} |x| & & \text{By def. of erasure}
\end{array}
$$

- **Case**
$$\frac{\overline{\Theta} \vdash P\ \mathsf{type}[\xi] \qquad \Theta;\Gamma \vdash v \Leftarrow P}{\Theta;\Gamma \vdash (v : P) \Rightarrow P}\ \text{Decl}\Rightarrow\text{ValAnnot}$$

$$
\begin{array}{lll}
|[\sigma]^{\mathsf{h}}(v : P)| = |([\sigma]v : [\sigma]P)| & & \text{By def. of substitution (at heads)} \\
= (|[\sigma]v| : |[\sigma]P|) & & \text{By def. of erasure} \\
= ([|\sigma|]|v| : |[\sigma]P|) & & \text{By i.h.} \\
= ([|\sigma|]|v| : |P|) & & \text{By Lemma D.28 (Erasure Subst. Invariant)} \\
= ([|\sigma|]|v| : [|\sigma|]|P|) & & \text{No variables in } |P| \\
= [|\sigma|]^{\mathsf{h}}(|v| : |P|) & & \text{By def. of substitution (at heads)} \\
= [|\sigma|]^{\mathsf{h}}|(v : P)| & & \text{By def. of erasure}\square
\end{array}
$$

Lemma D.33 (Erasure of Substitution Typing)**.**

(1) If $\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma$ *then* $|\Gamma_0| \vdash |\sigma| : |\Gamma|$.

(2) If $\vdash \delta : \Theta;\Gamma$ *then* $\vdash |\delta| : |\Gamma|$.

Proof.

(1) By structural induction on the derivation of $\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma$, considering cases for its concluding rule, and using the definition of erasure. The SubstVal case uses Lemma D.31 (Erasure of Typing) and Lemma D.28 (Erasure Subst. Invariant) and Lemma D.32 (Erasure and Substitution Commute).

(2) By structural induction on the derivation of $\vdash \delta : \Theta;\Gamma$, considering cases for its concluding rule, and using the definition of erasure. The Valδ case uses Lemma D.26 (Refinement Subset of Erasure). □

Lemma D.34 ($|-|$ and $[\![-]\!]$ Commute)**.**
If $\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma$ *and* $\vdash \delta : \Theta_0;\Gamma_0$ *then* $|[\![\sigma]\!]_\delta| = [\![|\sigma|]\!]_{|\delta|}$.

Proof. By structural induction on the derivation of $\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma$, considering cases for its concluding rule. □

Appendix E

Semantic Metatheory of Declarative (Refined) System

Definition E.1 (Fold)**.** In Figure A.49, we informally define *fold*. Given $\Theta \vdash F$ functor$[_]$ and $\Theta \vdash \alpha : F(\tau) \Rightarrow \tau$ and $\vdash \delta : \Theta$, we define the function $fold_{[\![F]\!]_\delta}\ [\![\alpha]\!]_\delta : \mu\,[\![F]\!]_\delta \to [\![\tau]\!]$ as follows. If $\mu\,[\![F]\!]_\delta$ is empty, then define it to be the empty function. If $\mu\,[\![F]\!]_\delta$ is not empty, then for each $V \in \mu\,[\![F]\!]_\delta$, choose $n > 0$ such that $V \in [\![F]\!]^n_\delta\,\emptyset$ (such an n exists because $\mu\,[\![F]\!]_\delta = \cup_{k\in\mathbb{N}}\,[\![F]\!]^k_\delta\,\emptyset$ by definition) and define $(fold_{[\![F]\!]_\delta}\ [\![\alpha]\!]_\delta)\ V$ by $(fold^n_{[\![F]\!]_\delta}\ [\![\alpha]\!]_\delta)\ V$ where $fold^0_{[\![F]\!]_\delta}\ [\![\alpha]\!]_\delta : \emptyset \to [\![\tau]\!]$ is the empty function and $fold^{k+1}_{[\![F]\!]_\delta}\ [\![\alpha]\!]_\delta : [\![F]\!]^{k+1}_\delta\,\emptyset \to [\![\tau]\!]$ is defined by $fold^{k+1}_{[\![F]\!]_\delta}\ [\![\alpha]\!]_\delta = [\![\alpha]\!]_\delta \circ ([\![F]\!]_\delta\,(fold^k_{[\![F]\!]_\delta}\ [\![\alpha]\!]_\delta))$. The function *fold* is well-defined because the choice of n does not matter.

In Lemma E.6 (Fold Membership), we prove membership $fold_{[\![F]\!]_\delta}\ [\![\alpha]\!]_\delta : \mu\,[\![F]\!]_\delta \to [\![\tau]\!]$ of Def. E.1 (the definition's other memberships are proved in order to prove Lemma E.6). In Lemma E.9 (Semantic Fold), we prove the equality that informally defines fold in Figure A.49.

For each sort τ, we give it the discrete order $d \sqsubseteq d$ for all $d \in [\![\tau]\!]$.

Lemma E.1 (Index Cpo)**.** *For any sort τ, the pair $([\![\tau]\!], \sqsubseteq_{[\![\tau]\!]})$, where $\sqsubseteq_{[\![\tau]\!]}$ is the discrete order $d \sqsubseteq_{[\![\tau]\!]} d$ for all $d \in [\![\tau]\!]$, is a predomain.*

Proof. By structural induction on τ. Straightforward. □

Lemma E.2 (Equal Functor Mu)**.** *Assume F and G are functors from* **Set** *to* **Set** *such that $F = G$. Then $\mu F = \mu G$.*

Proof. Since $F = G$, we know $FX = GX$ for all $X \in \mathbf{Set}$. If $k = 0$, then $F^k\emptyset = \emptyset = G^k\emptyset$. Assume $F^k\emptyset = G^k\emptyset$ for fixed $k \geq 0$; then $F^{k+1}\emptyset = F(F^k\emptyset) = G(G^k\emptyset) = G^{k+1}\emptyset$. Therefore, $\mu F = \cup_{k\in\mathbb{N}} F^k\emptyset = \cup_{k\in\mathbb{N}} G^k\emptyset = \mu G$. □

Lemma E.3 (Equal Func. Alg. Mu)**.** *Assume F and G are endofunctors on* **Set***. Let $X \in \mathbf{Set}$. Assume $\alpha : FX \to X$ and $\beta : GX \to X$ are algebras.*
If $F = G$ and $\alpha = \beta$ then $fold_F\ \alpha = fold_G\ \beta$.

Proof. Note that $fold_F\ \alpha : \mu F \to X$ and $fold_G\ \beta : \mu G \to X$. By Lemma E.2 (Equal Functor Mu), $\mu F = \mu G$, i.e., these functions have equal domain. By definition, $\mu F = \cup_{k\in\mathbb{N}} F^k\emptyset$. Denote by f_k and g_k the kth level functions $fold_F\ \alpha$ and $fold_G\ \beta$ from $F^k\emptyset$ to X, respectively. It suffices to prove that $f_k = g_k$ for all $k \in \mathbb{N}$. If $k = 0$, then both f_k and g_k equal the unique empty function. If $k = n+1$ then:

$$
\begin{aligned}
f_{n+1} &= \alpha \circ F f_n && \text{By def.} \\
&= \beta \circ G f_n && \text{Because } F = G \text{ and } \alpha = \beta \\
&= \beta \circ G g_n && \text{By i.h.} \\
&= g_{n+1} && \text{By def.}
\end{aligned}
$$

□

Lemma E.4 (Filter Out Program Vars.)**.** *If $\vdash \delta : \Theta; \Gamma$ then $\vdash \lfloor \delta \rfloor : \Theta$.*

Proof. By structural induction on the given semantic substitution derivation. □

Lemma E.5 ($[\![-]\!]$ Weakening Invariant)**.**

Assume $\vdash \delta : \Xi$ *and* $\vdash \delta' : \Xi'$ *and* $\Xi \subseteq \Xi'$ *and* $\delta'\restriction_{\Xi} = \delta$.

(1) If $\Xi \vdash A\ \mathsf{type}[\xi]$

then $[\![\Xi \vdash A\ \mathsf{type}[\xi]]\!]_{\delta} = [\![\Xi' \vdash A\ \mathsf{type}[\xi]]\!]_{\delta'}$.

(2) If $\Xi \vdash \mathscr{F}\ \mathsf{functor}[\xi]$ *then* $[\![\Xi \vdash \mathscr{F}\ \mathsf{functor}[\xi]]\!]_{\delta} = [\![\Xi' \vdash \mathscr{F}\ \mathsf{functor}[\xi]]\!]_{\delta'}$.

(3) If $\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$ *then* $[\![\Xi \vdash \alpha : F(\tau) \Rightarrow \tau]\!]_{\delta} = [\![\Xi' \vdash \alpha : F(\tau) \Rightarrow \tau]\!]_{\delta'}$.

Proof. By mutual structural induction on the given type, functor or algebra well-formedness derivation. Use Lemma C.7 (Ix. $[\![-]\!]$ Weak. Invariant), Lemma E.2, and Lemma E.3. Use Lemma C.41 (Ix.-Level Weakening). □

Lemma E.6 (Fold Membership)**.** *If* F *is a* **Set** *endofunctor and* τ *is a set and* $\alpha : F(\tau) \to \tau$ *is an algebra then* $fold_F\ \alpha : \mu F \to \tau$.

Proof. Suppose $V \in \mu F$. By Def. E.1, $(fold_F\ \alpha)\ V = (fold_F^n\ \alpha)\ V$ where $n \in \mathbb{N}$ satisfies $V \in F^n\emptyset$. It is straightforward to show, by induction on n, and by using Def. E.1, that $fold_F^n\ \alpha : F^n\emptyset \to \tau$. Therefore, $(fold_F\ \alpha)\ V = (fold_F^n\ \alpha)\ V \in \tau$. □

Lemma E.7 (Type WF Sound)**.** *Assume* $\vdash \delta : \Xi$.

(1) If $\Xi \vdash A\ \mathsf{type}[\xi_A]$ *then* $[\![A]\!]_{\delta} \in \mathbf{rCpo}$.

(2) If $\Xi \vdash \mathscr{F}\ \mathsf{functor}[\xi_{\mathscr{F}}]$ *then* $[\![\mathscr{F}]\!]_{\delta}$ *is a functor from* **rCpo** *to* **rCpo**.

(3) If $\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$ *then* $[\![\alpha]\!]_{\delta} : [\![F]\!]_{\delta}\,[\![\tau]\!] \to [\![\tau]\!]$.

(4) If $\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi_{\mathscr{M}}]$ *then* $[\![\mathscr{M}(F)]\!]_{\delta} \in \{\emptyset, \{\bullet\}\}$.

Proof. By mutual induction on the structure of the given formation derivation.

(1) Use Lemma D.15 (Unref. Type Denotations) and Lemma D.26 (Refinement Subset of Erasure). Also use Lemma E.6 (Fold Membership) and Lemma C.21 (Sorting Soundness) in the DeclTpμ case.

(2) Use Lemma D.15 (Unref. Type Denotations) and Lemma D.26 (Refinement Subset of Erasure) and Lemma D.27 (Refined and Unrefined fmap Agree) and (unstated refined version of) Lemma D.3 (Functor Monotone).

(3) • **Case**

$$\frac{\overset{\mathsf{d}}{\div}\Xi \vdash t : \tau}{\Xi \vdash () \Rightarrow t : I(\tau) \Rightarrow \tau}\ \mathrm{DeclAlg}I$$

Assume $V \in [\![I]\!]_\delta\,[\![\tau]\!]$. The latter equals $\{\bullet\}$ by definition, so $V = \bullet$.

$\overset{\mathsf{d}}{\div}\Xi \vdash t : \tau$	Premise
$\vdash \delta : \Xi$	Given
$\vdash \delta\restriction_{\overset{\mathsf{d}}{\div}\Xi} : \overset{\mathsf{d}}{\div}\Xi$	By property of restriction

Now,

$$\begin{aligned}
[\![() \Rightarrow t]\!]_\delta\, V &= [\![() \Rightarrow t]\!]_\delta\, \bullet \\
&= [\![t]\!]_{\delta\restriction_{\overset{\mathsf{d}}{\div}\Xi}} && \text{By def. of denotation} \\
&\in [\![\tau]\!] && \text{By Lemma C.21 (Sorting Soundness)}
\end{aligned}$$

• **Case**

$$\frac{\alpha \circ \mathsf{inj}_1 \mathrel{\dot{=}} \alpha_1 \qquad \mathscr{D}_1 :: \Xi \vdash \alpha_1 : F_1(\tau) \Rightarrow \tau \qquad \alpha \circ \mathsf{inj}_2 \mathrel{\dot{=}} \alpha_2 \qquad \mathscr{D}_2 :: \Xi \vdash \alpha_2 : F_2(\tau) \Rightarrow \tau}{\Xi \vdash \alpha : (F_1 \oplus F_2)(\tau) \Rightarrow \tau}\ \mathrm{DeclAlg}\oplus$$

Assume $V \in [\![F_1 \oplus F_2]\!]_\delta\,[\![\tau]\!]$. By definition, the latter equals $[\![F_1]\!]_\delta\,[\![\tau]\!] \uplus [\![F_2]\!]_\delta\,[\![\tau]\!]$,

so there exists $j \in \{1,2\}$ such that $V = (j, V_j)$ and $V_j \in [\![F_j]\!]_\delta [\![\tau]\!]$. By the induction hypothesis for subderivation $\mathscr{D}_j$, $[\![\alpha_j]\!]_\delta : [\![F_j]\!]_\delta [\![\tau]\!] \to [\![\tau]\!]$. Therefore, $[\![\alpha]\!]_\delta V = [\![\alpha]\!]_\delta (j, V_j) = [\![\alpha_j]\!]_\delta V_j \in [\![\tau]\!]$.

- **Case**

$$\frac{\mathscr{D}_1 :: \Xi, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id} \vdash q \Rightarrow t : \hat{I}(\tau) \Rightarrow \tau}{\Xi \vdash (a,q) \Rightarrow t : (\mathrm{Id} \otimes \hat{I})(\tau) \Rightarrow \tau}\ \text{DeclAlgId}$$

Assume $V \in [\![\mathrm{Id} \otimes \hat{I}]\!]_\delta [\![\tau]\!]$. By definition, the latter equals $[\![\tau]\!] \times [\![\hat{I}]\!]_\delta [\![\tau]\!]$, so there exist $d \in [\![\tau]\!]$ and $V' \in [\![\hat{I}]\!]_\delta [\![\tau]\!]$ such that $V = (d, V')$. By IxδId, $\vdash \delta, d/a : \Xi, a \overset{\mathsf{d}}{\div} \tau, a\,\mathrm{Id}$ By the i.h. for subderivation $\mathscr{D}_1$, $[\![q \Rightarrow t]\!]_{\delta, d/a} : [\![\hat{I}]\!]_{\delta, d/a} [\![\tau]\!] \to [\![\tau]\!]$. Therefore, $[\![(a,q) \Rightarrow t]\!]_\delta (d, V') = [\![q \Rightarrow t]\!]_{\delta, d/a} V' \in [\![\tau]\!]$.

- **Case**

$$\frac{\mathscr{D}_1 :: \Xi \vdash q \Rightarrow t : \hat{P}(\tau) \Rightarrow \tau \qquad \Xi \vdash Q\ \mathsf{type}[_]}{\Xi \vdash (\top, q) \Rightarrow t : (\underline{Q} \otimes \hat{P})(\tau) \Rightarrow \tau}\ \text{DeclAlgConst}$$

Assume $V \in [\![\underline{Q} \otimes \hat{P}]\!]_\delta [\![\tau]\!]$. By definition, the latter equals $[\![Q]\!]_\delta \times [\![\hat{P}]\!]_\delta [\![\tau]\!]$. Therefore, there exist $V_1 \in [\![Q]\!]_\delta$ and $V_2 \in [\![\hat{P}]\!]_\delta [\![\tau]\!]$ such that $V = (V_1, V_2)$. By the i.h. for subderivation $\mathscr{D}_1$, $[\![q \Rightarrow t]\!]_\delta : [\![\hat{P}]\!]_\delta [\![\tau]\!] \to [\![\tau]\!]$. Therefore, $[\![(\top, q) \Rightarrow t]\!]_\delta V = [\![(\top, q) \Rightarrow t]\!]_\delta (V_1, V_2) = [\![q \Rightarrow t]\!]_\delta V_2 \in [\![\tau]\!]$.

- **Case**

$$\frac{\mathscr{D}_1 :: \Xi, {}^{\mathsf{d}}\Xi' \vdash (\top, q) \Rightarrow t : (\underline{Q} \otimes \hat{P})(\tau) \Rightarrow \tau}{\Xi \vdash (\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q) \Rightarrow t : (\underline{\exists {}^{\mathsf{d}}\Xi' . Q} \otimes \hat{P})(\tau) \Rightarrow \tau}\ \text{DeclAlg}\exists$$

We want to show that

$$\left[\!\!\left[(\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q) \Rightarrow t\right]\!\!\right]_\delta : \left[\!\!\left[\underline{\exists {}^{\mathsf{d}}\Xi' . Q} \otimes \hat{P}\right]\!\!\right]_\delta [\![\tau]\!] \to [\![\tau]\!]$$

Suppose V is an element of the domain. Then, by definition of denotation, there exist $V_1 \in \left\{ V \in [\![|Q|]\!] \;\middle|\; \exists \delta' \in \left[\!\!\left[{}^{\mathsf{d}}\Xi'\right]\!\!\right] . V \in [\![Q]\!]_{\delta, \delta'} \right\}$ and $V_2 \in [\![\hat{P}]\!]_\delta [\![\tau]\!]$ such that

$V = (V_1, V_2)$. Now, by definition,

$$\left[\!\left[(\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q) \Rightarrow t\right]\!\right]_{\delta}(V_1, V_2) = [\![(\top, q) \Rightarrow t]\!]_{\delta,\delta'}(V_1, V_2)$$

where $\delta' \in \left[\!\left[{}^{\mathsf{d}}\Xi'\right]\!\right]$ is fixed such that $V_1 \in [\![Q]\!]_{\delta,\delta'}$ (that V_1 inhabits the set above implies such a δ' exists). By the i.h. for subderivation $\mathscr{D}_1$ (and by definition of denotation), $[\![(\top, q) \Rightarrow t]\!]_{\delta,\delta'} : [\![Q]\!]_{\delta,\delta'} \times \left[\!\left[\hat{P}\right]\!\right]_{\delta,\delta'} [\![\tau]\!] \to [\![\tau]\!]$. Because $dom({}^{\mathsf{d}}\Xi')$ is disjoint from $FV(\hat{P})$, we have $\left[\!\left[\hat{P}\right]\!\right]_{\delta,\delta'} [\![\tau]\!] = \left[\!\left[\hat{P}\right]\!\right]_{\delta} [\![\tau]\!]$. Therefore, $[\![(\top, q) \Rightarrow t]\!]_{\delta,\delta'}(V_1, V_2) \in [\![\tau]\!]$, as desired. □

Lemma E.8. *If* $\Xi \vdash \mathscr{F}$ functor[_] *and* $\Xi \vdash F$ functor[_] *and* $\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$
and $\vdash \delta : \Xi$ *and* $n \in \mathbb{N}$ *and* $V \in [\![\mathscr{F}]\!]_{\delta}([\![F]\!]^n_{\delta} \emptyset)$
then $([\![\mathscr{F}]\!]_{\delta}(fold^n_{[\![F]\!]_{\delta}} [\![\alpha]\!]_{\delta}))\, V = ([\![\mathscr{F}]\!]_{\delta}(fold_{[\![F]\!]_{\delta}} [\![\alpha]\!]_{\delta}))\, V$

Proof. By structural induction on $\mathscr{F}$. □

Lemma E.9 (Semantic Fold)**.**
If $\Xi \vdash F$ functor[_] *and* $\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$ *and* $\vdash \delta : \Xi$
then $fold_{[\![F]\!]_{\delta}} [\![\alpha]\!]_{\delta} = [\![\alpha]\!]_{\delta} \circ ([\![F]\!]_{\delta}(fold_{[\![F]\!]_{\delta}} [\![\alpha]\!]_{\delta}))$.

Proof. By Lemma E.6 (Fold Membership) and Lemma E.7 (Type WF Sound), both sides of the equality are elements of $\mu\, [\![F]\!]_{\delta} \to [\![\tau]\!]$. Suppose $V \in \mu\, [\![F]\!]_{\delta}$. By Def. E.1, there exists $n \in \mathbb{N}$ such that $V \in [\![F]\!]^n_{\delta} \emptyset$ and $(fold_{[\![F]\!]_{\delta}} [\![\alpha]\!]_{\delta})\, V = (fold^n_{[\![F]\!]_{\delta}} [\![\alpha]\!]_{\delta})\, V$. Continuing,

$$\begin{aligned}(fold^n_{[\![F]\!]_{\delta}} [\![\alpha]\!]_{\delta})\, V &= [\![\alpha]\!]_{\delta}((([\![F]\!]_{\delta}(fold^{n-1}_{[\![F]\!]_{\delta}} [\![\alpha]\!]_{\delta}))\, V) && \text{By Def. E.1 } (n > 0 \text{ since } V \in [\![F]\!]^n_{\delta} \emptyset)\\ &= [\![\alpha]\!]_{\delta}((([\![F]\!]_{\delta}(fold_{[\![F]\!]_{\delta}} [\![\alpha]\!]_{\delta}))\, V) && \text{By Lemma E.8}\end{aligned}$$

The goal follows by function extensionality. □

Lemma E.10 (Ref. Equal Functor Mu)**.**

If $\Xi \vdash F$ $\mathsf{functor}[\xi]$ *then* $\mu\,[\![F]\!]_\delta = [\![F]\!]_\delta\,(\mu\,[\![F]\!]_\delta)$ *for all* $\vdash \delta : \Xi$.

Proof. Similar to Lemma D.8 (Mu is Fixed Point). □

Lemma E.11 (Semantic Unroll)**.** *If* $\Xi \vdash \{\nu : \mu F \mid \mathscr{M}(F)\}$ $\mathsf{type}[\xi]$ *and* $\delta \in [\![\Xi]\!]$

then the set

$$\left\{V \in [\![F]\!]_\delta\,(\mu\,[\![F]\!]_\delta) \;\middle|\; [\![\boldsymbol{t}]\!]_\delta\,([\![\alpha]\!]_\delta\,([\![F]\!]_\delta\,(\mathit{fold}_{[\![F]\!]_\delta}\,[\![\alpha]\!]_\delta)\,V)) = [\![t]\!]_\delta \text{ for all } (\!|\alpha|\!)_F\,\nu\,\boldsymbol{t} =_\tau t \in \mathscr{M}(F)\right\}$$

is equal to the set

$$\{V \in \mu\,[\![F]\!]_\delta \mid [\![\mathscr{M}(F)]\!]_\delta\,V = \{\bullet\}\}$$

Proof. This follows from Lemma E.10 (Ref. Equal Functor Mu) and Lemma E.9 (Semantic Fold) and definitions. □

Lemma E.12. *If* $\delta_1 \restriction_{\xi_1} = \delta_2 \restriction_{\xi_1}$ *and* $\delta_1 \restriction_{\xi_2} = \delta_2 \restriction_{\xi_2}$ *then* $\delta_1 \restriction_{\xi_1 \cup \xi_2} = \delta_2 \restriction_{\xi_1 \cup \xi_2}$.

Proof. By structural induction on $\delta_1 \restriction_{\xi_2} = \delta_2 \restriction_{\xi_2}$. Straightforward. □

Lemma E.13. *If* $\delta_1 \restriction_{\xi'} = \delta_2 \restriction_{\xi'}$ *and* $\xi \subseteq \xi'$ *then* $\delta_1 \restriction_{\xi} = \delta_2 \restriction_{\xi}$.

Proof. By structural induction on $\delta_1 \restriction_{\xi'} = \delta_2 \restriction_{\xi'}$. Straightforward. □

Lemma E.14.

If $(\delta_1, \delta) \restriction_\xi = (\delta_2, \delta) \restriction_\xi$ *and* $\vdash \delta_1, \delta : \Xi, {}^{\mathsf{d}}\Xi$ *and* $\vdash \delta_2, \delta : \Xi, {}^{\mathsf{d}}\Xi$ *and* $\vdash \delta : {}^{\mathsf{d}}\Xi$

then $\delta_1 \restriction_{\xi - {}^{\mathsf{d}}\Xi} = \delta_2 \restriction_{\xi - {}^{\mathsf{d}}\Xi}$.

Proof. Suppose $(\mathfrak{D} \rightarrow a) \in (\xi - {}^{\mathsf{d}}\Xi)$. Then there exists $\mathfrak{D}'$ such that $\mathfrak{D}' \rightarrow a \in \xi$ and $a \notin \mathit{dom}({}^{\mathsf{d}}\Xi)$ and $\mathfrak{D}' - {}^{\mathsf{d}}\Xi = \mathfrak{D}$. Assume $\delta_1 \restriction_{\mathfrak{D}} = \delta_2 \restriction_{\mathfrak{D}}$. It suffices to show $\delta_1(a) = \delta_2(a)$. By inversion on the given $(\delta_1, \delta_1') \restriction_\xi = (\delta_2, \delta_2') \restriction_\xi$ we know that if $(\delta_1, \delta_1') \restriction_{\mathfrak{D}'} = (\delta_2, \delta_2') \restriction_{\mathfrak{D}'}$ then

$(\delta_1, \delta_1')(a) = (\delta_2, \delta_2')(a)$ that is $\delta_1(a) = \delta_2(a)$. Therefore, it suffices to show $(\delta_1, \delta_1')\restriction_{\mathfrak{D}'} = (\delta_2, \delta_2')\restriction_{\mathfrak{D}'}$. But this follows from $\delta_1' = \delta_2'$ (given) and $\delta_1\restriction_{\mathfrak{D}'} = \delta_2\restriction_{\mathfrak{D}'}$, which follows from a property of restriction with $\mathfrak{D}' - {}^{\mathsf{d}}\Xi = \mathfrak{D}$ (above) and $\delta_1\restriction_{\mathfrak{D}} = \delta_2\restriction_{\mathfrak{D}}$ (above) and $dom(\delta_1) = dom(\delta_2) = dom(\Xi)$ which is disjoint from $dom({}^{\mathsf{d}}\Xi)$ (by inversion on the presupposed $(\Xi, {}^{\mathsf{d}}\Xi)$ ctx). □

Lemma E.15 (Dependency Agreement Closure)**.**

If $\delta_1\restriction_{\xi} = \delta_2\restriction_{\xi}$ *and* $\delta_1\restriction_{\Xi} = \delta_2\restriction_{\Xi}$ *and* $\xi - \Xi \vdash {}^{\mathsf{d}}\Xi_0$ det

and $\vdash \delta_1 : \Xi, {}^{\mathsf{d}}\Xi$ *and* $\vdash \delta_2 : \Xi, {}^{\mathsf{d}}\Xi$ *and* ${}^{\mathsf{d}}\Xi_0 \subseteq {}^{\mathsf{d}}\Xi$ *and* $FV(\xi) \subseteq dom(\frac{\mathsf{d}\cdot}{\cdot}\Xi, {}^{\mathsf{d}}\Xi)$

then $\delta_1\restriction_{{}^{\mathsf{d}}\Xi_0} = \delta_2\restriction_{{}^{\mathsf{d}}\Xi_0}$.

Proof. By induction on the number of elements in $dom({}^{\mathsf{d}}\Xi_0)$. We implicitly use Lemma B.4 (Equivalence of *cl* and det). Assume $a \in dom({}^{\mathsf{d}}\Xi_0)$. It suffices to show $\delta_1(a) = \delta_2(a)$. There exists $(\mathfrak{D} - \Xi)\rightarrow a \in \xi - \Xi$ such that $\mathfrak{D} - \Xi \subseteq cl(\xi - \Xi)(\emptyset)$ and $a \notin (\mathfrak{D} - \Xi)$. By definition of subtraction, $\mathfrak{D}\rightarrow a \in \xi$. But $\delta_1\restriction_{\xi} = \delta_2\restriction_{\xi}$ so $\delta_1\restriction_{\mathfrak{D}} = \delta_2\restriction_{\mathfrak{D}}$ implies $\delta_1(a) = \delta_2(a)$. But $\mathfrak{D} = (\mathfrak{D} - \Xi) \cup (\mathfrak{D} \cap \Xi)$ and $\delta_1\restriction_{\Xi} = \delta_2\restriction_{\Xi}$ hence $\delta_1\restriction_{\mathfrak{D}\cap\Xi} = \delta_2\restriction_{\mathfrak{D}\cap\Xi}$ so it suffices to show $\delta_1\restriction_{\mathfrak{D}-\Xi} = \delta_2\restriction_{\mathfrak{D}-\Xi}$. But this follows from the i.h. with ${}^{\mathsf{d}}\Xi_0\restriction_{\mathfrak{D}-\Xi}$ as ${}^{\mathsf{d}}\Xi_0$ (the set ${}^{\mathsf{d}}\Xi_0\restriction_{\mathfrak{D}-\Xi}$ loses at least the a relative to $dom({}^{\mathsf{d}}\Xi_0)$). □

Lemma E.16 (Val.-Det. Indexes Sound (Prop.))**.**

If $\vdash \delta_1 : \Xi$ *and* $\delta_2 : \Xi$ *and* $\Xi \vdash \varphi : \tau\ [\xi]$ *and* $[\![\varphi]\!]_{\delta_1} = \{\bullet\} = [\![\varphi]\!]_{\delta_2}$ *then* $\delta_1\restriction_{\xi} = \delta_2\restriction_{\xi}$.

Proof. By structural induction on $\Xi \vdash \varphi : \mathbb{B}\ [\xi]$. Use Lemma E.12 in the Ix$\wedge$ case. Use Lemma E.13 in the IxSub case. In the Ix=, Ix=L, Ix=R, Ix=LR, Ix=$\times$ cases, use the given $[\![\varphi]\!]_{\delta_1} = \{\bullet\} = [\![\varphi]\!]_{\delta_2}$ and the definition of denotation. □

Lemma E.17 (Soundness of Value-Determined Dependencies)**.**

Assume $\vdash \delta_1 : \Xi$ *and* $\delta_2 : \Xi$.

(1) *If* $\Xi \vdash P\ \mathsf{type}[\xi]$ *and* $V \in [\![P]\!]_{\delta_1}$ *and* $V \in [\![P]\!]_{\delta_2}$ *then* $\delta_1 \restriction_\xi = \delta_2 \restriction_\xi$.

(2) *If* $\Xi \vdash \mathscr{F}\ \mathsf{functor}[\xi]$ *and* X_1 *and* X_2 *are sets and* $V \in [\![\mathscr{F}]\!]_{\delta_1} X_1$ *and* $V \in [\![\mathscr{F}]\!]_{\delta_2} X_2$ *then* $\delta_1 \restriction_\xi = \delta_2 \restriction_\xi$.

(3) *If* $\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi]$ *and* $V \in \mu\,[\![F]\!].$ *and* $[\![\mathscr{M}(F)]\!]_{\delta_1} V = \{\bullet\} = [\![\mathscr{M}(F)]\!]_{\delta_2} V$ *then* $\delta_1 \restriction_\xi = \delta_2 \restriction_\xi$.

Proof. By induction on structure of the given well-formedness derivation. Parts (1) and (2) are mutually recursive. We elide reasoning about restrictions and weakening (for example, uses of Lemma C.7 (Ix. $[\![-]\!]$ Weak. Invariant)).

(1)

- **Case**

$$\frac{}{\Xi \vdash 0\ \mathsf{type}[\cdot]}\ \text{DeclTp0}$$

  By a rule, $\delta_1 \restriction_\cdot = \delta_2 \restriction_\cdot$.

- **Cases** DeclTp1, DeclTp+, DeclTp↓: Similar to DeclTp0 case.

- **Case**

$$\frac{\Xi \vdash R_1\ \mathsf{type}[\xi_1] \qquad \Xi \vdash R_2\ \mathsf{type}[\xi_2]}{\Xi \vdash R_1 \times R_2\ \mathsf{type}[\xi_1 \cup \xi_2]}\ \text{DeclTp}\times$$

  Assume $V \in [\![R_1 \times R_2]\!]_{\delta_k}$ for $k = 1,2$. By definition of denotation, $[\![R_1 \times R_2]\!]_{\delta_k} = [\![R_1]\!]_{\delta_k} \times [\![R_2]\!]_{\delta_k}$ for $k = 1,2$, so there exist V_1 and V_2 such that $V_j \in [\![R_j]\!]_{\delta_k}$ for all $j \in \{1,2\}$ and $k \in \{1,2\}$. By the i.h. (twice), we have $\delta_1 \restriction_{\xi_k} = \delta_2 \restriction_{\xi_k}$ for $k = 1,2$. By Lemma E.12, $\delta_1 \restriction_{\xi_1 \cup \xi_2} = \delta_2 \restriction_{\xi_1 \cup \xi_2}$.

- **Case**

$$\frac{\Xi, {}^{\mathsf{d}}\Xi \vdash Q\ \mathsf{type}[\xi_Q] \qquad \xi_Q - {}^{\mathsf{d}\div}\Xi \vdash {}^{\mathsf{d}}\Xi\ \mathsf{det}}{\Xi \vdash \exists^{\mathsf{d}}\Xi.\, Q\ \mathsf{type}[\xi_Q - {}^{\mathsf{d}}\Xi]}\ \text{DeclTp}\exists$$

Assume $V \in \left[\!\left[\exists^{\mathsf{d}} \Xi.\, Q\right]\!\right]_{\delta_k}$ for $k = 1,2$. By definition of denotation, there exist $\delta_1' \in \left[\!\left[{}^{\mathsf{d}}\Xi\right]\!\right]$ and $\delta_2' \in \left[\!\left[{}^{\mathsf{d}}\Xi\right]\!\right]$ such that $V \in [\![Q]\!]_{\delta_k,\delta_k'}$ for $k = 1,2$. By i.h., $(\delta_1,\delta_1')\restriction_{\xi_Q} = (\delta_2,\delta_2')\restriction_{\xi_Q}$. By Lemma E.15 (Dependency Agreement Closure), $\delta_1' = \delta_2'$. By Lemma E.14, $\delta_1\restriction_{\xi_Q - {}^{\mathsf{d}}\Xi} = \delta_2\restriction_{\xi_Q - {}^{\mathsf{d}}\Xi}$.

- **Case**
$$\frac{\Xi \vdash R\ \mathsf{type}[\xi_R] \qquad \Xi \vdash \overrightarrow{\varphi} : \mathbb{B}\ [\xi_{\overrightarrow{\varphi}}]}{\Xi \vdash R \wedge \overrightarrow{\varphi}\ \mathsf{type}[\xi_R \cup \xi_{\overrightarrow{\varphi}}]}\ \text{DeclTp}\wedge$$
Assume $V \in \left[\!\left[R \wedge \overrightarrow{\varphi}\right]\!\right]_{\delta_k}$ for $k = 1,2$. By definition of denotation, we have $V \in [\![R]\!]_{\delta_k}$ for $k = 1,2$. By the i.h., $\delta_1\restriction_{\xi_R} = \delta_2\restriction_{\xi_R}$. By repeated Lemma E.12 and Lemma E.16 (Val.-Det. Indexes Sound (Prop.)), $\delta_1\restriction_{\xi_{\overrightarrow{\varphi}}} = \delta_2\restriction_{\xi_{\overrightarrow{\varphi}}}$. By Lemma E.12, $\delta_1\restriction_{\xi_R \cup \xi_{\overrightarrow{\varphi}}} = \delta_2\restriction_{\xi_R \cup \xi_{\overrightarrow{\varphi}}}$.

- **Case**
$$\frac{\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi]}{\Xi \vdash \{\nu : \mu F \mid \mathscr{M}(F)\}\ \mathsf{type}[\xi]}\ \text{DeclTp}\mu$$
Assume $V \in [\![\{\nu : \mu F \mid \mathscr{M}(F)\}]\!]_{\delta_k}$ for $k = 1,2$.
By inversion, $V \in \mu\, [\![F]\!]_{\cdot}$ and $[\![\mathscr{M}(F)]\!]_{\delta_1} V = \{\bullet\}$ and $[\![\mathscr{M}(F)]\!]_{\delta_2} V = \{\bullet\}$.
The goal follows by part (3).

(2) - **Case**
$$\frac{\Xi \vdash P\ \mathsf{type}[\xi]}{\Xi \vdash \underline{P}\ \mathsf{functor}[\xi]}\ \text{DeclFuncConst}$$
Assume $V \in \left[\!\left[\underline{P}\right]\!\right]_{\delta_k} X_k$ for $k = 1,2$. By definition of denotation, $\left[\!\left[\underline{P}\right]\!\right]_{\delta_k} X_k = \left[\!\left[P\right]\!\right]_{\delta_k}$ for $k = 1,2$. Therefore, $V \in \left[\!\left[P\right]\!\right]_{\delta_k}$ for $k = 1,2$. By the i.h. (part (1)) for subderivation $\Xi \vdash P\ \mathsf{type}[\xi]$, we have $\delta_1\restriction_{\xi} = \delta_2\restriction_{\xi}$.

- **Case**

$$\frac{}{\Xi \vdash \mathrm{Id}\ \mathsf{functor}[\underbrace{\ \cdot\ }_{\Xi}]}\ \mathrm{DeclFuncId}$$

By a rule, $\delta_1 \restriction_{\cdot} = \delta_2 \restriction_{\cdot}$.

- **Case** DeclFunc$I$: Similar to DeclFuncId case.

- **Case**

$$\frac{\Xi \vdash \hat{B}\ \mathsf{functor}[\xi_1] \qquad \Xi \vdash \hat{P}\ \mathsf{functor}[\xi_2]}{\Xi \vdash \hat{B} \otimes \hat{P}\ \mathsf{functor}[\underbrace{\xi_1 \cup \xi_2}_{\xi}]}\ \mathrm{DeclFunc}\otimes$$

Assume $V \in [\![\hat{B} \otimes \hat{P}]\!]_{\delta_k} X_k$ for $k = 1,2$. By definition of denotation, $[\![\hat{B} \otimes \hat{P}]\!]_{\delta_k} X_k = [\![\hat{B}]\!]_{\delta_k} X_k \times [\![\hat{P}]\!]_{\delta_k} X_k$ for $k = 1,2$. By the i.h. for subderivation $\Xi \vdash \hat{B}\ \mathsf{functor}[\xi_1]$, we have $\delta_1 \restriction_{\xi_1} = \delta_2 \restriction_{\xi_1}$. By the i.h. for subderivation $\Xi \vdash \hat{P}\ \mathsf{functor}[\xi_2]$, we have $\delta_1 \restriction_{\xi_2} = \delta_2 \restriction_{\xi_2}$. By Lemma E.12 we have $\delta_1 \restriction_{\xi_1 \cup \xi_2} = \delta_2 \restriction_{\xi_1 \cup \xi_2}$.

- **Case**

$$\frac{\Xi \vdash F_1\ \mathsf{functor}[\xi_1] \qquad \Xi \vdash F_2\ \mathsf{functor}[\xi_2]}{\Xi \vdash F_1 \oplus F_2\ \mathsf{functor}[\underbrace{\ \cdot\ }_{\xi}]}\ \mathrm{DeclFunc}\oplus$$

Similar to DeclFunc$I$ case.

(3) It suffices to show that,

for all $(\mathsf{fold}_F\ \alpha)\, v\, \boldsymbol{t} =_\tau b \in \mathscr{M}(F)$, if $\delta_1 \restriction_{FV(\boldsymbol{t})} = \delta_2 \restriction_{FV(\boldsymbol{t})}$ then $\delta_1(b) = \delta_2(b)$.

Assume $(\mathsf{fold}_F\ \alpha)\, v\, \boldsymbol{t} =_\tau b \in \mathscr{M}(F)$.

By definition, for all $k \in \{1,2\}$, we have $V \in \mu\, [\![F]\!]_{\delta_k}$ and

$$\delta_k(b) = [\![\boldsymbol{t}]\!]_{\delta_k}\, ((\mathit{fold}_{[\![F]\!]_{\delta_k}}\ [\![\alpha]\!]_{\delta_k})\, V) \tag{A}$$

Assume $\delta_1 \restriction_{FV(\boldsymbol{t})} = \delta_2 \restriction_{FV(\boldsymbol{t})}$.

By (A), it suffices to show

$$[\![t]\!]_{\delta_1}\left((\mathit{fold}_{[\![F]\!]_{\delta_1}}\ [\![\alpha]\!]_{\delta_1})\ V\right) = [\![t]\!]_{\delta_2}\left((\mathit{fold}_{[\![F]\!]_{\delta_2}}\ [\![\alpha]\!]_{\delta_2})\ V\right)$$

Because $\delta_1 \restriction_{FV(t)} = \delta_2 \restriction_{FV(t)}$, we have $[\![t]\!]_{\delta_1} = [\![t]\!]_{\delta_2}$, and it therefore suffices to show

$$(\mathit{fold}_{[\![F]\!]_{\delta_1}}\ [\![\alpha]\!]_{\delta_1})\ V = (\mathit{fold}_{[\![F]\!]_{\delta_2}}\ [\![\alpha]\!]_{\delta_2})\ V$$

But this follows from the fact that F and α are closed. □

Lemma E.18 (Type WF Substitution Soundness)**.**
Assume $\vdash \delta : \Xi_0$ *and* $\Xi_0; \cdot \vdash \sigma : \Xi; \cdot$.

(1) If $\Xi \vdash A\ \mathsf{type}[\xi]$ *then* $[\![[\sigma]A]\!]_\delta = [\![A]\!]_{[\![\sigma]\!]_\delta}$.

(2) If $\Xi \vdash \mathscr{F}\ \mathsf{functor}[\xi]$ *then* $[\![[\sigma]\mathscr{F}]\!]_\delta = [\![\mathscr{F}]\!]_{[\![\sigma]\!]_\delta}$.

(3) If $\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$ *then* $[\![[\sigma]\alpha]\!]_\delta = [\![\alpha]\!]_{[\![\sigma]\!]_\delta}$.

(4) If $\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi]$ *then* $[\![[\sigma]\mathscr{M}(F)]\!]_\delta = [\![\mathscr{M}(F)]\!]_{[\![\sigma]\!]_\delta}$.

Proof. By mutual induction on the structure of the given derivation. Use Lemma C.25 (Index Subst. Typing Sound), Lemma C.45 (Ix.-Level Id. Subs. Extension), Lemma D.28, Lemma C.24 (Index Substitution Soundness), Lemma E.2 (Equal Functor Mu), Lemma E.3 (Equal Func. Alg. Mu), Lemma D.28, Lemma C.12, Lemma C.51 (WF Syn. Substitution), and Lemma C.81 as needed. □

Lemma E.19 (liftapps Sound)**.**
If $\Xi; \Xi''; \overrightarrow{(a, (\mathsf{fold}_F\ \alpha)\ v\ _ =_\tau _)} \vdash \mathscr{O} \rightsquigarrow \breve{\Xi}; \mathscr{M}'(F); \mathscr{O}'$ *and* $\vdash \delta, \delta_2, \delta_1 : \Xi, \Xi'', \overrightarrow{a \overset{\mathsf{d}}{\div} \tau}$
and $dom(\delta_2) = dom(\Xi'')$ *and* $dom(\delta_1) = dom(\overrightarrow{a \overset{\mathsf{d}}{\div} \tau}) = \overrightarrow{a} = \pi_1\left(unzip\left(\overrightarrow{(a, (\mathsf{fold}_F\ \alpha)\ v\ _ =_\tau _)}\right)\right)$

then $FV(\mathscr{O}') \cap dom(\overrightarrow{a \overset{\mathsf{d}}{\div} \tau}) = \emptyset$ *and* $[\![\mathscr{O}]\!]_{\delta,\delta_2,\delta_1} = [\![\mathscr{O}']\!]_{\delta,\delta_2,[\![\breve{\Xi}]\!]^{\mathsf{fix}}_{\delta,\delta_2,\delta_1}}$;
moreover, if $V \in \mu\,[\![F]\!]_\delta$ *then:*
$d_k = (fold_{[\![F]\!]_\delta}\ [\![\alpha_k]\!]_\delta)\,V$ *for all* $(d_k/a_k, (\mathsf{fold}_F\ \alpha_k)\,\nu_ =_{\tau_k} _) \in zip(\delta_1)(\overrightarrow{(\mathsf{fold}_F\ \alpha)\,\nu_ =_\tau _})$
implies $[\![\mathscr{M}'(F)]\!]_{\delta,\delta_2,[\![\breve{\Xi}]\!]^{\mathsf{fix}}_{\delta,\delta_2,\delta_1}}$ V *holds.*

Proof. By structural induction on $\Xi, \Xi''; \overrightarrow{(a, (\mathsf{fold}_F\ \alpha)\,\nu_ =_\tau _)} \vdash \mathscr{O} \rightsquigarrow \breve{\Xi}; \mathscr{M}'(F); \mathscr{O}'$. Use weakening and the fact that weakened derivations are semantically equivalent when interpreted (weakening invariance). Use Lemma C.54 (liftapps WF). □

Lemma E.20 (Unrolling Soundness)**.** *If* $\Xi \vdash \wr\, \overrightarrow{\beta}; G; \mathscr{M}(F) \wr \triangleq {}^{\mathsf{d}}\Theta; R$ *and* $\vdash \delta : \Xi$
then the set of all semantic values $V \in [\![G]\!]_\delta\,(\mu\,[\![F]\!]_\delta)$ *such that*

$$\forall(\beta, (\!|\alpha|\!)_F\ \nu\, \boldsymbol{t} =_\tau t) \in zip(\overrightarrow{\beta})(\mathscr{M}(F)).\ [\![\boldsymbol{t}]\!]_\delta\,([\![\beta]\!]_\delta\,([\![G]\!]_\delta\,(fold_{[\![F]\!]_\delta}\ [\![\alpha]\!]_\delta)\,V)) = [\![t]\!]_\delta$$

is equal to the set $[\![\exists^{\mathsf{d}}\Theta.\, R \wedge {}^{\mathsf{d}}\Theta]\!]_\delta$.

Proof. By structural induction on the given unrolling derivation. We implicitly use Lemma C.54 (liftapps WF). We may elide uses of Lemma C.21 (Sorting Soundness) and weakening/restriction reasoning such as Lemma C.6 and Lemma C.7. We only show one case; the other cases are straightforward.

- **Case**

$$\frac{\begin{array}{c}\overrightarrow{a \stackrel{\mathrm{d}}{\div} \tau} = \overrightarrow{a} \stackrel{\mathrm{d}}{\div} \mathscr{M}(F) \\ \Xi, \overrightarrow{a \stackrel{\mathrm{d}}{\div} \tau, a\,\mathrm{Id}} \vdash \wr \overrightarrow{q \Rightarrow t'}; \hat{I}; \mathscr{M}(F) \int \doteq \Xi'', \overrightarrow{\psi''}; R'' \\ \Xi; \Xi''; zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash \overrightarrow{\psi''} \rightsquigarrow \check{\Xi}_1; \mathscr{M}_1(F); \overrightarrow{\psi'} \\ \Xi; \Xi''; zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash R'' \rightsquigarrow \check{\Xi}_2; \mathscr{M}_2(F); R' \\ \check{\Xi} = \check{\Xi}_1 \cup \check{\Xi}_2 \qquad \mathscr{M}'(F) = \mathscr{M}_1(F) \cup \mathscr{M}_2(F) \\ dom(\Xi') \cap dom(\Xi, \overrightarrow{a \stackrel{\mathrm{d}}{\div} \tau}, \Xi'', \check{\Xi}) = \emptyset \qquad \rho = \Xi'/\check{\Xi} \text{ is a variable renaming}\end{array}}{\Xi \vdash \wr \overrightarrow{(a,q) \Rightarrow t'}; \mathrm{Id} \otimes \hat{I}; \mathscr{M}(F) \int \doteq \Xi', \Xi'', [\rho]\overrightarrow{\psi'}; \left\{ \nu : \mu F \;\middle|\; [\rho]\mathscr{M}'(F) \right\} \times [\rho]R'} \wr \mathrm{Id} \int$$

  - ($\subseteq$) Assume V is in the left-hand side.

    Unpacking definitions, there exist V_1 and V_2

    such that $V = (V_1, V_2)$ and $V_1 \in \mu \llbracket F \rrbracket_\delta$ and $V_2 \in \llbracket \hat{I} \rrbracket_\delta (\mu \llbracket F \rrbracket_\delta)$

    and for all $((a,q) \Rightarrow t', (\mathsf{fold}_F\ \alpha)\, \nu\, \boldsymbol{t} =_\tau t) \in zip(\overrightarrow{(a,q) \Rightarrow t'})(\mathscr{M}(F))$ we have

$$\llbracket \boldsymbol{t} \rrbracket_\delta (\llbracket q \Rightarrow t' \rrbracket_{\delta, d/a} (\llbracket \hat{I} \rrbracket_\delta\ (fold_{\llbracket F \rrbracket_\delta}\ \llbracket \alpha \rrbracket_\delta)\ V_2)) = \llbracket t \rrbracket_\delta$$

    where $d = (fold_{\llbracket F \rrbracket_\delta}\ \llbracket \alpha \rrbracket_\delta)\ V_1$ which is in $\llbracket \tau \rrbracket$ by Lemma E.6 (Fold Membership).

    Thus we have $\mathrm{a} \vdash \delta_1 : \overrightarrow{a \stackrel{\mathrm{d}}{\div} \tau, a\,\mathrm{Id}}$ such that each $d/a \in \delta_1$ has the above properties. Therefore,

$$\llbracket \boldsymbol{t} \rrbracket_\delta (\llbracket q \Rightarrow t' \rrbracket_{\delta, \delta_1} (\llbracket \hat{I} \rrbracket_\delta\ (fold_{\llbracket F \rrbracket_\delta}\ \llbracket \alpha \rrbracket_\delta)\ V_2)) = \llbracket t \rrbracket_\delta$$

    for all $(q \Rightarrow t', (\!|\alpha|\!)_F\, \nu\, \boldsymbol{t} =_\tau t) \in zip(\overrightarrow{q \Rightarrow t'})(\mathscr{M}(F))$ and we label this (**).

    By definitions and Lemma D.26 (Refinement Subset of Erasure), it suffices to

show there exists $\delta' \in [\![\Xi', \Xi'']\!]$ such that

$$(V_1, V_2) \in [\![\{\nu : \mu F \mid [\rho].\mathscr{M}'(F)\} \times [\rho]R']\!]_{\delta,\delta'}$$

and $[\![\psi']\!]_{\delta,\delta'} = \{\bullet\}$ for all $\psi' \in [\rho]\overrightarrow{\psi'}$.

We already have $V_1 \in \mu\,[\![F]\!]_\delta = \mu\,[\![F]\!]_{\delta,\delta'}$.

By i.h. (and definitions and weakening invariance),

the set of elements $V \in [\![\hat{I}]\!]_\delta\,(\mu\,[\![F]\!]_\delta)$ such that

for all $(q \Rightarrow t', (\!|\alpha|\!)_F\,\nu\,\boldsymbol{t} =_\tau t) \in zip(\overrightarrow{q \Rightarrow t'})(\mathscr{M}(F))$,

$$[\![\boldsymbol{t}]\!]_\delta\,([\![q \Rightarrow t']\!]_{\delta,\delta_1}\,([\![\hat{I}]\!]_\delta\,(fold_{[\![F]\!]_\delta}\,[\![\alpha]\!]_\delta)\,V)) = [\![t]\!]_\delta$$

is equal to the set $\left[\!\!\left[\exists \Xi''.\, R'' \wedge \overrightarrow{\psi''}\right]\!\!\right]_{\delta,\delta_1}$.

This with (**) gives $V_2 \in \left[\!\!\left[\exists \Xi''.\, R'' \wedge \overrightarrow{\psi''}\right]\!\!\right]_{\delta,\delta_1}$.

By def. of denotation, there exists $\delta_2 \in [\![\Xi'']\!]$

such that $V_2 \in [\![R'']\!]_{\delta,\delta_1,\delta_2}$ and $[\![\psi'']\!]_{\delta,\delta_1,\delta_2} = \{\bullet\}$ for all $\psi'' \in \overrightarrow{\psi''}$.

By two uses of Lemma E.19 (liftapps Sound) (including its "moreover")

and weakening invariance it follows that

$[\![R'']\!]_{\delta,\delta_2,\delta_1} = [\![R']\!]_{\delta,\delta_2,[\![\check{\Xi}]\!]^{\mathsf{fix}}_{\delta,\delta_2,\delta_1}}$ and $\left[\!\!\left[\overrightarrow{\psi''}\right]\!\!\right]_{\delta,\delta_2,\delta_1} = \left[\!\!\left[\overrightarrow{\psi'}\right]\!\!\right]_{\delta,\delta_2,[\![\check{\Xi}]\!]^{\mathsf{fix}}_{\delta,\delta_2,\delta_1}}$

and $[\![\mathscr{M}'(F)]\!]_{\delta,\delta_2,[\![\check{\Xi}]\!]^{\mathsf{fix}}_{\delta,\delta_2,\delta_1}} V_1$.

By Lemma E.18 (Type WF Substitution Soundness) and Lemma C.24 (Index Substitution Soundness) and definitions and $[\rho^{-1}]\rho = id_{\check{\Xi}}$ it follows that $\left([\![\rho^{-1}]\!]_{[\![\check{\Xi}]\!]^{\mathsf{fix}}_{\delta,\delta_2,\delta_1}}, \delta_2\right)$ meets the requirements for δ' (where ρ^{-1} is defined by $\cdot^{-1} = \cdot$ and $(\rho, b/\check{a}^{b(\boldsymbol{u})})^{-1} = \rho^{-1}, \check{a}^{b(\boldsymbol{u})}/b$).

– ($\supseteq$) Assume $V \in \left[\!\!\left[\exists \Xi', \Xi''.\, (\{\nu : \mu F \mid [\rho].\mathscr{M}'(F)\} \times [\rho]R') \wedge [\rho]\overrightarrow{\psi'}\right]\!\!\right]_\delta$.

By def. of denotation,

there exist $\delta_0 \in [\![\Xi', \Xi'']\!]$, $V_1 \in \mu\,[\![F]\!]_\delta$, and $V_2 \in [\![[\rho]R']\!]_{\delta,\delta_0}$

such that $V = (V_1, V_2)$ and $[\![[\rho]\mathscr{M}'(F)]\!]_{\delta,\delta_0}\, V_1$

and $[\![\psi']\!]_{\delta,\delta_0} = \{\bullet\}$ for all $\psi' \in [\rho]\,\overrightarrow{\psi'}$.

Let $\delta_2 = \delta_0 \restriction_{\Xi''}$. Let $\delta_1' = \delta_0 \restriction_{\Xi'}$.

Let δ_1 be defined similarly to the δ_1 of the other direction ($\subseteq$).

By Lemma E.18 (Type WF Substitution Soundness)

and Lemma C.24 (Index Substitution Soundness),

$V_2 \in [\![R']\!]_{\delta,\delta_2,[\![\rho]\!]_{\delta_1'}}$ and $[\![\mathscr{M}'(F)]\!]_{\delta,\delta_2,[\![\rho]\!]_{\delta_1'}}\, V_1$

and $[\![\psi']\!]_{\delta,\delta_2,[\![\rho]\!]_{\delta_1'}} = \{\bullet\}$ for all $\psi' \in \overrightarrow{\psi'}$.

By (two uses of) Lemma E.19 (liftapps Sound) (including its "moreover")

and weakening invariance it follows that

$[\![R'']\!]_{\delta,\delta_2,\delta_1} = [\![R']\!]_{\delta,\delta_2,[\![\check{\Xi}]\!]^{\mathsf{fix}}_{\delta,\delta_2,\delta_1}}$ and $\left[\!\left[\overrightarrow{\psi''}\right]\!\right]_{\delta,\delta_2,\delta_1} = \left[\!\left[\overrightarrow{\psi'}\right]\!\right]_{\delta,\delta_2,[\![\check{\Xi}]\!]^{\mathsf{fix}}_{\delta,\delta_2,\delta_1}}$

and $[\![\mathscr{M}'(F)]\!]_{\delta,\delta_2,[\![\check{\Xi}]\!]^{\mathsf{fix}}_{\delta,\delta_2,\delta_1}}\, V_1$.

It follows

from Lemma E.17 (Soundness of Value-Determined Dependencies)

and Lemma E.15 (Dependency Agreement Closure)

(with $\xi_{\mathscr{M}'} - (\Xi, \Xi'') \vdash \check{\Xi}$ det from Lemma C.54 and Lemma B.4),

that $[\![\rho]\!]_{\delta_1'} = [\![\check{\Xi}]\!]^{\mathsf{fix}}_{\delta,\delta_2,\delta_1}$.

Therefore, $V_2 \in \left[\!\left[\exists \Xi''.\, R'' \wedge \overrightarrow{\psi''}\right]\!\right]_{\delta,\delta_2}$,

and the goal follows straightforwardly by the i.h. and definitions. □

Lemma E.21 (μ Subset)**.** *If* $\Xi \vdash F$ functor[_] *and* $\Xi \vdash G$ functor[_] *and* $\vdash \delta : \Xi$

and $[\![F]\!]_\delta X \subseteq [\![G]\!]_\delta X$ *for all* $X \in$ **Set**

then $\mu\,[\![F]\!]_\delta \subseteq \mu\,[\![G]\!]_\delta$.

Proof. It suffices to show $[\![F]\!]^n_\delta \emptyset \subseteq [\![G]\!]^n_\delta \emptyset$ for all $n \in \mathbb{N}$ by induction on n.

The $n = 0$ case is trivial. Assume $n > 0$.

$$
\begin{aligned}
[\![F]\!]^n_\delta &= [\![F]\!]^{n-1}_\delta ([\![F]\!]_\delta \emptyset) && \text{By def.} \\
&= [\![F]\!]_\delta ([\![F]\!]^{n-1}_\delta \emptyset) && \text{By Lemma D.2 (Functor Apps. Commute)} \\
&\subseteq [\![F]\!]_\delta ([\![G]\!]^{n-1}_\delta \emptyset) && \text{By i.h. and (refined version of) Lemma D.3 (Functor Monotone)} \\
&\subseteq [\![G]\!]_\delta ([\![G]\!]^{n-1}_\delta \emptyset) && \text{Given} \\
&= [\![G]\!]^n_\delta \emptyset && \text{By def., Lemma D.2}
\end{aligned}
$$

□

Lemma E.22 (Fold Subset)**.** *If* $\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$ *and* $\Xi \vdash \beta : G(\tau) \Rightarrow \tau$ *and* $\vdash \delta : \Xi$ *and* $[\![\alpha]\!]_\delta = [\![\beta]\!]_\delta$ *on* $[\![F]\!]_\delta [\![\tau]\!]$ *and* $[\![F]\!]_\delta X \subseteq [\![G]\!]_\delta X$ *for all* $X \in$ **Set** *then* $fold_{[\![F]\!]_\delta} [\![\alpha]\!]_\delta = fold_{[\![G]\!]_\delta} [\![\beta]\!]_\delta$ *on* $\mu [\![F]\!]_\delta$.

Proof. Follows from Lemma E.6 (Fold Membership), Lemma E.9 (Semantic Fold), Lemma E.21 (μ Subset), and the given information. □

Lemma E.23 (Subtyping Soundness)**.**

(1) If $\Theta \vdash A \leq^{\pm} B$ *and* $\delta \in [\![\Theta]\!]$ *then* $[\![A]\!]_\delta \subseteq [\![B]\!]_\delta$.

(2) If $\Xi \vdash \alpha; F \leq_\tau \beta; G$ *and* $\delta \in [\![\Xi]\!]$ *then* $[\![F]\!]_\delta X \subseteq [\![G]\!]_\delta X$ *for any* $X \in$ **Set**.

(3) If $\Xi \vdash \alpha; F \leq_\tau \beta; G$ *and* $\delta \in [\![\Xi]\!]$ *then* $[\![\alpha]\!]_\delta = [\![\beta]\!]_\delta$ *on* $[\![F]\!]_\delta [\![\tau]\!]$.

(4) If $\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)$ *and* $\delta \in [\![\Theta]\!]$
then for all V *we know* $[\![\mathscr{M}'(F')]\!]_\delta V$ *implies* $[\![\mathscr{M}(F)]\!]_\delta V$.

Proof. By mutual induction on the given subtyping/submeasuring derivation, case analyzing its concluding rule. We only show parts (1) and (4); parts (2) and (3) are similar but part

(3) also uses Lemma C.33 (Ix. Equiv. Sound) in the Meas$\leq I$ case, as well as Lemma E.17 (Soundness of Value-Determined Dependencies) and Lemma E.15 (Dependency Agreement Closure) in the Meas$\leq\exists$R case, and the latter (Meas$\leq\exists$R) is otherwise similar to the $\leq^+\exists$R case of part (1). We elide uses of Lemma E.5 ($[\![-]\!]$ Weakening Invariant).

- **Case**

$$\frac{}{\Theta \vdash 0 \leq^+ 0}\leq^+ 0$$

$$[\![0]\!]_\delta = \emptyset \subseteq \emptyset = [\![0]\!]_\delta$$

- **Case**

$$\frac{}{\Theta \vdash 1 \leq^+ 1}\leq^+ 1$$

$$[\![1]\!]_\delta = \{\bullet\} \subseteq \{\bullet\} = [\![1]\!]_\delta$$

- **Case**

$$\frac{\Theta \vdash R_1 \leq^+ R_1' \qquad \Theta \vdash R_2 \leq^+ R_2'}{\Theta \vdash R_1 \times R_2 \leq^+ R_1' \times R_2'}\leq^+ \times$$

$$\begin{aligned}
[\![R_1 \times R_2]\!]_\delta &= [\![R_1]\!]_\delta \times [\![R_2]\!]_\delta && \text{By def. of denotation}\\
&\subseteq [\![R_1']\!]_\delta \times [\![R_2']\!]_\delta && \text{By i.h. (twice) and set theory}\\
&= [\![R_1' \times R_2']\!]_\delta && \text{By def. of denotation}
\end{aligned}$$

- **Case**

$$\frac{\Theta \vdash P_1 \leq^+ P_1' \qquad \Theta \vdash P_2 \leq^+ P_2'}{\Theta \vdash P_1 + P_2 \leq^+ P_1' + P_2'}\leq^+ +$$

$$
\begin{aligned}
\Theta \vdash P_k \leq^+ P'_k & \qquad \text{Subderivations} \\
[\![P_k]\!]_\delta \subseteq [\![P'_k]\!]_\delta & \qquad \text{By i.h. (twice)} \\
[\![P_1 + P_2]\!]_\delta = [\![P_1]\!]_\delta \uplus [\![P_2]\!]_\delta & \qquad \text{By def. of denotation} \\
= [\![P'_1]\!]_\delta \uplus [\![P'_2]\!]_\delta & \qquad \text{By above } \subseteq \\
= [\![P'_1 + P'_2]\!]_\delta & \qquad \text{By def. of denotation}
\end{aligned}
$$

- **Case**
$$\frac{\Theta, \overrightarrow{\varphi} \vdash R \leq^+ P}{\Theta \vdash R \wedge \overrightarrow{\varphi} \leq^+ P} \leq^+\wedge\mathrm{L}$$
Suppose $V \in [\![R \wedge \overrightarrow{\varphi}]\!]_\delta$. By definition of denotation, $V \in [\![R]\!]_\delta$ and $[\![\varphi]\!]_\delta = \{\bullet\}$ for all $\varphi \in \overrightarrow{\varphi}$. By repeated Prop$\delta$, $\vdash \delta : \Theta, \overrightarrow{\varphi}$. By i.h., $V \in [\![P]\!]_\delta$.

- **Case**
$$\frac{\Theta, {}^{\mathsf{d}}\Xi \vdash Q \leq^+ P}{\Theta \vdash \exists^{\mathsf{d}}\Xi.\, Q \leq^+ P} \leq^+\exists\mathrm{L}$$
Similar to $\leq^+\wedge$L case.

- **Case**
$$\frac{\Theta \vdash R \leq^+ Q \qquad \Theta \vdash \overrightarrow{\varphi}\,\mathsf{true}}{\Theta \vdash R \leq^+ Q \wedge \overrightarrow{\varphi}} \leq^+\wedge\mathrm{R}$$

$V \in [\![\Theta \vdash R\ \mathsf{type}[\xi]]\!]_\delta$	Suppose
$\Theta \vdash R \leq^+ Q$	Subderivation
$[\![\Theta \vdash R\ \mathsf{type}[\xi]]\!]_\delta \subseteq [\![\Theta \vdash Q\ \mathsf{type}[\xi']]\!]_\delta$	By i.h.
$V \in [\![\Theta \vdash Q\ \mathsf{type}[\xi']]\!]_\delta$	Follows from above
$[\![\Theta \vdash Q\ \mathsf{type}[\xi']]\!]_\delta \subseteq [\![\lvert Q \rvert]\!]$	By Lemma D.26 (Refinement Subset of Erasure)
$V \in [\![\lvert Q \rvert]\!]$	Follows from above
$\Theta \vdash \overrightarrow{\varphi}\ \mathsf{true}$	Premise

By inversion on PropTrue, $[\![\varphi]\!]_\delta = \{\bullet\}$ for all $\varphi \in \overrightarrow{\varphi}$. Therefore,

$V \in \{V \in [\![\lvert Q \rvert]\!] \mid V \in [\![Q]\!]_\delta \wedge [\![\varphi]\!]_\delta = \{\bullet\} \text{ for all } \varphi \in \overrightarrow{\varphi}\}$	Follows from above
$= [\![Q \wedge \overrightarrow{\varphi}]\!]_\delta$	By def. of denotation

- **Case**
$$\frac{\overset{\mathsf{d}}{\div}\Theta \vdash \overrightarrow{t}/^{\mathsf{d}}\Xi' : {}^{\mathsf{d}}\Xi' \qquad \Theta \vdash R \leq^+ [\overrightarrow{t}/^{\mathsf{d}}\Xi']Q}{\Theta \vdash R \leq^+ \exists^{\mathsf{d}}\Xi'.\, Q}\ \leq^+\exists\mathrm{R}$$

$V \in [\![R]\!]_\delta$	Suppose
$\Theta \vdash R \leq^+ [\overrightarrow{t}/^{\mathsf{d}}\Xi']Q$	Subderivation
$[\![R]\!]_\delta \subseteq \left[\!\left[[\overrightarrow{t}/^{\mathsf{d}}\Xi']Q\right]\!\right]_\delta$	By i.h.
$\overset{\mathsf{d}}{\div}\Theta \vdash \overrightarrow{t}/^{\mathsf{d}}\Xi' : {}^{\mathsf{d}}\Xi'$	Premise
$\left[\!\left[\overrightarrow{t}/^{\mathsf{d}}\Xi'\right]\!\right]_\delta \in \left[\!\left[{}^{\mathsf{d}}\Xi'\right]\!\right]$	By repeated Lemma C.21 (Sorting Soundness)

But

$$\begin{aligned}
\left[\!\left[[\overrightarrow{t}/^{\mathsf{d}}\Xi']Q\right]\!\right]_\delta &= \left[\!\left[[\Theta/\Theta, \overrightarrow{t}/^{\mathsf{d}}\Xi']Q\right]\!\right]_\delta && \text{Identity substitution} \\
&= [\![Q]\!]_{[\![\Theta/\Theta, \overrightarrow{t}/^{\mathsf{d}}\Xi']\!]_\delta} && \text{Lemma E.18 (Type WF Substitution Soundness)} \\
&= [\![Q]\!]_{\delta, [\![\sigma]\!]_\delta} && \text{By def. of denotation}
\end{aligned}$$

Therefore, $V \in [\![Q]\!]_{\delta,[\![\sigma]\!]_\delta}$. Further:

$$\begin{aligned}
[\![Q]\!]_{\delta,[\![\sigma]\!]_\delta} \subseteq [\![|Q|]\!] && \text{By Lemma D.26 (Refinement Subset of Erasure)} \\
V \in [\![|Q|]\!] && \text{Follows from above}
\end{aligned}$$

Therefore,

$$\begin{aligned}
V \in \left\{ V \in [\![|Q|]\!] \;\middle|\; \exists \delta' \in \left[\!\left[{}^{\mathsf{d}}\Xi'\right]\!\right]. V \in [\![Q]\!]_{\delta,\delta'} \right\} && \text{Follows from above (set } \delta' = [\![\sigma]\!]_\delta) \\
= \left[\!\left[\exists^{\mathsf{d}}\Xi'.Q\right]\!\right]_\delta && \text{By def. of denotation}
\end{aligned}$$

- **Case**
$$\frac{\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)}{\Theta \vdash \left\{\nu : \mu F' \;\middle|\; \mathscr{M}'(F')\right\} \leq^+ \{\nu : \mu F \mid \mathscr{M}(F)\}} \leq^+\mu$$
By definitions and i.h., part (4).

- **Case**
$$\frac{\Theta \vdash N \leq^- N}{\Theta \vdash \downarrow N \leq^+ \downarrow N} \leq^+\downarrow$$
Straightforward (use i.h. for subderivation $\Theta \vdash N \leq^- N'$ and the definition of denotation).

- **Case** $\leq^-\uparrow$: Similar to case for dual rule $\leq^+\downarrow$.

- **Cases** $\leq^{-}\supset$R, $\leq^{-}\forall$R: Similar to cases for dual rules $\leq^{+}\wedge$L and $\leq^{+}\exists$L.

- **Case** $\leq^{-}\supset$L: Similar to case for dual rule $\leq^{+}\wedge$R.

- **Case** $\leq^{-}\forall$L: Similar to case for dual rule $\leq^{+}\exists$R.

- **Case**
$$\frac{\Theta \vdash R' \leq^{+} R \qquad \Theta \vdash L \leq^{-} L'}{\Theta \vdash R \to L \leq^{-} R' \to L'} \leq^{-}\to$$
Suppose $f \in [\![R \to L]\!]_\delta$. We want to show that $f \in [\![R' \to L']\!]_\delta$.

By def. of denotation, $f \in [\![|R \to L|]\!]$ and for all $V \in [\![R]\!]_\delta$, we have $f(V) \in [\![L]\!]_\delta$.

By def. of denotation, it suffices to show that $f \in [\![|R' \to L'|]\!]$ and that for all $V \in [\![R']\!]_\delta$, we have $f(V) \in [\![L']\!]_\delta$. The former follows form Lemma D.29 (Subtyping Erases to Equality). To prove the latter, suppose $V \in [\![R']\!]_\delta$. By the i.h. for the positive subtyping subderivation, $V \in [\![R]\!]_\delta$. Therefore, $f(V) \in [\![L]\!]_\delta$. By the i.h. for the negative subtyping subderivation, $f(V) \in [\![L']\!]_\delta$.

As for part (4), the case where $\mathscr{M}(F) = \cdot_F$ holds vacuously,

and the other case follows from the i.h. and the following:

$\cdot \vdash \alpha'; F' \leq_\tau \alpha; F$	Subderivation
$\Theta \vdash \alpha'; F' \leq_\tau \alpha; F$	By weakening
$[\![F']\!]_\delta X \subseteq [\![F]\!]_\delta X$ for all $X \in \mathbf{Set}$	By i.h.
$[\![\alpha']\!]_\delta = [\![\alpha]\!]_\delta$ on $[\![F']\!]_\delta [\![\tau]\!]$	$''$
$\mu [\![F']\!]_\delta \subseteq \mu [\![G]\!]_\delta$	By Lemma E.21 (μ Subset)
$fold_{[\![F']\!]_\delta} [\![\alpha']\!]_\delta = fold_{[\![F]\!]_\delta} [\![\alpha]\!]_\delta$ on $\mu [\![F']\!]_\delta$	By Lemma E.22 (Fold Subset)
$\stackrel{\mathsf{d}}{\div}\Theta; [\tau] \vdash \boldsymbol{t}' \equiv \boldsymbol{t} : \kappa$	Premise
$[\![\boldsymbol{t}']\!]_\delta = [\![\boldsymbol{t}]\!]_\delta$	By Lemma C.33 (Ix. Equiv. Sound)
$\stackrel{\mathsf{d}}{\div}\Theta \vdash t' = t$ true	Premise
$[\![t']\!]_\delta = [\![t]\!]_\delta$	By inversion on PropTrue $\square$

Lemma E.24 (Fold Continuous)**.** *Assume* $\vdash \delta : \Xi$ *and* $\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$.

(1) If $[\![\alpha]\!]_\delta$ *is monotone then* $fold_{[\![F]\!]_\delta} [\![\alpha]\!]_\delta$ *is monotone.*

(2) If $[\![\alpha]\!]_\delta$ *respects least upper bounds*
then $fold_{[\![F]\!]_\delta} [\![\alpha]\!]_\delta$ *respects least upper bounds.*

(3) If $[\![\alpha]\!]_\delta$ *is continuous then* $fold_{[\![F]\!]_\delta} [\![\alpha]\!]_\delta$ *is continuous.*

Proof. We only show part (1). Part (2) is similar. Part (3) follows from parts (1) and (2) and the definition of continuous.

By definition of *fold* (Def. E.1), it suffices to show that $fold^k_{[\![F]\!]_\delta} [\![\alpha]\!]_\delta$ is monotone for all $k \in \mathbb{N}$.

The empty function $fold^0_{[\![F]\!]_\delta}\ [\![\alpha]\!]_\delta$ vacuously is monotone.

For the inductive step, we assume $fold^n_{[\![F]\!]_\delta}\ [\![\alpha]\!]_\delta$ is monotone and prove $fold^{n+1}_{[\![F]\!]_\delta}\ [\![\alpha]\!]_\delta$ is monotone. By definition,

$$fold^{n+1}_{[\![F]\!]_\delta}\ [\![\alpha]\!]_\delta = [\![\alpha]\!]_\delta \circ ([\![F]\!]_\delta\ (fold^n_{[\![F]\!]_\delta}\ [\![\alpha]\!]_\delta))$$

By i.h., $fold^n_{[\![F]\!]_\delta}\ [\![\alpha]\!]_\delta$ is monotone. It is straightforward to check that $[\![F]\!]_\delta$ takes monotone functions to monotone functions. Therefore, $[\![F]\!]_\delta\ (fold^n_{[\![F]\!]_\delta}\ [\![\alpha]\!]_\delta)$ is monotone. We are given that $[\![\alpha]\!]_\delta$ is monotone. Because the composition of monotone functions is monotone, $[\![\alpha]\!]_\delta \circ ([\![F]\!]_\delta\ (fold^n_{[\![F]\!]_\delta}\ [\![\alpha]\!]_\delta))$ is monotone, which concludes the proof. □

Lemma E.25. *If* $\Xi \vdash \mathscr{F}$ $\mathsf{functor}[_]$ *and* $\Xi \vdash F$ $\mathsf{functor}[_]$ *and* $k \in \mathbb{N}$ *and* $V \in [\![|\mathscr{F}|]\!]\ ([\![|F|]\!]^k\ \emptyset)$ *and* $\vdash \delta : \Xi$ *and* $V \in [\![\mathscr{F}]\!]_\delta\ (\mu\ [\![F]\!]_\delta)$*, then* $V \in [\![\mathscr{F}]\!]_\delta\ ([\![F]\!]^k_\delta\ \emptyset)$.

Proof. By lexicographic induction, first, on k, and, second, on the structure of $\mathscr{F}$. □

Lemma E.26 (Upward Closure)**.** *Assume* $\vdash \delta : \Xi$.

(1) If $\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$ *then* $[\![\alpha]\!]_\delta$ *is monotone.*

(2) If $\Xi \vdash \mathscr{F}$ $\mathsf{functor}[_]$ *and* $\Xi \vdash F$ $\mathsf{functor}[_]$ *and* $k \in \mathbb{N}$
and $V \in [\![\mathscr{F}]\!]_\delta\ ([\![F]\!]^k_\delta\ \emptyset)$ *and* $V \sqsubseteq_{[\![|\mathscr{F}|]\!]\ ([\![|F|]\!]^k\ \emptyset)} V'$
then $V' \in [\![\mathscr{F}]\!]_\delta\ ([\![F]\!]^k_\delta\ \emptyset)$.

(3) If $\Xi \vdash A$ $\mathsf{type}[_]$ *and* $V \in [\![A]\!]_\delta$ *and* $V \sqsubseteq_{[\![|A|]\!]} V'$ *then* $V' \in [\![A]\!]_\delta$.

Proof. By lexicographic induction, first, on the structure of A or F (parts (1), (2) and (3), mutually), and, second, on $\langle k, \mathscr{F}$ structure$\rangle$ (part (2)), where $\langle \ldots \rangle$ denotes lexicographic order.

(1) By Lemma E.7 (Type WF Sound), $[\![\alpha]\!]_\delta : [\![F]\!]_\delta\, [\![\tau]\!] \to [\![\tau]\!]$. We case analyze the given algebra well-formedness derivation.

- **Case**
$$\frac{\Xi, {}^{\mathsf{d}}\Xi' \vdash (\top, q) \Rightarrow t : (\underline{Q} \otimes \hat{P})(\tau) \Rightarrow \tau}{\Xi \vdash (\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q) \Rightarrow t : (\underline{\exists^{\mathsf{d}}\Xi'.\, Q} \otimes \hat{P})(\tau) \Rightarrow \tau}\ \text{DeclAlg}\exists$$
Suppose $V \sqsubseteq_{[\![\underline{\exists^{\mathsf{d}}\Xi'.\, Q} \otimes \hat{P}]\!]_\delta [\![\tau]\!]} V'$. By def. of $[\![-]\!]$, $\left[\!\left[\underline{\exists^{\mathsf{d}}\Xi'.\, Q} \otimes \hat{P}\right]\!\right]_\delta [\![\tau]\!] = \left[\!\left[\exists^{\mathsf{d}}\Xi'.\, Q\right]\!\right]_\delta \times \left[\!\left[\hat{P}\right]\!\right]_\delta [\![\tau]\!]$. Therefore, there exist V_1 and V_1' in $[\![\exists \xi'.\, Q]\!]_\delta$ and V_2 and V_2' in $\left[\!\left[\hat{P}\right]\!\right]_\delta [\![\tau]\!]$ such that $V = (V_1, V_2)$ and $V' = (V_1', V_2')$ and $V_1 \sqsubseteq_{[\![\exists^{\mathsf{d}}\Xi'.\, Q]\!]_\delta} V_1'$ and $V_2 \sqsubseteq_{[\![\hat{P}]\!]_\delta [\![\tau]\!]} V_2'$.

  On one hand,

  $\Xi, {}^{\mathsf{d}}\Xi' \vdash (\top, q) \Rightarrow t : (\underline{Q} \otimes \hat{P})(\tau) \Rightarrow \tau$ Subderivation

  By def., there exists $\delta' \in \left[\!\left[{}^{\mathsf{d}}\Xi'\right]\!\right]$ such that $V_1 \in [\![Q]\!]_{\delta,\delta'}$ and

  $$\left[\!\left[(\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q) \Rightarrow t\right]\!\right]_\delta\ (V_1, V_2) = [\![(\top, q) \Rightarrow t]\!]_{\delta,\delta'}\ (V_1, V_2)$$
  $$\sqsubseteq [\![(\top, q) \Rightarrow t]\!]_{\delta,\delta'}\ (V_1', V_2') \quad \text{By i.h.}$$

  On the other hand, by def., there exists $\delta'' \in \left[\!\left[{}^{\mathsf{d}}\Xi'\right]\!\right]$ such that $V_1' \in [\![Q]\!]_{\delta,\delta''}$ and

  $$\left[\!\left[(\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q) \Rightarrow t\right]\!\right]_\delta\ (V_1', V_2') = [\![(\top, q) \Rightarrow t]\!]_{\delta,\delta''}\ (V_1', V_2')$$

  By def. of $V_1 \sqsubseteq_{[\![\exists^{\mathsf{d}}\Xi'.\, Q]\!]_\delta} V_1'$, we have $V_1 \sqsubseteq_{[\![|Q|]\!]} V_1'$. Above, we have $V_1 \in [\![Q]\!]_{\delta,\delta'}$. By i.h. (part (3)), $V_1' \in [\![Q]\!]_{\delta,\delta'}$. But (above) $V_1' \in [\![Q]\!]_{\delta,\delta''}$, so, by Lemma E.17 (Soundness of Value-Determined Dependencies) and Lemma E.15 (Dependency Agreement Closure), $\delta' = \delta''$, which concludes this case.

- The remaining cases are straightforward.

(2) We case analyze $\mathscr{F}$.

- **Case** $\mathscr{F} = I$:

$$\begin{aligned}
[\![I]\!]\,([\![|F|]\!]^k\,\emptyset) &= \{\bullet\} && \text{By def.}\\
V &\sqsubseteq_{\{\bullet\}} V' && \text{Rewrite given}\\
V' &\in \{\bullet\} && \text{By def. of } \sqsubseteq\\
&= [\![I]\!]_\delta\,([\![F]\!]_\delta^k\,\emptyset) && \text{By def.}
\end{aligned}$$

- **Case** $\mathscr{F} = \mathrm{Id}$:

$$\begin{aligned}
[\![\mathrm{Id}]\!]\,([\![|F|]\!]^k\,\emptyset) &= [\![|F|]\!]^k\,\emptyset && \text{By def.}\\
V &\sqsubseteq_{[\![|F|]\!]\,([\![|F|]\!]^{k-1}\,\emptyset)} V' && \text{Rewrite given}\\
V &\in [\![F]\!]_\delta\,([\![F]\!]_\delta^{k-1}\,\emptyset) && \text{Rewrite given similarly}\\
V' &\in [\![F]\!]_\delta\,([\![F]\!]_\delta^{k-1}\,\emptyset) && \text{By i.h.}\\
&= [\![\mathrm{Id}]\!]_\delta\,([\![F]\!]_\delta^k\,\emptyset) && \text{By def.}
\end{aligned}$$

- **Case** $\mathscr{F} = \underline{P}$:

$$\begin{aligned}
[\![\underline{P}]\!]\,([\![|F|]\!]^k\,\emptyset) &= [\![P]\!] && \text{By def.}\\
V &\sqsubseteq_{[\![P]\!]} V' && \text{Rewrite given}\\
V &\in [\![P]\!]_\delta && \text{Rewrite given similarly}\\
V' &\in [\![P]\!]_\delta && \text{By i.h.}\\
&= [\![\underline{P}]\!]_\delta\,([\![F]\!]_\delta^k\,\emptyset) && \text{By def.}
\end{aligned}$$

- **Case** $\mathscr{F} = \hat{B} \otimes \hat{P}$:

$$\llbracket |\hat{B}| \otimes |\hat{P}| \rrbracket\,(\llbracket |F| \rrbracket^k \emptyset) = \llbracket |\hat{B}| \rrbracket\,(\llbracket |F| \rrbracket^k \emptyset) \times \llbracket |\hat{P}| \rrbracket\,(\llbracket |F| \rrbracket^k \emptyset) \quad \text{By def.}$$

$$V \sqsubseteq_{\llbracket |\hat{B}| \rrbracket\,(\llbracket |F| \rrbracket^k \emptyset) \times \llbracket |\hat{P}| \rrbracket\,(\llbracket |F| \rrbracket^k \emptyset)} V' \quad \text{Rewrite given}$$

$$V \in \llbracket \hat{B} \rrbracket_\delta\,(\llbracket F \rrbracket_\delta^k \emptyset) \times \llbracket \hat{P} \rrbracket_\delta\,(\llbracket F \rrbracket_\delta^k \emptyset) \quad \text{Rewrite given (similarly)}$$

$$V = (V_1, V_2) \quad \text{By inversion}$$

$$V_1 \in \llbracket \hat{B} \rrbracket_\delta\,(\llbracket F \rrbracket_\delta^k \emptyset) \quad ''$$

$$V_2 \in \llbracket \hat{P} \rrbracket_\delta\,(\llbracket F \rrbracket_\delta^k \emptyset) \quad ''$$

$$V' \in \llbracket |\hat{B}| \rrbracket\,(\llbracket |F| \rrbracket^k \emptyset) \times \llbracket |\hat{P}| \rrbracket\,(\llbracket |F| \rrbracket^k \emptyset) \quad \text{By def. of } \sqsubseteq_\times$$

$$V' = (V_1', V_2') \quad ''$$

$$V_1' \in \llbracket |\hat{B}| \rrbracket\,(\llbracket |F| \rrbracket^k \emptyset) \quad ''$$

$$V_2' \in \llbracket |\hat{P}| \rrbracket\,(\llbracket |F| \rrbracket^k \emptyset) \quad ''$$

$$V_1 \sqsubseteq_{\llbracket |\hat{B}| \rrbracket\,(\llbracket |F| \rrbracket^k \emptyset)} V_1' \quad ''$$

$$V_2 \sqsubseteq_{\llbracket |\hat{P}| \rrbracket\,(\llbracket |F| \rrbracket^k \emptyset)} V_2' \quad ''$$

$$V_1' \in \llbracket \hat{B} \rrbracket_\delta\,(\llbracket F \rrbracket_\delta^k \emptyset) \quad \text{By i.h.}$$

$$V_2' \in \llbracket \hat{P} \rrbracket_\delta\,(\llbracket F \rrbracket_\delta^k \emptyset) \quad \text{By i.h.}$$

$$V' = (V_1', V_2') \quad \text{Above}$$

$$\in \llbracket \hat{B} \rrbracket_\delta\,(\llbracket F \rrbracket_\delta^k \emptyset) \times \llbracket \hat{P} \rrbracket_\delta\,(\llbracket F \rrbracket_\delta^k \emptyset) \quad \text{By set theory}$$

$$= \llbracket \hat{B} \otimes \hat{P} \rrbracket_\delta\,(\llbracket F \rrbracket_\delta^k \emptyset) \quad \text{By def.}$$

- **Case** $\mathscr{F} = F_1 \oplus F_2$: Similar to $\mathscr{F} = \hat{B} \otimes \hat{P}$ case.

(3) We case analyze A.

- **Case** $A = \{\nu : \mu F \mid \mathscr{M}(F)\}$:

  $V \in [\![\{\nu : \mu F \mid \mathscr{M}(F)\}]\!]_\delta$ — Given

  $= \{V \in \mu\,[\![|F|]\!] \mid V \in \mu\,[\![F]\!]_\delta \text{ and } [\![\mathscr{M}(F)]\!]_\delta V = \{\bullet\}\}$ — By def.

  $V \in \mu\,[\![F]\!]_\delta$ — By set theory

  By def., $[\![\boldsymbol{t}]\!]_\delta\,((\mathit{fold}_{[\![F]\!]_\delta}\,[\![\alpha]\!]_\delta)\,V) = [\![t]\!]_\delta$ for all $(\mathsf{fold}_F\ \alpha)\,\nu\,\boldsymbol{t} =_\tau t \in \mathscr{M}(F)$.

  $V \sqsubseteq_{[\![\mu|F|]\!]} V'$ — Given

  $V \sqsubseteq_{[\![|F|]\!]([\![|F|]\!]^n\emptyset)} V'$ — By def. of $\sqsubseteq$, there exists such an n

  $V \in [\![|F|]\!]\,([\![|F|]\!]^n\,\emptyset)$ — $''$

  $V \in \mu\,[\![F]\!]_\delta$ — Above

  $= [\![F]\!]_\delta\,(\mu\,[\![F]\!]_\delta)$ — By Lemma E.10 (Ref. Equal Functor Mu)

  $V \in [\![F]\!]_\delta\,([\![F]\!]_\delta^n\,\emptyset)$ — By Lemma E.25

  $V' \in [\![F]\!]_\delta\,([\![F]\!]_\delta^n\,\emptyset)$ — By i.h.

  $\subseteq \cup_{k\in\mathbb{N}}\,[\![F]\!]_\delta^k\,\emptyset$ — By set theory

  $= \mu\,[\![F]\!]_\delta$ — By def.

  $\subseteq \mu\,[\![|F|]\!]$ — By Lemma D.26 (Refinement Subset of Erasure) …

  … and def. of μ

  $\Xi \vdash \{\nu : \mu F \mid \mathscr{M}(F)\}\ \mathsf{type}[_]$ — Given

  By i.h. with algebra formations subderivations,

  for all $(\mathsf{fold}_F\ \alpha)\,\nu\,\boldsymbol{t} =_\tau t \in \mathscr{M}(F)$, we have the following:

$$
\begin{aligned}
\llbracket \alpha \rrbracket_\delta \text{ monotone} &\qquad \text{By i.h.}\\
\llbracket \alpha \rrbracket_\delta : \llbracket F \rrbracket_\delta \, \llbracket \tau \rrbracket \to \llbracket \tau \rrbracket &\qquad \text{By Lemma E.7 (Type WF Sound)}\\
(\mathit{fold}_{\llbracket F \rrbracket_\delta} \, \llbracket \alpha \rrbracket_\delta) \text{ monotone} &\qquad \text{By Lemma E.24 (Fold Continuous)}\\
V \sqsubseteq_{\llbracket \mu F \rrbracket} V' &\qquad \text{Above}\\
(\mathit{fold}_{\llbracket F \rrbracket_\delta} \, \llbracket \alpha \rrbracket_\delta) \, V \sqsubseteq_{\llbracket \tau \rrbracket} (\mathit{fold}_{\llbracket F \rrbracket_\delta} \, \llbracket \alpha \rrbracket_\delta) \, V' &\qquad \text{By def. of monotone}\\
\llbracket t \rrbracket_\delta = \llbracket \boldsymbol{t} \rrbracket_\delta \, ((\mathit{fold}_{\llbracket F \rrbracket_\delta} \, \llbracket \alpha \rrbracket_\delta) \, V) &\qquad \text{Above}\\
= \llbracket \boldsymbol{t} \rrbracket_\delta \, ((\mathit{fold}_{\llbracket F \rrbracket_\delta} \, \llbracket \alpha \rrbracket_\delta) \, V') &\qquad \llbracket \tau \rrbracket \text{ has discrete order}
\end{aligned}
$$

$$V' \in \llbracket \{ \nu : \mu F \mid \mathscr{M}(F) \} \rrbracket_\delta \qquad \text{Follows from above}$$

- The remaining cases are straightforward. □

Appendix E.1 Refined Type and Substitution Soundness

Theorem E.1 (Program Typing Soundness)**.** *Assume* $\vdash \delta : \Theta; \Gamma$.

(1) If $\Theta; \Gamma \vdash h \Rightarrow P$ *then* $\llbracket h \rrbracket_\delta \in \llbracket P \rrbracket_{\lfloor \delta \rfloor}$.

(2) If $\Theta; \Gamma \vdash g \Rightarrow \uparrow P$ *then* $\llbracket g \rrbracket_\delta \in \llbracket \uparrow P \rrbracket_{\lfloor \delta \rfloor}$.

(3) If $\Theta; \Gamma \vdash v \Leftarrow P$ *then* $\llbracket v \rrbracket_\delta \in \llbracket P \rrbracket_{\lfloor \delta \rfloor}$.

(4) If $\Theta; \Gamma \vdash e \Leftarrow N$ *then* $\llbracket e \rrbracket_\delta \in \llbracket N \rrbracket_{\lfloor \delta \rfloor}$.

(5) If $\Theta; \Gamma; [P] \vdash \{ r_i \Rightarrow e_i \}_{i \in I} \Leftarrow N$ *then* $\llbracket \{ r_i \Rightarrow e_i \}_{i \in I} \rrbracket_\delta \in \llbracket P \rrbracket_{\lfloor \delta \rfloor} \Rightarrow \llbracket N \rrbracket_{\lfloor \delta \rfloor}$.

(6) If $\Theta; \Gamma; [N] \vdash s \Rightarrow \uparrow P$ *then* $\llbracket s \rrbracket_\delta \in \llbracket N \rrbracket_{\lfloor \delta \rfloor} \Rightarrow \llbracket \uparrow P \rrbracket_{\lfloor \delta \rfloor}$.

Proof. By mutual induction on the structure of the given typing derivation. Note that, throughout the proof, we implicitly use Lemma E.4 (Filter Out Program Vars), Lemma D.32 (Erasure and Substitution Commute), and Lemma C.2 (Filter Out Propositions). By Lemma D.23 (Unrefined Typing Soundness), we only need to show that the denotation of the program term is in the refined set (as stated in the theorem).

(1) • **Case**

$$\frac{(x : R) \in \Gamma}{\Theta; \Gamma \vdash x \Rightarrow R}\ \text{Decl}{\Rightarrow}\text{Var}$$

$$\begin{aligned}
[\![x]\!]_\delta &= [\![|x|]\!]_{|\delta|} && \text{By def. of } [\![-]\!] \\
&= [\![x]\!]_{|\delta|} && \text{By def. of } |-| \\
&= (|\delta|)(x) && \text{By def. of } [\![-]\!] \\
&= \delta(x) && \text{By def. of } |-| \\
&\in [\![R]\!]_{\lfloor \delta \rfloor} && \text{By inversion on } \vdash \delta : \Theta; \Gamma \text{ and } (x : R) \in \Gamma, \\
& && \quad \text{using Lemma E.5 } ([\![-]\!] \text{ Weakening Invariant}) \text{ if needed}
\end{aligned}$$

• **Case**

$$\frac{\overline{\Theta} \vdash P\ \mathsf{type}[\xi] \qquad \Theta; \Gamma \vdash v \Leftarrow P}{\Theta; \Gamma \vdash (v : P) \Rightarrow P}\ \text{Decl}{\Rightarrow}\text{ValAnnot}$$

$$\begin{array}{rll}
\vdash \delta : \Theta;\Gamma & & \text{Given} \\
\Theta;\Gamma \vdash v \Leftarrow P & & \text{Subderivation} \\
\\
[\![(v : P)]\!]_\delta & = [\![|(v : P)|]\!]_{|\delta|} & \text{By def. of } [\![-]\!] \\
& = [\![(|v| : |P|)]\!]_{|\delta|} & \text{By def. of } |-| \\
& = [\![|v|]\!]_{|\delta|} & \text{By def. of } [\![-]\!] \\
& = [\![v]\!]_\delta & \text{By def. of } [\![-]\!] \\
& \in [\![P]\!]_{\lfloor\delta\rfloor} & \text{By i.h.}
\end{array}$$

(2) • **Case**

$$\dfrac{\Theta;\Gamma \vdash h \Rightarrow \downarrow N \qquad \Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P}{\Theta;\Gamma \vdash h(s) \Rightarrow \uparrow P}\ \text{Decl}\Rightarrow\text{App}$$

$$
\begin{array}{rll}
\vdash \delta : \Theta;\Gamma & & \text{Given} \\
\Theta;\Gamma \vdash h \Rightarrow \downarrow N & & \text{Subderivation} \\
[\![h]\!]_\delta \in [\![\downarrow N]\!]_{\lfloor\delta\rfloor} & & \text{By i.h.} \\
= [\![N]\!]_{\lfloor\delta\rfloor} & & \text{By def. of } [\![-]\!] \\
\Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P & & \text{Subderivation} \\
[\![s]\!]_\delta \in [\![N]\!]_{\lfloor\delta\rfloor} \Rightarrow [\![\uparrow P]\!]_{\lfloor\delta\rfloor} & & \text{By i.h.}
\end{array}
$$

$$
\begin{array}{rll}
[\![h(s)]\!]_\delta = [\![|h(s)|]\!]_{|\delta|} & & \text{By def. of } [\![-]\!] \\
= [\![|h|(|s|)]\!]_{|\delta|} & & \text{By def. of } |-| \\
= [\![|s|]\!]_{|\delta|}\,[\![|h|]\!]_{|\delta|} & & \text{By def. of } [\![-]\!] \\
= [\![s]\!]_\delta\,[\![h]\!]_\delta & & \text{By def. of } [\![-]\!] \\
\in [\![\uparrow P]\!]_{\lfloor\delta\rfloor} & & \text{Function application}
\end{array}
$$

- **Case**

$$\frac{\overline{\Theta} \vdash P\ \mathsf{type}[\xi] \qquad \Theta;\Gamma \vdash e \Leftarrow \uparrow P}{\Theta;\Gamma \vdash (e : \uparrow P) \Rightarrow \uparrow P}\ \text{Decl}\Rightarrow\text{ExpAnnot}$$

  Similar to the Decl⇒ValAnnot case of part (1).

(3) • **Case**

$$\frac{(x : R') \in \Gamma \qquad \Theta \vdash R' \leq^{+} R}{\Theta;\Gamma \vdash x \Leftarrow R}\ \text{Decl}\Leftarrow\text{Var}$$

$$
\begin{aligned}
& \vdash \delta : \Theta; \Gamma && \text{Given} \\
[\![x]\!]_\delta &= [\![|x|]\!]_{|\delta|} && \text{By def. of } [\![-]\!] \\
&= [\![x]\!]_{|\delta|} && \text{By def. of } |-| \\
&= (|\delta|)(x) && \text{By def. of } [\![-]\!] \\
&= \delta(x) && \text{By def. of } |-| \\
&\in [\![R']\!]_{\lfloor \delta \rfloor} && \text{By inversion on } \vdash \delta : \Theta; \Gamma \text{ and premise } (x : R') \in \Gamma, \\
& && \quad \text{using Lemma E.5 } ([\![-]\!] \text{ Weakening Invariant}) \text{ if needed} \\
&\subseteq [\![R']\!]_{\lfloor \delta \rfloor} && \text{By Lemma E.23 (Subtyping Soundness)} \\
& && \quad \text{with premise } \Theta \vdash R' \leq^+ R
\end{aligned}
$$

- **Case** Decl$\Leftarrow$1: Straightforward.
- **Case** Decl$\Leftarrow\times$: Straightforward.
- **Case** Decl$\Leftarrow+_k$: Straightforward.
- **Case**

$$
\frac{\Theta; \Gamma \vdash v \Leftarrow [\overrightarrow{t}/^{\mathsf{d}}\Xi']Q \qquad \overset{\mathsf{d}}{\div}\Theta \vdash \overrightarrow{t}/^{\mathsf{d}}\Xi' : {}^{\mathsf{d}}\Xi'}{\Theta; \Gamma \vdash v \Leftarrow (\exists^{\mathsf{d}}\Xi'.Q)}\ \text{Decl}\Leftarrow\exists
$$

$$\begin{array}{ll}
\overset{\mathsf{d}}{\div}\Theta \vdash \overrightarrow{t}/^{\mathsf{d}}\Xi' : {}^{\mathsf{d}}\Xi' & \text{Premise} \\
\text{Let } \sigma = \overrightarrow{t}/^{\mathsf{d}}\Xi'. & \\
[\![\sigma]\!]_{\lfloor\delta\rfloor} \in \left[\!\!\left[{}^{\mathsf{d}}\Xi'\right]\!\!\right] & \text{By repeated Lemma C.21}
\end{array}$$

$$\begin{array}{ll}
[\![v]\!]_\delta = [\![|v|]\!]_{|\delta|} & \text{By def. of } [\![-]\!] \\
= [\![\Theta;\Gamma \vdash v \Leftarrow [\sigma]Q]\!]_\delta & \text{By def. of } [\![-]\!] \\
\in [\![[\sigma]Q]\!]_{\lfloor\delta\rfloor} & \text{By i.h.} \\
= [\![[id_\Theta, \sigma]Q]\!]_{\lfloor\delta\rfloor} & \text{By Lemma C.81} \\
= [\![Q]\!]_{[\![id_\Theta,\sigma]\!]_{\lfloor\delta\rfloor}} & \text{By Lemma E.18} \\
= [\![Q]\!]_{\lfloor\delta\rfloor,[\![\sigma]\!]_{\lfloor\delta\rfloor}} & \text{By def. of } [\![-]\!] \\
\subseteq \left\{V \in [\![|Q|]\!] \;\middle|\; \exists \delta' \in \left[\!\!\left[{}^{\mathsf{d}}\Xi'\right]\!\!\right].\, V \in [\![Q]\!]_{\lfloor\delta\rfloor,\delta'}\right\} & \text{By Lemma D.26} \\
& \quad \text{(and } [\![\sigma]\!]_{\lfloor\delta\rfloor} \in \left[\!\!\left[{}^{\mathsf{d}}\Xi'\right]\!\!\right]\text{)} \\
= \left[\!\!\left[\exists^{\mathsf{d}}\Xi'.\, Q\right]\!\!\right]_{\lfloor\delta\rfloor} & \text{By def. of } [\![-]\!]
\end{array}$$

- **Case** Decl⇐∧: Similar to case for Decl⇐∃.
- **Case**

$$\frac{\nexists x.\, v_0 = \overrightarrow{\mathrm{inj}_{k_i}}^{i}\left(\overrightarrow{\langle __j, - \rangle}^{j} x\right) \qquad \mathscr{M}(F) \rightsquigarrow \overrightarrow{\alpha}; \overrightarrow{\tau} \qquad \overset{\mathsf{d}}{\div}\Theta \vdash \wr \overrightarrow{\alpha}; F; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta; R \qquad \Theta;\Gamma \vdash v_0 \Leftarrow \exists^{\mathsf{d}}\Theta.\, R \wedge {}^{\mathsf{d}}\Theta}{\Theta;\Gamma \vdash \mathtt{into}(v_0) \Leftarrow \{\nu : \mu F \mid \mathscr{M}(F)\}}\ \mathrm{Decl}{\Leftarrow}\mu$$

  By definition of $[\![-]\!]$ and $|-|$, by the i.h., and by Lemma E.20 (Unrolling Soundness), Lemma E.11 (Semantic Unroll), and Lemma D.26 (Refinement Subset of Erasure).
- **Case** Decl⇐↓: Straightforward.

(4)

- **Case** Decl⇐↑: Straightforward.
- **Case**

$$\frac{\Theta;\Gamma \vdash g \Rightarrow \uparrow(\exists^{\mathsf{d}}\Xi.R \wedge \overrightarrow{\psi}) \qquad \Theta,{}^{\mathsf{d}}\Xi,\overrightarrow{\psi};\Gamma,x:R \vdash e_0 \Leftarrow L}{\Theta;\Gamma \vdash \mathsf{let}\ x = g;\ e_0 \Leftarrow L}\ \mathrm{Decl}{\Leftarrow}\mathsf{let}$$

$\Theta;\Gamma \vdash g \Rightarrow \uparrow(\exists^{\mathsf{d}}\Xi.R \wedge \overrightarrow{\psi})$ Subderivation

$[\![g]\!]_\delta \in [\![\uparrow(\exists^{\mathsf{d}}\Xi.R \wedge \overrightarrow{\psi})]\!]_{\lfloor\delta\rfloor}$ By i.h.

$= \left\{(1,V) \;\middle|\; V \in [\![\exists^{\mathsf{d}}\Xi.R \wedge \overrightarrow{\psi}]\!]_{\lfloor\delta\rfloor}\right\}$ By def. of $[\![-]\!]$

Therefore, there exist $\delta' \in \left[\!\left[{}^{\mathsf{d}}\Xi'\right]\!\right]$ and V such that $V \in [\![R]\!]_{\lfloor\delta\rfloor,\delta'}$ and $[\![\psi]\!]_{\lfloor\delta\rfloor,\delta'} = \{\bullet\}$ for all $\psi \in \overrightarrow{\psi}$ and $[\![g]\!]_\delta = (1,V)$. By definition, $[\![g]\!]_\delta = [\![|g|]\!]_{|\delta|}$. By transitivity, $[\![|g|]\!]_{|\delta|} = (1,V)$.

$\vdash \delta : \Theta;\Gamma$ Given

$\vdash \delta,\delta',V/x : \Theta,{}^{\mathsf{d}}\Xi';\Gamma,x:R$ By rules

$\Theta,{}^{\mathsf{d}}\Xi',\overrightarrow{\psi};\Gamma,x:R \vdash e_0 \Leftarrow L$ Subderivation

$$\begin{aligned}
[\![\mathsf{let}\ x=g;\ e_0]\!]_\delta &= [\![|\mathsf{let}\ x=g;\ e_0|]\!]_{|\delta|} && \text{By def. of } [\![-]\!]\\
&= [\![\mathsf{let}\ x=|g|;\ |e_0|]\!]_{|\delta|} && \text{By def. of } |-|\\
&= [\![|e_0|]\!]_{|\delta|,V/x} && \text{By def. of } [\![-]\!]\\
&= [\![|e_0|]\!]_{|\delta,\delta',V/x|} && \text{By def. of } |-|\\
&= [\![e_0]\!]_{\delta,\delta',V/x} && \text{By def. of } [\![-]\!]\\
&\in [\![L]\!]_{\lfloor\delta,\delta',V/x\rfloor} && \text{By i.h.}\\
&= [\![L]\!]_{\lfloor\delta\rfloor,\delta'} && \text{By def. of } \lfloor-\rfloor\\
&= [\![L]\!]_{\lfloor\delta\rfloor} && FV(L)\cap dom({}^{\mathsf{d}}\Xi') = \emptyset
\end{aligned}$$

- **Case** Decl⇐match: Straightforward.

- **Case** Decl⇐λ: Similar to Decl⇐let case, but simpler.

- **Case**

$$\frac{\Theta \vdash \forall a \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, M \leq^{-} L \qquad \Theta, a \div \mathbb{N}; \Gamma, x : \downarrow \forall a' \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, a' < a \supset [a'/a]M \vdash e_0 \Leftarrow \forall^{\mathsf{d}}\Xi.\, M}{\Theta; \Gamma \vdash \mathsf{rec}\, x : (\forall a \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, M).\, e_0 \Leftarrow L}\ \text{Decl}\Leftarrow\mathsf{rec}$$

For each $k \in \mathbb{N}$, define $X_k = \left[\!\left[\downarrow \forall a' \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, a' < a \supset [a'/a]M\right]\!\right]_{\lfloor\delta\rfloor, k/a}$.

For all $k \in \mathbb{N}$,

$$\begin{aligned}
X_k &= \left[\!\left[\downarrow \forall a' \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, a' < a \supset [a'/a]M\right]\!\right]_{\lfloor\delta\rfloor, k/a} && \text{By } X_k \text{ def.} \\
&= \left[\!\left[\forall a' \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, a' < a \supset [a'/a]M\right]\!\right]_{\lfloor\delta\rfloor, k/a} && \text{By } [\![-]\!] \text{ def.} \\
&= \left\{ f \in [\![|[a'/a]M|]\!] \;\middle|\; \forall n < k, \delta' \in [\![{}^{\mathsf{d}}\Xi]\!].\, f \in [\![[a'/a]M]\!]_{\lfloor\delta\rfloor, k/a, n/a', \delta'} \right\} && \text{By } [\![-]\!] \text{ def.} \\
&= \left\{ f \in [\![|M|]\!] \;\middle|\; \forall n < k, \delta' \in [\![{}^{\mathsf{d}}\Xi]\!].\, f \in [\![[a'/a]M]\!]_{\lfloor\delta\rfloor, k/a, n/a', \delta'} \right\} && \text{Lemma D.28} \\
&= \left\{ f \in [\![|M|]\!] \;\middle|\; \forall n < k, \delta' \in [\![{}^{\mathsf{d}}\Xi]\!].\, f \in [\![[a'/a]M]\!]_{\lfloor\delta\rfloor, n/a', \delta'} \right\} && \text{Lemma E.5} \\
&= \left\{ f \in [\![|M|]\!] \;\middle|\; \forall n < k, \delta' \in [\![{}^{\mathsf{d}}\Xi]\!].\, f \in [\![M]\!]_{\lfloor\delta\rfloor, n/a, \delta'} \right\} && \text{Lemma E.18}
\end{aligned}$$

Define $g : [\![|M|]\!] \to [\![|M|]\!]$ by $V \mapsto [\![|e_0|]\!]_{|\delta|, V/x}$ (well-defined by Lemma D.23 (Unrefined Typing Soundness)). By Lemma D.22 (Continuous Maps), g is continuous (i.e., g is monotone and respects lubs).

For each $k \in \mathbb{N}$, define g_k as follows:

$$g_0 = \bot_{[\![|M|]\!]}$$

$$g_{j+1} = g(g_j)$$

Because g_0 is the bottom element of $[\![|M|]\!]$ and $g : [\![|M|]\!] \to [\![|M|]\!]$ is monotone, we know $g_k \sqsubseteq_{[\![|M|]\!]} g_{k+1}$ for all $k \in \mathbb{N}$.

By definition of $[\![-]\!]$ and $|-|$,

$$\left[\!\left[\mathtt{rec}\, x : (\forall a \overset{\mathsf{d}\cdot}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.\, M).\, e_0\right]\!\right]_{\delta} = [\![\mathtt{rec}\, x.\, |e_0|]\!]_{|\delta|} = \bigsqcup_{k\in\mathbb{N}} g_k$$

Therefore, to complete this case, it suffices to show that $\sqcup_{k\in\mathbb{N}} g_k \in [\![L]\!]_{\lfloor\delta\rfloor}$.

To this end, we show that $g_k \in X_k$ for all $k \in \mathbb{N}$ by induction on k. Now, $g_0 = \bot_{[\![|M|]\!]} \in [\![|M|]\!]$ because $[\![|M|]\!] \in \mathbf{Cppo}$ by Lemma D.15 (Unref. Type Denotations). For the inductive step, we assume $g_m \in X_m$ and will prove $g_{m+1} \in X_{m+1}$. Now,

$g_{m+1} = g(g_m)$		By def. of g_k
$= [\![\lvert e_0\rvert]\!]_{\lvert\delta\rvert, g_m/x}$		By def. of g
$= [\![e_0]\!]_{\delta, m/a, g_m/x}$		By def. of $[\![-]\!]$
$\in \left[\!\left[\forall^{\mathsf{d}}\Xi.\, M\right]\!\right]_{\lfloor\delta\rfloor, m/a}$		By i.h. and def. of $\lfloor-\rfloor$
$\subseteq [\![\lvert M\rvert]\!]$		By Lemma D.26 (Refinement Subset of Erasure)

$g_m \sqsubseteq_{[\![\lvert M\rvert]\!]} g_{m+1}$	Above
$g_m \in X_m$	Above
$g_{m+1} \in X_m$	By Lemma E.26 (Upward Closure)

$$
\begin{aligned}
g_{m+1} &\in X_m \cap \left[\!\left[\forall^{\mathsf{d}}\Xi.M\right]\!\right]_{\lfloor\delta\rfloor,m/a} && \text{Set theory}\\
&= \left\{ f \in [\![|M|]\!] \;\middle|\; \forall n < m, \delta' \in {}^{\mathsf{d}}\Xi. f \in [\![M]\!]_{\lfloor\delta\rfloor,n/a,\delta'} \right\} \cap \left[\!\left[\forall^{\mathsf{d}}\Xi.M\right]\!\right]_{\lfloor\delta\rfloor,m/a} && \text{Above}\\
&= \left\{ f \in [\![|M|]\!] \;\middle|\; \forall n < m+1, \delta' \in \left[\!\left[{}^{\mathsf{d}}\Xi\right]\!\right]. f \in [\![M]\!]_{\lfloor\delta\rfloor,n/a,\delta'} \right\} && \text{Set theory}\\
& && \text{and } [\![-]\!] \text{ def.}\\
&= X_{m+1} && \text{Above}
\end{aligned}
$$

We have just proved that $g_k \in X_k$ for all $k \in \mathbb{N}$.

Because $[\![|M|]\!] \in \mathbf{Cppo}$, it is chain-complete. But $g_0 \sqsubseteq g_1 \sqsubseteq \cdots$ is a chain in $[\![|M|]\!]$, so it has a lub $\sqcup_{k\in\mathbb{N}} g_k$ in $[\![|M|]\!]$ (by chain-completeness).

Next, we show that $\sqcup_{k\in\mathbb{N}} g_k \in \left[\!\left[\forall^{\mathsf{d}}\Xi.M\right]\!\right]_{\lfloor\delta\rfloor,j/a}$ for all $j \in \mathbb{N}$. Suppose $j \in \mathbb{N}$. Then:

$$
\begin{aligned}
g_{j+1} &\sqsubseteq_{[\![|M|]\!]} \sqcup_{k\in\mathbb{N}} g_k && \text{By def. of lub}\\
g_{j+1} &\in X_{j+1} && \text{Above}\\
&= \left\{ f \in [\![|M|]\!] \;\middle|\; \forall n < j+1, \delta' \in \left[\!\left[{}^{\mathsf{d}}\Xi\right]\!\right]. f \in [\![M]\!]_{\lfloor\delta\rfloor,n/a,\delta'} \right\} && \text{Above}\\
&\subseteq \left[\!\left[\forall^{\mathsf{d}}\Xi.M\right]\!\right]_{\lfloor\delta\rfloor,j/a} && \text{Straightforward}
\end{aligned}
$$

$\sqcup_{k\in\mathbb{N}} g_k \in \left[\!\left[\forall^{\mathsf{d}}\Xi.M\right]\!\right]_{\lfloor\delta\rfloor,j/a}$ By Lemma E.26 (Upward Closure)

Therefore, $\sqcup_{k\in\mathbb{N}} g_k \in \left[\!\left[\forall^{\mathsf{d}}\Xi.M\right]\!\right]_{\lfloor\delta\rfloor,j/a}$ for all $j \in \mathbb{N}$, and we have:

$$
\begin{aligned}
\sqcup_{k\in\mathbb{N}} g_k &\in \textstyle\bigcap_{j\in\mathbb{N}} \left[\!\left[\forall^{\mathsf{d}}\Xi.M\right]\!\right]_{\lfloor\delta\rfloor,j/a} && \text{By def. of intersection}\\
&= \left[\!\left[\forall a \overset{\mathsf{d}}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.M\right]\!\right]_{\lfloor\delta\rfloor} && \text{By def. of } [\![-]\!], \text{ set theory, and Lemma D.26}\\
&\subseteq [\![L]\!]_{\lfloor\delta\rfloor} && \text{By Lemma E.23 (Subtyping Soundness)}\\
& && \text{(with premise } \Theta \vdash \forall a \overset{\mathsf{d}}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.M \leq^{-} L)
\end{aligned}
$$

That completes this case.

- **Case**
$$\frac{\Theta, {}^{\mathsf{d}}\Xi; \Gamma \vdash e \Leftarrow M}{\Theta; \Gamma \vdash e \Leftarrow \forall^{\mathsf{d}}\Xi.M}\ \text{Decl}{\Leftarrow}\forall$$
Assume $\delta' \in [\![{}^{\mathsf{d}}\Xi]\!]$.
$$[\![e]\!]_\delta = [\![e]\!]_{\delta,\delta'} \qquad FV(e) \subseteq dom(\Theta;\Gamma)$$
$$\in [\![M]\!]_{\delta,\delta'} \qquad \text{By i.h.}$$
$$[\![e]\!]_\delta \in [\![\forall^{\mathsf{d}}\Xi.M]\!]_\delta \quad \text{By def. of } [\![-]\!]$$

- **Case** Decl$\Leftarrow\supset$: Similar to Decl$\Leftarrow\forall$ case.

- **Case** Decl$\Leftarrow$Unreachable: Straightforward. Use Lemma D.15 (Unref. Type Denotations).

(5)

- **Case** DeclMatch$\exists$: Similar to Decl$\Leftarrow\forall$ case of part (4).

- **Case** DeclMatch$\wedge$: Similar to DeclMatch$\exists$ case.

- **Case** DeclMatchμ: Similar to Decl$\Leftarrow\mu$ case of part (3).

- The remaining cases are straightforward or similar to previous cases.

(6)

- **Case** DeclSpine$\forall$: Similar to case for dual rule Decl$\Leftarrow\exists$ of part (3).

- **Cases** DeclSpine$\supset$: Similar to DeclSpine$\forall$ case.

- **Case** DeclSpineApp: Straightforward.

- **Case** DeclSpineNil: Straightforward. □

Lemma E.27 ($\lfloor - \rfloor$ and $[\![-]\!]$ Commute)**.** *If* $\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma$ *and* $\vdash \delta : \Theta_0; \Gamma_0$ *then* $\lfloor [\![\sigma]\!]_\delta \rfloor = [\![\lfloor \sigma \rfloor]\!]_{\lfloor \delta \rfloor}$.

Proof. By structural induction on $\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma$. □

Lemma E.28 (Substitution Typing Soundness)**.**

If $\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma$ *then* $\vdash [\![\sigma]\!]_\delta : \Theta; \Gamma$ *for all* $\vdash \delta : \Theta_0; \Gamma_0$,

Proof. By structural induction on the given substitution typing derivation. Consider cases for the rule concluding it:

- **Case**

$$\dfrac{}{\Theta_0; \Gamma_0 \vdash \cdot : \cdot ; \cdot}\ \text{SubstEmpty}$$

$\vdash \cdot : \cdot ; \cdot$ By Emptyδ

- **Case**

$$\dfrac{\Theta_0; \Gamma_0 \vdash \sigma' : \Theta'; \Gamma \qquad \overline{\Theta_0} \vdash t : \tau \qquad a \notin dom(\Theta')}{\Theta_0; \Gamma_0 \vdash \sigma', t/a : \Theta', a \div \tau; \Gamma}\ \text{SubstIx}$$

$\Theta_0; \Gamma_0 \vdash \sigma' : \Theta'; \Gamma$	Subderivation
$\vdash [\![\sigma']\!]_\delta : \Theta'; \Gamma$	By i.h.
$\overline{\Theta_0} \vdash t : \tau$	Premise
$[\![t]\!]_{\lfloor \delta \rfloor} \in [\![\tau]\!]$	By Lemma C.21
$\vdash \underbrace{[\![\sigma']\!]_\delta, [\![t]\!]_{\lfloor \delta \rfloor}/a}_{[\![\sigma]\!]_\delta} : \Theta', a \div \tau; \Gamma$	By Ixδ and Def. C.2

- **Case** SubstIxDet: Similar to SubstIx case.

- **Case**

$$\frac{\Theta_0;\Gamma_0 \vdash \sigma : \Theta';\Gamma \qquad \Theta_0 \vdash [\lfloor\sigma\rfloor]\varphi\ \mathsf{true}}{\Theta_0;\Gamma_0 \vdash \sigma : \Theta',\varphi;\Gamma}\ \text{SubstProp}$$

$$\begin{array}{rll}
\Theta_0;\Gamma_0 \vdash \sigma : \Theta';\Gamma & & \text{Subderivation}\\
\vdash [\![\sigma]\!]_\delta : \Theta';\Gamma & & \text{By i.h.}\\
\vdash \lfloor\delta\rfloor : \Theta_0 & & \text{By Lemma E.4 (Filter Out Program Vars)}\\
\Theta_0 \vdash \lfloor\sigma\rfloor : \Theta' & & \text{By Lemma C.1 (Filter Out Prog. Vars. Syn)}\\
\Theta_0 \vdash [\lfloor\sigma\rfloor]\varphi\ \mathsf{true} & & \text{Premise}\\
\{\bullet\} = [\![[\lfloor\sigma\rfloor]\varphi]\!]_{\lfloor\delta\rfloor} & & \text{By inversion on PropTrue}\\
= [\![\varphi]\!]_{[\![\lfloor\sigma\rfloor]\!]_{\lfloor\delta\rfloor}} & & \text{By Lemma C.24 (Index Substitution Soundness)}\\
= [\![\varphi]\!]_{\lfloor[\![\sigma]\!]_\delta\rfloor} & & \text{By Lemma E.27 ($\lfloor-\rfloor$ and $[\![-]\!]$ Commute)}\\
\vdash [\![\sigma]\!]_\delta : \Theta',\varphi;\Gamma & & \text{By Prop}\delta
\end{array}$$

- **Case**

$$\frac{\Theta_0;\Gamma_0 \vdash \sigma' : \Theta;\Gamma' \qquad \Theta_0;\Gamma_0 \vdash v \Leftarrow [\lfloor\sigma'\rfloor]R \qquad x \notin dom(\Gamma')}{\Theta_0;\Gamma_0 \vdash \sigma',v : [\lfloor\sigma'\rfloor]R/x : \Theta;\Gamma',x : R}\ \text{SubstVal}$$

$$\begin{array}{rll}
\Theta_0;\Gamma_0 \vdash \sigma' : \Theta;\Gamma' & & \text{Subderivation}\\
\vdash [\![\sigma']\!]_\delta : \Theta;\Gamma' & & \text{By i.h.}\\
[\![v]\!]_\delta \in [\![[\lfloor\sigma'\rfloor]R]\!]_{\lfloor\delta\rfloor} & & \text{By Theorem E.1}\\
= [\![R]\!]_{[\![\lfloor\sigma'\rfloor]\!]_{\lfloor\delta\rfloor}} & & \text{By Theorem E.18}\\
= [\![R]\!]_{\lfloor[\![\sigma']\!]_\delta\rfloor} & & \text{By Lemma E.27 ($\lfloor-\rfloor$ and $[\![-]\!]$ Commute)}\\
\vdash [\![\sigma']\!]_\delta, [\![v]\!]_\delta /x : \Theta;\Gamma',x : R & & \text{By Val}\delta\\
\vdash [\![\sigma',v/x]\!]_\delta : \Theta;\Gamma',x : R & & \text{By def. of } [\![-]\!]
\end{array}$$

□

Theorem E.2 (Soundness of Substitution)**.**

Assume $\vdash \delta : \Theta_0; \Gamma_0$ *and* $\Theta_0; \Gamma_0 \vdash \sigma : \Theta; \Gamma$.

(1) If $\Theta; \Gamma \vdash h \Rightarrow P$

then for all Q *such that* $\Theta_0 \vdash Q \leq^+ [\lfloor\sigma\rfloor]P$ *and* $\Theta_0; \Gamma_0 \vdash [\sigma]h \Rightarrow Q$

we have $[\![\Theta_0; \Gamma_0 \vdash [\sigma]h \Rightarrow Q]\!]_\delta = [\![\Theta; \Gamma \vdash h \Rightarrow P]\!]_{[\![\sigma]\!]_\delta}$.

(2) If $\Theta; \Gamma \vdash g \Rightarrow \uparrow P$

then for all Q *such that* $\Theta_0 \vdash \uparrow Q \leq^- [\lfloor\sigma\rfloor]\uparrow P$ *and* $\Theta_0; \Gamma_0 \vdash [\sigma]g \Rightarrow \uparrow Q$

we have $[\![\Theta_0; \Gamma_0 \vdash [\sigma]g \Rightarrow \uparrow Q]\!]_\delta = [\![\Theta; \Gamma \vdash g \Rightarrow \uparrow P]\!]_{[\![\sigma]\!]_\delta}$.

(3) If $\Theta; \Gamma \vdash v \Leftarrow P$ *then* $[\![\Theta_0; \Gamma_0 \vdash [\sigma]v \Leftarrow [\lfloor\sigma\rfloor]P]\!]_\delta = [\![\Theta; \Gamma \vdash v \Leftarrow P]\!]_{[\![\sigma]\!]_\delta}$.

(4) If $\Theta; \Gamma \vdash e \Leftarrow N$ *then* $[\![\Theta_0; \Gamma_0 \vdash [\sigma]e \Leftarrow [\lfloor\sigma\rfloor]N]\!]_\delta = [\![\Theta; \Gamma \vdash e \Leftarrow N]\!]_{[\![\sigma]\!]_\delta}$.

(5) If $\Theta; \Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$ *then*

$$[\![\Theta_0; \Gamma_0; [[\lfloor\sigma\rfloor]P] \vdash \{r_i \Rightarrow [\sigma]e_i\}_{i \in I} \Leftarrow [\lfloor\sigma\rfloor]N]\!]_\delta = [\![\Theta; \Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N]\!]_{[\![\sigma]\!]_\delta}$$

(6) If $\Theta; \Gamma; [N] \vdash s \Rightarrow \uparrow P$

then $[\![\Theta_0; \Gamma_0; [[\lfloor\sigma\rfloor]N] \vdash [\sigma]s \Rightarrow \uparrow[\lfloor\sigma\rfloor]P]\!]_\delta = [\![\Theta; \Gamma; [N] \vdash s \Rightarrow \uparrow P]\!]_{[\![\sigma]\!]_\delta}$.

Proof. Note that by Lemma E.28 (Substitution Typing Soundness), $\vdash [\![\sigma]\!]_\delta : \Theta; \Gamma$.

The proof of each part is similar. The first two parts are the most complicated, but also similar to each other. So, we will show only part (1):

We are given a derivation $\Theta; \Gamma \vdash h \Rightarrow P$. Suppose $\Theta_0 \vdash Q \leq^+ [\lfloor\sigma\rfloor]P$ and $\Theta_0; \Gamma_0 \vdash [\sigma]h \Rightarrow Q$. By Lemma D.31 (Erasure of Typing), $|\Gamma| \vdash |h| \Rightarrow |P|$ and $|\Gamma_0| \vdash |[\sigma]h| \Rightarrow |Q|$.

Further,

$$\begin{aligned} |Q| &= |[\lfloor\sigma\rfloor]P| && \text{By Lemma D.29 (Subtyping Erases to Equality)} \\ &= |P| && \text{By Lemma D.28 (Erasure Subst. Invariant)} \end{aligned}$$

We are also given derivations $\Theta_0;\Gamma_0 \vdash \sigma : \Theta;\Gamma$ and $\vdash \delta : \Theta_0;\Gamma_0$. By Lemma D.33 (Erasure of Substitution Typing), $|\Gamma_0| \vdash |\sigma| : |\Gamma|$ and $\vdash |\delta| : |\Gamma_0|$.

Therefore,

$$\begin{aligned} &[\![\Theta_0;\Gamma_0 \vdash [\sigma]h \Rightarrow Q]\!]_\delta \\ &= [\![|\Gamma_0| \vdash |[\sigma]h| \Rightarrow |Q|]\!]_{|\delta|} && \text{By def. of } [\![-]\!] \\ &= [\![|\Gamma_0| \vdash [|\sigma|]|h| \Rightarrow |Q|]\!]_{|\delta|} && \text{By Lemma D.32 (Erasure and Substitution Commute)} \\ &= [\![|\Gamma| \vdash |h| \Rightarrow |Q|]\!]_{[\![|\sigma|]\!]_{|\delta|}} && \text{By Lemma D.25 (Unrefined Substitution Soundness)} \\ &= [\![|\Gamma| \vdash |h| \Rightarrow |Q|]\!]_{|[\![\sigma]\!]_\delta|} && \text{By Lemma D.34 ($|-|$ and $[\![-]\!]$ Commute)} \\ &= [\![|\Gamma| \vdash |h| \Rightarrow |P|]\!]_{|[\![\sigma]\!]_\delta|} && \text{Above } (|Q| = |P|) \\ &= [\![\Theta;\Gamma \vdash h \Rightarrow P]\!]_{[\![\sigma]\!]_\delta} && \text{By def. of } [\![-]\!] \square \end{aligned}$$

Appendix F

Basic Properties of Algorithmic System

Algorithmic contexts are always a logical context followed by entries for existential variables and (possibly) their solutions. In the algorithmic system, we therefore add universal variables to the left of algorithmic contexts, against the norm of growing contexts rightward. Algorithmic context well-formedness is maintained by weakening solutions and propositions as needed. We leave these details implicit in the algorithmic metatheory.

We often implicitly use the following lemmas.

Lemma F.1 (Right-hand Subst.)**.**

(1) If $\hat{\Xi} \rhd t : \tau\ [\xi]$ *then* $\hat{\Xi} \rhd [\hat{\Xi}]t : \tau\ [^{\mathsf{d}}[\hat{\Xi}]\xi]$.

(2) If $\hat{\Xi}; [\tau] \rhd \boldsymbol{t} : \kappa$ *then* $\hat{\Xi}; [\tau] \rhd [\hat{\Xi}]\boldsymbol{t} : \kappa$.

(3) If $\|\hat{\Xi}\| \rhd u : \omega\ [_]$ *and* $\hat{\Xi}; [\omega] \rhd \boldsymbol{t} : \kappa$ *then* $\hat{\Xi} \rhd \langle u \mid \boldsymbol{t} \rangle : \kappa\ [_]$.

(4) If $\hat{\Xi} \rhd A\ \mathsf{type}[\xi]$ *then* $\hat{\Xi} \rhd [\hat{\Xi}]A\ \mathsf{type}[^{\mathsf{d}}[\hat{\Xi}]\xi]$.

(5) If $\hat{\Xi} \rhd \mathscr{F}\ \mathsf{functor}[\xi]$ *then* $\hat{\Xi} \rhd [\hat{\Xi}]\mathscr{F}\ \mathsf{functor}[^{\mathsf{d}}[\hat{\Xi}]\xi]$.

(6) If $\hat{\Xi} \rhd \alpha : F(\tau) \Rightarrow \tau$ *then* $\hat{\Xi} \rhd [\hat{\Xi}]\alpha : ([\hat{\Xi}]F)(\tau) \Rightarrow \tau$.

(7) If $\hat{\Xi} \rhd \mathscr{M}(F)\ \mathsf{msmts}[\xi]$ *then* $\hat{\Xi} \rhd [\hat{\Xi}]\mathscr{M}([\hat{\Xi}]F)\ \mathsf{msmts}[[\hat{\Xi}]\xi]$.

Similar statements hold for constraint well-formedness.

Proof. Similar to proof of Lemma C.17 (Ix. Syntactic Substitution) (parts (1) through (3)) and proof of Lemma C.51 (WF Syn. Substitution) (parts (4) through (7)). □

Lemma F.2 (Right-hand Subst. (Unroll))**.**
If $\hat{\Xi} \rhd \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta; R$ *then* $\hat{\Xi} \rhd \wr [\hat{\Xi}]\overrightarrow{\beta}; [\hat{\Xi}]G; [\hat{\Xi}]\mathscr{M}([\hat{\Xi}]F) \int \doteq [\hat{\Xi}]{}^{\mathsf{d}}\Theta; [\hat{\Xi}]R$.

Proof. Similar to proof of Lemma C.57 (Unrolling Syntactic Substitution). □

Lemma F.3 (Alg. Unrolling Output WF)**.**
If $\hat{\Xi} \rhd \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta; R$ *and* $\hat{\Xi} \rhd G\ \mathsf{functor}[\xi_G]$
then there exists ξ *such that* $\hat{\Xi} \rhd \exists {}^{\mathsf{d}}\Theta.\, R \wedge {}^{\mathsf{d}}\Theta\ \mathsf{type}[\xi]$ *and* $\xi_G \subseteq \xi$.

Proof. Similar to proof of Lemma C.55 (Unrolling Output WF). □

Appendix F.1 Algorithmic Extension

Lemma F.4 (Extension Sound)**.** *If* $\hat{\Theta} \longrightarrow \hat{\Theta}'$ *then* $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \hat{\Theta}'$.

Proof. Almost immediate from definitions. □

Lemma F.5 (Ext. Reflexive)**.** *If* $\hat{\Theta}\ \mathsf{algctx}$ *then*

(1) $\hat{\Theta} \longrightarrow \hat{\Theta}$

(2) $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \hat{\Theta}$

Proof. Almost immediate from definitions. □

Lemma F.6 (Ext. Transitive)**.**

(1) If $\hat{\Theta}_1 \longrightarrow \hat{\Theta}_2$ *and* $\hat{\Theta}_2 \longrightarrow \hat{\Theta}_3$ *then* $\hat{\Theta}_1 \longrightarrow \hat{\Theta}_3$.

(2) If $\hat{\Theta}_1 \xrightarrow{\mathsf{SMT}} \hat{\Theta}_2$ *and* $\hat{\Theta}_2 \xrightarrow{\mathsf{SMT}} \hat{\Theta}_3$ *then* $\hat{\Theta}_1 \xrightarrow{\mathsf{SMT}} \hat{\Theta}_3$.

(3) If $\hat{\Theta}_1 \xrightarrow{\mathsf{SMT}} \hat{\Theta}_2$ *and* $\hat{\Theta}_2 \longrightarrow \hat{\Theta}_3$ *then* $\hat{\Theta}_1 \xrightarrow{\mathsf{SMT}} \hat{\Theta}_3$.

Proof. Each part by straightforward induction on the first given extension derivation. Parts (2) and (3) use Lemma C.27 (Prop. Truth Equiv. Relation). □

Lemma F.7 (Ext. Weakening (Ixs.))**.** *Assume* $\hat{\Xi} \longrightarrow \hat{\Xi}'$ *or* $\hat{\Xi} \xrightarrow{\mathsf{SMT}} \hat{\Xi}'$.

(1) If $\hat{\Xi} \rhd t : \tau\ [\xi]$ *then* $\hat{\Xi}' \rhd t : \tau\ [\xi]$.

(2) If $\hat{\Xi}; [\tau] \rhd \boldsymbol{t} : \kappa$ *then* $\hat{\Xi}'; [\tau] \rhd \boldsymbol{t} : \kappa$.

Proof. By mutual induction on the structure of the given sorting derivation □

We often use the following lemma Lemma F.8 (Extension Restriction) implicitly.

Lemma F.8 (Extension Restriction)**.**

(1) If $\hat{\Theta} \longrightarrow \hat{\Theta}'$ *then* $\overset{\mathsf{d}}{\cdot}\hat{\Theta} \longrightarrow \overset{\mathsf{d}}{\cdot}\hat{\Theta}'$.

(2) If $\hat{\Xi} \xrightarrow{\mathsf{SMT}} \hat{\Xi}'$ *then* $\overset{\mathsf{d}}{\cdot}\hat{\Xi} \xrightarrow{\mathsf{SMT}} \overset{\mathsf{d}}{\cdot}\hat{\Xi}'$.

(3) If $\hat{\Theta} \longrightarrow \hat{\Theta}'$ *then* $\overline{\hat{\Theta}} \longrightarrow \overline{\hat{\Theta}'}$.

(The same statements but with relaxed extension do not hold because relaxed extension may use propositions which $\overset{\mathsf{d}}{\cdot}-$ *and* $\overline{\ }$ *remove.)*

Proof. (1) By structural induction on the given extension derivation. All evars are value-determined, so $dom(\overset{\mathsf{d}}{\cdot}\hat{\Theta})$ has every evar of $dom(\hat{\Theta})$ (by definition of $\overset{\mathsf{d}}{\cdot}-$). Further, each evar solution is well-formed under its (restricted, prefix) context because it is

presupposed that $\hat{\Theta}$ and $\hat{\Theta}'$ are well-formed, and solutions in well-formed algorithmic contexts are value-determined. Also, $\overset{\mathsf{d}}{\div}-$ removes propositions, so we need not worry about their well-formedness in the goal extension (as they cannot exist there).

(2) By structural induction on the given extension derivation. The extension is relaxed but a $\hat{\Xi}$ does not have propositions.

(3) By structural induction on the given extension derivation. Propositions aren't used in extension (relaxed or otherwise) to check propositional truth because $\overset{\mathsf{d}}{\div}-$ removes propositions. □

Lemma F.9 (Ext. Weakening (Types))**.** *Assume* $\hat{\Xi} \longrightarrow \hat{\Xi}'$ *or* $\hat{\Xi} \xrightarrow{\mathsf{SMT}} \hat{\Xi}'$.

(1) If $\hat{\Xi} \rhd A\ \mathsf{type}[\xi]$ *then* $\hat{\Xi}' \rhd A\ \mathsf{type}[\xi]$.

(2) If $\hat{\Xi} \rhd \mathscr{F}\ \mathsf{functor}[\xi]$ *then* $\hat{\Xi}' \rhd \mathscr{F}\ \mathsf{functor}[\xi]$.

(3) If $\hat{\Xi} \rhd \alpha : F(\tau) \Rightarrow \tau$ *then* $\hat{\Xi}' \rhd \alpha : F(\tau) \Rightarrow \tau$.

(4) If $\hat{\Xi} \rhd \mathscr{M}(F)\ \mathsf{msmts}[\xi]$ *then* $\hat{\Xi}' \rhd \mathscr{M}(F)\ \mathsf{msmts}[\xi]$.

Proof. By mutual induction on the structure of the given well-formedness derivation. Use Lemma F.7 (Ext. Weakening (Ixs.)) and Lemma F.8 (Extension Restriction) as needed. □

Lemma F.10 (Ext. Weaken (Unroll))**.**
If $\hat{\Xi} \rhd \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \wr \mathrel{\hat{=}} {}^{\mathsf{d}}\Theta; R$ *and* $\hat{\Xi} \longrightarrow \hat{\Xi}'$ *then* $\hat{\Xi}' \rhd \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \wr \mathrel{\hat{=}} {}^{\mathsf{d}}\Theta; R$.

Proof. By structural induction on the given unrolling derivation. Use Lemma F.9 (Ext. Weakening (Types)). □

Lemma F.11 (Inst. Extends)**.**

(1) *If* $\hat{\Theta} \vdash {}^{(\forall)}W\ \mathsf{Inst} \dashv \hat{\Theta}'$ *then* $\hat{\Theta} \longrightarrow \hat{\Theta}'$.

(2) *If* $\hat{\Theta} \vdash W\ \mathsf{Inst}\blacktriangleright \dashv \hat{\Theta}'$ *then* $\hat{\Theta} \longrightarrow \hat{\Theta}'$.

(3) *If* $\hat{\Theta} \vdash W\ \mathsf{fixInst} \dashv \hat{\Theta}'$ *then* $\hat{\Theta} \longrightarrow \hat{\Theta}'$.

(4) *If* $\hat{\Theta}; \Gamma \vdash \chi\ \mathsf{fixInstChk} \dashv \Omega$ *then* $\hat{\Theta} \longrightarrow \Omega$.

Proof. (1) By structural induction on $\hat{\Theta} \vdash {}^{(\forall)}W\ \mathsf{Inst} \dashv \hat{\Theta}'$; case analyze its structure. Use Lemma F.5 (Ext. Reflexive) and Lemma F.6 (Ext. Transitive).

(2) By structural induction on $\hat{\Theta} \vdash W\ \mathsf{Inst}\blacktriangleright \dashv \hat{\Theta}'$. Similar to part (1).

(3) By structural induction on $\hat{\Theta} \vdash W\ \mathsf{fixInst} \dashv \hat{\Theta}'$. Use part (1) and Lemma F.6 (Ext. Transitive).

(4) By part (3), part (2), and Lemma F.6 (Ext. Transitive). □

Appendix F.2 Algorithmic Substitution and Well-Formedness Properties

Lemma F.12 (Alg. to Decl. WF)**.** *Assume* $\hat{\Xi} \longrightarrow \Omega$ *or* $\hat{\Xi} \xrightarrow{\mathsf{SMT}} \Omega$.

(1) *If* $\hat{\Xi} \rhd t : \tau\ [\xi]$ *then* $\|\hat{\Xi}\| \vdash [\Omega]^2 t : \tau\ [^{\mathsf{d}}[\Omega]^2 \xi]$.

(2) *If* $\hat{\Xi}; [\tau] \rhd \boldsymbol{t} : \kappa$ *then* $\|\hat{\Xi}\|; [\tau] \vdash [\Omega]^2 \boldsymbol{t} : \kappa$.

(3) *If* $\hat{\Xi} \rhd A\ \mathsf{type}[\xi]$ *then* $\|\hat{\Xi}\| \vdash [\Omega]^2 A\ \mathsf{type}[^{\mathsf{d}}[\Omega]^2 \xi]$.

(4) *If* $\hat{\Xi} \rhd \mathscr{F}\ \mathsf{functor}[\xi]$ *then* $\|\hat{\Xi}\| \vdash [\Omega]^2 \mathscr{F}\ \mathsf{functor}[^{\mathsf{d}}[\Omega]^2 \xi]$.

(5) *If* $\hat{\Xi} \rhd \alpha : F(\tau) \Rightarrow \tau$ *then* $\|\hat{\Xi}\| \vdash [\Omega]^2 \alpha : ([\Omega]^2 F)(\tau) \Rightarrow \tau$.

(6) *If* $\hat{\Xi} \rhd \mathscr{M}(F)\ \mathsf{msmts}[\xi]$ *then* $\|\hat{\Xi}\| \vdash [\Omega]^2 \mathscr{M}(F)\ \mathsf{msmts}[^{\mathsf{d}}[\Omega]^2 \xi]$.

Proof. By Lemma F.7 (Ext. Weakening (Ixs.)) and Lemma F.9 (Ext. Weakening (Types)) and Lemma F.1 (Right-hand Subst) and the straightforward correspondence between ground algorithmic derivations and declarative derivations. For example, for part (1), we get $\Omega \rhd [\Omega]^2 t : \tau\ [^{\mathsf{d}}[\Omega]^2 \xi]$. Because $[\Omega]^2 t$ ground, we have $\|\Omega\| \vdash [\Omega]^2 t : \tau\ [^{\mathsf{d}}[\Omega]^2 \xi]$. But $\|\Omega\| = \|\hat{\Xi}\|$. □

Lemma F.13 (Complete Unroll)**.** *If* $\hat{\Xi} \rhd \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta; R$ *and* $\hat{\Xi} \longrightarrow \Omega$
then $\|\hat{\Xi}\| \vdash \wr [\Omega]^2 \overrightarrow{\beta}; [\Omega]^2 G; [\Omega]^2 \mathscr{M}([\Omega]^2 F) \int \doteq [\Omega]^2({}^{\mathsf{d}}\Theta); [\Omega]^2 R$.

Proof. By Lemma F.10 (Ext. Weaken (Unroll)), $\Omega \rhd \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta; R$. By Lemma F.2 (Right-hand Subst. (Unroll)) $\Omega \rhd \wr [\Omega]^2 \overrightarrow{\beta}; [\Omega]^2 G; [\Omega]^2 \mathscr{M}([\Omega]^2 F) \int \doteq [\Omega]^{2\mathsf{d}}\Theta; [\Omega]^2 R$.
We know $[\Omega]\Omega$ (which is complete) has ground solutions under $\|\Omega\| = \|\hat{\Xi}\|$.
Therefore, $\|\Omega\| \vdash \wr [\Omega]^2 \overrightarrow{\beta}; [\Omega]^2 G; [\Omega]^2 \mathscr{M}([\Omega]^2 F) \int \doteq [\Omega]^{2\mathsf{d}}\Theta; [\Omega]^2 R$. (By Barendregt's lemma, $[\Omega]^2 - = [[\Omega]\Omega] -$.) The goal follows by $\|\Omega\| = \|\hat{\Xi}\|$. □

Lemma F.14 (Uncomplete Unrolling)**.**
If $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$ *and* $\hat{\Theta}$ present *and* $\overset{\mathsf{d}\cdot}{\div}\|\hat{\Theta}\|, {}^{\mathsf{d}}\Xi \vdash \wr [\Omega] \overrightarrow{\beta}; [\Omega] G; [\Omega] \mathscr{M}([\Omega] F) \int \doteq [\Omega]({}^{\mathsf{d}}\Theta); [\Omega] R$
and $\overline{\hat{\Theta}} \rhd \mathscr{M}(F)$ msmts[_] *and* $\overline{\hat{\Theta}}, {}^{\mathsf{d}}\Xi \rhd \overrightarrow{\beta} : G(\mathscr{M}(F)) \Rightarrow \mathscr{M}(F)$
then there exist ${}^{\mathsf{d}}\Theta'$ *and* R' *such that* $\overset{\mathsf{d}\cdot}{\div}\hat{\Theta}, {}^{\mathsf{d}}\Xi \rhd \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta'; R'$
and $[\Omega]({}^{\mathsf{d}}\Theta') = {}^{\mathsf{d}}\Theta$ *and* $[\Omega] R' = R$.

Proof. By structural induction on the given unrolling derivation, using similar but unstated and straightforward counterparts of this lemma ("Uncomplete Unrolling") for other judgments. Use the algorithmic version of Lemma C.20 (Ix. Barendregt) in the Alg$\wr I \int$ case. □

Lemma F.15 (Unroll Applied)**.** *If* $\hat{\Xi} \rhd \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta; R$
and $[\hat{\Xi}]F = F$ *and* $[\hat{\Xi}](\mathscr{M}(F)) = \mathscr{M}(F)$ *and* $[\hat{\Xi}]G = G$ *and* $[\hat{\Xi}]\overrightarrow{\beta} = \overrightarrow{\beta}$
then $[\hat{\Xi}]{}^{\mathsf{d}}\Theta = {}^{\mathsf{d}}\Theta$ *and* $[\hat{\Xi}]R = R$.

Proof. By structural induction on the given unrolling derivation, case analyzing its concluding rule. Each case is straightforward. Use similar but unstated and straightforward counterparts of this lemma ("Unroll Applied") for other judgments (such as $\beta \circ \mathsf{inj}_1 \mathrel{\mathring{=}} \beta_1$). □

Appendix F.3 Decidability

When giving judgments an induction metric, we only take input metavariables into account, and often put "_" ("don't care") for outputs to reflect this.

It is straightforward to check that all the well-formedness (including index and index spine sorting) judgments are decidable (and we often don't mention this in subsequent proofs). For example consider the judgment $\xi \vdash \mathfrak{D}\ \mathsf{det}$, used in type well-formedness:

Lemma F.16 (Det. Decidable)**.** *Given ξ and $\mathfrak{D}$ it is decidable whether $\xi \vdash \mathfrak{D}\ \mathsf{det}$.*

Proof. There are only finitely many $d \in \mathfrak{D}\restriction_{FV(\xi)}$ to check whether $\xi \vdash d\ \mathsf{det}$. Rule DetUnit is decidable because checking whether an element is in a finite set is decidable. In rule DetCut, the cardinality of the finite set $\mathit{neg}(\xi)$ strictly decreases for each premise. □

Lemma F.17 (Prop. Truth Decidable)**.** *Given $\Theta\ \mathsf{ctx}$ and $\overline{\Theta} \vdash t : \mathbb{B}$,*
it is decidable whether $\Theta \vdash t\ \mathsf{true}$.

Proof. See Barrett et al. [2009]. □

Lemma F.18 (Ix. Equiv. Decidable)**.** *Given $\Xi \vdash u : \tau\ [_]$ and $\Xi \vdash t : \tau\ [_]$,*
it is decidable whether $\Xi \vdash u \equiv t : \tau$.

Proof. In each rule concluding $\Xi \vdash u \equiv t : \tau$, for every premise of this form, the structure of both u and t decreases in size. The Ix≡SMT case uses Lemma F.17 (Prop. Truth Decidable). □

Lemma F.19 (Ix. Spine Equiv. Decidable)**.** *Given* $\Xi;[\tau] \vdash \boldsymbol{u} : \kappa$ *and* $\Xi;[\tau] \vdash \boldsymbol{t} : \kappa$*, it is decidable whether* $\Xi;[\tau] \vdash \boldsymbol{u} \equiv \boldsymbol{t} : \kappa$.

Proof. The IxSpine≡Nil case has no premises. For rules IxSpine≡Entry and IxSpine≡Proj$_k$, for each premise of form $\Xi;[\tau] \vdash \boldsymbol{u} \equiv \boldsymbol{t} : \kappa$, the structure of both $\boldsymbol{u}$ and $\boldsymbol{t}$ get smaller. The IxSpine≡Entry case uses Lemma F.17 (Prop. Truth Decidable). □

Lemma F.20 (Inst. Succeeds)**.** *Assume* $\hat{\Theta}$ algctx.

(1) Given $\overline{\hat{\Theta}} \vdash {}^{(\forall)}W$ wf*, we can compute the unique* $\hat{\Theta}'$ *such that* $\hat{\Theta} \vdash {}^{(\forall)}W$ Inst $\dashv \hat{\Theta}'$.

(2) Given $\overline{\hat{\Theta}} \vdash W$ wf*, we can compute the unique* $\hat{\Theta}'$ *such that* $\hat{\Theta} \vdash W$ Inst▶ $\dashv \hat{\Theta}'$.

(3) Given $\overline{\hat{\Theta}} \vdash W$ wf*, we can compute the unique* $\hat{\Theta}'$ *such that* $\hat{\Theta} \vdash W$ fixInst $\dashv \hat{\Theta}'$.

Proof. Straightforward. □

Lemma F.21 (Algebra Pattern Match Succeeds)**.**

(1) Given algebra α*, we can compute the unique* α_1 *such that* $\alpha \circ \mathsf{inj}_1 \doteq \alpha_1$.

(2) Given algebra α*, we can compute the unique* α_2 *such that* $\alpha \circ \mathsf{inj}_2 \doteq \alpha_2$.

Proof. Straightforward (measure $\alpha \circ \mathsf{inj}_k \doteq _$ by the number of clauses in α). □

Theorem F.1 (Subtyping Decidable)**.**

(1) Given Θ ctx *and* $\overline{\Theta} \vdash {}^{(\forall)}W$ wf*, it is decidable whether* $\Theta \models {}^{(\forall)}W$.

(2) Given $\hat{\Theta}$ algctx *and* $\overline{\hat{\Theta}} \rhd A$ type[_] *and* $\overline{\hat{\Theta}} \rhd B$ type[_]*, it is decidable whether there exists* ${}^{(\forall)}W$ *such that* $\hat{\Theta} \vdash A <:^{\pm} B \mathbin{/} {}^{(\forall)}W$.

(3) Given $\hat{\Theta}$ algctx *and* $\overline{\|\hat{\Theta}\|} \rhd A$ type[_] *and* $\overline{\|\hat{\Theta}\|} \rhd B$ type[_]*, it is decidable whether* $\|\hat{\Theta}\| \vdash A <:^{\pm} B$.

(4) Given Ξ ctx *and* $\Xi \rhd \alpha : F(\tau) \Rightarrow \tau$ *and* $\Xi \rhd \beta : G(\tau) \Rightarrow \tau$,
it is decidable whether $\Xi \rhd \alpha; F <:_\tau \beta; G$.

(5) Given $\hat{\Theta}$ algctx *and* $\overline{\hat{\Theta}} \rhd \mathscr{M}'(F')$ msmts[_] *and* $\overline{\hat{\Theta}} \rhd \mathscr{M}(F)$ msmts[_],
it is decidable whether there exists W *such that* $\hat{\Theta} \vdash \mathscr{M}'(F') \geq \mathscr{M}(F) \,/\, W$.

(6) Given $\overline{\hat{\Theta}} \vdash W$ wf,
it is decidable whether there exists Ω *such that* $\hat{\Theta}; \cdot \vdash W$ fixInstChk $\dashv \Omega$.

Proof. By mutual induction the structure of the given derivation

$$
\begin{array}{ll}
\Theta \models^{(\forall)} W & \text{or} \\
\hat{\Theta} \vdash A <:^{\pm} B \,/\, _ & \text{or} \\
\hat{\Theta} \vdash A <:^{\pm} B & \text{or} \\
\Xi \rhd \alpha; F <:_\tau \beta; G & \text{or} \\
\hat{\Theta} \vdash \mathscr{M}'(F') \geq \mathscr{M}(F) \,/\, _ & \text{or} \\
\hat{\Theta}; \cdot \vdash W \text{ fixInstChk} \dashv _ &
\end{array}
$$

For each rule deriving one of the above judgments, every premise is either smaller than the conclusion (according to the above measure) or already known to be decidable.

Use Lemma F.20 (Inst. Succeeds), Lemma F.21 (Algebra Pattern Match Succeeds), Lemma F.17 (Prop. Truth Decidable), Lemma F.18 (Ix. Equiv. Decidable), and Lemma F.19 (Ix. Spine Equiv. Decidable).

Regarding part (5), fixInstChk: There are finitely many possible choices of $\overrightarrow{W^b}$; after running Inst▶ on a $\overrightarrow{W^b}$, it is decidable to check whether its output context is complete (Ω), and then whether $\|\hat{\Theta}\| \models [\Omega][\Omega]W$. □

Lemma F.22 (Unrolling Decidable)**.**
Given $\hat{\Xi}$ algctx *and* $\hat{\Xi} \rhd \overrightarrow{\beta} : G(\mathscr{M}(F)) \Rightarrow \mathscr{M}(F)$ *and* $\hat{\Xi} \rhd \mathscr{M}(F)$ msmts[_] *it is decidable whether there exist* ${}^{\mathsf{d}}\Theta$ *and* R *such that* $\hat{\Xi} \rhd \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta; R$.

Proof. Measuring the judgment $\hat{\Xi} \rhd \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \int \doteq _;_$ (putting "_" for the outputs) by the structure of G, it is easy to show that, in each rule deriving said judgment, every main premise gets smaller. In the Alg$\wr \oplus \int$ case use Lemma F.21 (Algebra Pattern Match Succeeds). Similarly, it is easy to see that the side conditions of other cases are decidable.

□

Theorem F.2 (Decidability of Typing)**.** *Assume* $\hat{\Theta}$ algctx *and* $\overline{\|\hat{\Theta}\|} \vdash \Gamma$ ctx.

(1) Given $\overline{\|\hat{\Theta}\|} \vdash \chi$ Wf*, it is decidable whether* $\|\hat{\Theta}\|; \Gamma \lhd \chi$.

(2) Given head h*, it is decidable whether there exists* P *such that* $\|\hat{\Theta}\|; \Gamma \rhd h \Rightarrow P$.

(3) Given bound expression g*,*
it is decidable whether there exists P *such that* $\|\hat{\Theta}\|; \Gamma \rhd g \Rightarrow \uparrow P$.

(4) Given $\overline{\hat{\Theta}} \rhd P$ type[_] *and value* v*,*
it is decidable whether there exist χ *and* Δ *such that* $\hat{\Theta}; \Gamma \vdash v \Leftarrow P \,/\, \chi \dashv \Delta$.

(5) Given $\overline{\|\hat{\Theta}\|} \vdash P$ type[_] *and value* v*,*
it is decidable whether such that $\|\hat{\Theta}\|; \Gamma \rhd v \Leftarrow P$.

(6) Given $\overline{\|\hat{\Theta}\|} \vdash N$ type[_] *and expression* e*,*
it is decidable whether $\|\hat{\Theta}\|; \Gamma \rhd e \Leftarrow N$.

(7) Given $\overline{\|\hat{\Theta}\|} \vdash P$ type[_] *and* $\overline{\|\hat{\Theta}\|} \vdash N$ type[_] *and match expression* $\{r_i \Rightarrow e_i\}_{i \in I}$*,*
it is decidable whether $\|\hat{\Theta}\|; \Gamma; [P] \rhd \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$.

(8) *Given* $\overline{\hat{\Theta}} \rhd N$ type$[_]$ *and spine* s,

it is decidable whether there exist P, χ, *and* Δ

such that $\hat{\Theta};\Gamma;[N] \vdash s \Rightarrow \uparrow P \,/\, \chi \dashv \Delta$.

(9) *Given* $\overline{\|\hat{\Theta}\|} \vdash N$ type$[_]$ *and spine* s,

it is decidable whether there exists P *such that* $\|\hat{\Theta}\|;\Gamma;[N] \rhd s \Rightarrow \uparrow P$.

(10) *Given* $\overline{\|\hat{\Theta}\|} \vdash \chi$ Wf,

it is decidable whether there exists Ω *such that* $\hat{\Theta};\Gamma \vdash \chi$ fixInstChk $\dashv \Omega$.

Proof. Let $\Theta = \|\hat{\Theta}\|$. For rules deriving judgments of the form

$$
\begin{array}{ll}
\Theta;\Gamma \lhd \chi & \text{or} \\
\Theta;\Gamma \rhd h \Rightarrow _ & \text{or} \\
\Theta;\Gamma \rhd g \Rightarrow _ & \text{or} \\
\hat{\Theta};\Gamma \vdash v \Leftarrow A \,/\, _ \dashv _ & \text{or} \\
\Theta;\Gamma \rhd v \Leftarrow A & \text{or} \\
\Theta;\Gamma \rhd e \Leftarrow A & \text{or} \\
\Theta;\Gamma;[A] \rhd \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N & \text{or} \\
\hat{\Theta};\Gamma;[A] \vdash s \Rightarrow _ \,/\, _ \dashv _ & \text{or} \\
\Theta;\Gamma;[A] \rhd s \Rightarrow _ & \text{or} \\
\hat{\Theta};\Gamma \vdash \chi \text{ fixInstChk} \dashv \Omega &
\end{array}
$$

(where we write "_" for parts of the judgments that are outputs), the following induction measure on such judgments is adequate to prove decidability: the structure of the derivation of the judgment (these judgments are all mutually recursive).

Use Lemma F.22 (Unrolling Decidable), Lemma F.3 (Alg. Unrolling Output WF),

Lemma F.17 (Prop. Truth Decidable), and Lemma F.20 (Inst. Succeeds). □

Appendix F.4 Algorithmic Soundness

Lemma F.23 (Erase ▶ From Extension)**.**

If $\hat{\Theta} \longrightarrow \Omega$ *and no* ▶*-evar is solved in* $\hat{\Theta}$ *then* $|\hat{\Theta}|_{\blacktriangleright} \longrightarrow |[\Omega]\Omega|_{\blacktriangleright}$

where $|-|_{\blacktriangleright}$ *merely removes all occurrences of* ▶*,*

that is, $|-|_{\blacktriangleright}$ *is defined by*

$$|\Theta|_{\blacktriangleright} = \Theta$$

$$|\hat{\Theta}, [\blacktriangleright]a \stackrel{\mathsf{d}}{\div} \kappa|_{\blacktriangleright} = |\hat{\Theta}|_{\blacktriangleright}, a \stackrel{\mathsf{d}}{\div} \kappa$$

$$|\hat{\Theta}, [\blacktriangleright]\hat{a} : \kappa{=}t|_{\blacktriangleright} = |\hat{\Theta}|_{\blacktriangleright}, \hat{a} : \kappa{=}t$$

Proof. Straightforward. □

Theorem F.3 (Alg. Sub. Sound)**.**

(1) If $\hat{\Theta} \vdash R <:^{+} Q \,/\, {}^{(\forall)}W$ *and* $\hat{\Theta} \longrightarrow \Omega$ *and* $\|\hat{\Theta}\| \models [\Omega]^{(\forall)}W$ *and* $R\,\mathsf{ground}$ *and* $\overline{\hat{\Theta}} \rhd Q\,\mathsf{type}[\xi]$ *and* $[\hat{\Theta}]Q = Q$ *and* $\hat{\Theta}\,\mathsf{present}$ *then* $\|\hat{\Theta}\| \vdash R \leq^{+} [\Omega]Q$*.*

(2) If $\Theta \vdash R <:^{+} P$ *then* $\Theta \vdash R \leq^{+} P$*.*

(3) If $\hat{\Theta} \vdash M <:^{-} L \,/\, {}^{(\forall)}W$ *and* $\hat{\Theta} \longrightarrow \Omega$ *and* $\|\hat{\Theta}\| \models [\Omega]^{(\forall)}W$ *and* $\overline{\hat{\Theta}} \rhd L\,\mathsf{type}[\Xi]$ *and* $L\,\mathsf{ground}$ *and* $[\hat{\Theta}]M = M$ *and* $\hat{\Theta}\,\mathsf{present}$ *then* $\|\hat{\Theta}\| \vdash [\Omega]M \leq^{-} L$*.*

(4) If $\Theta \vdash N <:^{-} L$ *then* $\Theta \vdash N \leq^{-} L$*.*

(5) If $\Xi \rhd \alpha; F <:_\tau \beta; G$ *then* $\Xi \vdash \alpha; F \leq_\tau \beta; G$.

Proof. By induction on the sum of the height of the given subtyping derivation and the height of the given subtyping constraint derivation. All parts are mutually recursive. Use Lemma F.11 (Inst. Extends), Lemma F.23 (Erase ▶ From Extension), Barendregt's substitution distribution lemma, and Lemma F.5 (Ext. Reflexive). □

Theorem F.4 (Alg. Typing Sound)**.**

(1) If $\Theta; \Gamma \rhd h \Rightarrow P$ *then* $\Theta; \Gamma \vdash h \Rightarrow P$.

(2) If $\Theta; \Gamma \rhd g \Rightarrow \uparrow P$ *then* $\Theta; \Gamma \vdash g \Rightarrow \uparrow P$.

(3) If $\hat{\Theta}; \Gamma \vdash v \Leftarrow P \mathbin{/} \chi \dashv \Delta$ *and* $\hat{\Theta}, \Delta \longrightarrow \Omega$ *and* $\|\hat{\Theta}\|; \Gamma \lhd [\Omega]\chi$
and $\overline{\hat{\Theta}} \rhd P\ \mathsf{type}[\xi]$ *and* $[\hat{\Theta}]P = P$ *and* $\hat{\Theta}\ \mathsf{present}$
then $\|\hat{\Theta}\|; \Gamma \vdash [\Omega]v \Leftarrow [\Omega]P$.

(4) If $\Theta; \Gamma \rhd v \Leftarrow P$ *then* $\Theta; \Gamma \vdash v \Leftarrow P$.

(5) If $\Theta; \Gamma \rhd e \Leftarrow N$ *then* $\Theta; \Gamma \vdash e \Leftarrow N$.

(6) If $\Theta; \Gamma; [P] \rhd \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$ *then* $\Theta; \Gamma; [P] \vdash \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$.

(7) If $\hat{\Theta}; \Gamma; [M] \vdash s \Rightarrow \uparrow P \mathbin{/} \chi \dashv \Delta$ *and* $\hat{\Theta}, \Delta \longrightarrow \Omega$ *and* $\|\hat{\Theta}\|; \Gamma \lhd [\Omega]\chi$
and $\overline{\hat{\Theta}} \rhd M\ \mathsf{type}[\xi]$ *and* $[\hat{\Theta}]M = M$ *and* $\hat{\Theta}\ \mathsf{present}$
then $\|\hat{\Theta}\|; \Gamma; [[\Omega]M] \vdash [\Omega]s \Rightarrow [\Omega]\uparrow P$.

(8) If $\Theta; \Gamma; [N] \rhd s \Rightarrow \uparrow P$ *then* $\Theta; \Gamma; [N] \vdash s \Rightarrow \uparrow P$.

Proof. By induction on the sum of the height of the given typing derivation and the height of the given constraint verification derivation. All parts are mutually recursive.

Straightforward.

Use (roughly in order of part numbers) Lemma F.3 (Alg. Unrolling Output WF), Lemma F.15 (Unroll Applied), Lemma F.13 (Complete Unroll), Lemma F.11 (Inst. Extends), Lemma F.23 (Erase ▶ From Extension), Barendregt's substitution distribution lemma, Lemma F.5 (Ext. Reflexive) as needed. □

Appendix G

Algorithmic Completeness

Lemma G.1 (Weaken Inst.)**.** *Assume* $\hat{\Xi} \subseteq \hat{\Xi}_0$ *and* $\Theta, \hat{\Xi}_0$ algctx.

(1) If $\Theta, \hat{\Xi} \vdash {}^{(\forall)}W$ Inst $\dashv \Theta, \hat{\Xi}'$ *and* $[\hat{\Xi}]^{2(\forall)}W = {}^{(\forall)}W$
then $\Theta, \hat{\Xi}_0 \vdash {}^{(\forall)}W$ Inst $\dashv \Theta, \hat{\Xi}_0'$ *and* $\hat{\Xi}_0'\restriction_{dom(\hat{\Xi}')} = \hat{\Xi}'$ *and* $\hat{\Xi}_0' - \hat{\Xi}' = \hat{\Xi}_0 - \hat{\Xi}$.

(2) If $\Theta, \hat{\Xi} \vdash {}^{(\forall)}W$ Inst$\blacktriangleright$ $\dashv \Theta, \hat{\Xi}'$ *and* $[\hat{\Xi}]^{2(\forall)}W = {}^{(\forall)}W$
then $\Theta, \hat{\Xi}_0 \vdash {}^{(\forall)}W$ Inst$\blacktriangleright$ $\dashv \Theta, \hat{\Xi}_0'$ *and* $\hat{\Xi}_0'\restriction_{dom(\hat{\Xi}')} = \hat{\Xi}'$ *and* $\hat{\Xi}_0' - \hat{\Xi}' = \hat{\Xi}_0 - \hat{\Xi}$.

(3) If $\Theta, \hat{\Xi} \vdash W$ fixInst $\dashv \Theta, \hat{\Xi}'$ *and* $[\hat{\Xi}]^2 W = W$
then $\Theta, \hat{\Xi}_0 \vdash W$ fixInst $\dashv \Theta, \hat{\Xi}_0'$ *and* $\hat{\Xi}_0'\restriction_{dom(\hat{\Xi}')} = \hat{\Xi}'$ *and* $\hat{\Xi}_0' - \hat{\Xi}' = \hat{\Xi}_0 - \hat{\Xi}$.

Proof. Each part by structural induction on the given instantiation derivation. Part (3) uses part (2). Part (2) is similar to part (1). □

Lemma G.2 (Inst. Compose)**.**

(1) If $\Theta, \hat{\Xi}_1 \vdash {}^{(\forall)}W_1$ Inst $\dashv \Theta, \hat{\Xi}_1'$ *and* $\Theta, \hat{\Xi}_2 \vdash {}^{(\forall)}W_2$ Inst $\dashv \Theta, \hat{\Xi}_2'$
and $[\hat{\Xi}_1]^{2(\forall)}W_1 = {}^{(\forall)}W_1$ *and* $[\hat{\Xi}_2]^{2(\forall)}W_2 = {}^{(\forall)}W_2$ *and* $dom(\hat{\Xi}_1) \cap dom(\hat{\Xi}_2) = \emptyset$
then $\Theta, \hat{\Xi}_1, \hat{\Xi}_2 \vdash {}^{(\forall)}W_1 \wedge {}^{(\forall)}W_2$ Inst $\dashv \Theta, \hat{\Xi}_1', \hat{\Xi}_2'$.

(2) *If* $\Theta, \hat{\Xi}_1 \vdash {}^{(\forall)}W_1$ $\mathsf{Inst}\blacktriangleright \dashv \Theta, \hat{\Xi}_1'$ *and* $\Theta, \hat{\Xi}_2 \vdash {}^{(\forall)}W_2$ $\mathsf{Inst}\blacktriangleright \dashv \Theta, \hat{\Xi}_2'$
and $[\hat{\Xi}_1]^{2(\forall)}W_1 = {}^{(\forall)}W_1$ *and* $[\hat{\Xi}_2]^{2(\forall)}W_2 = {}^{(\forall)}W_2$ *and* $dom(\hat{\Xi}_1) \cap dom(\hat{\Xi}_2) = \emptyset$
then $\Theta, \hat{\Xi}_1, \hat{\Xi}_2 \vdash {}^{(\forall)}W_1 \wedge {}^{(\forall)}W_2$ $\mathsf{Inst}\blacktriangleright \dashv \Theta, \hat{\Xi}_1', \hat{\Xi}_2'$.

(3) *If* $\Theta, {}^{\mathsf{d}}\hat{\Xi} \vdash W$ $\mathsf{fixInst} \dashv \Theta, {}^{\mathsf{d}}\hat{\Xi}_0$ *and* $\Theta, {}^{\mathsf{d}}\hat{\Xi}' \vdash W'$ $\mathsf{fixInst} \dashv \Theta, {}^{\mathsf{d}}\hat{\Xi}_0'$
and $[{}^{\mathsf{d}}\hat{\Xi}]^2 W = W$ *and* $[{}^{\mathsf{d}}\hat{\Xi}']^2 W' = W'$ *and* $dom({}^{\mathsf{d}}\hat{\Xi}) \cap dom({}^{\mathsf{d}}\hat{\Xi}') = \emptyset$
then $\Theta, {}^{\mathsf{d}}\hat{\Xi}, {}^{\mathsf{d}}\hat{\Xi}' \vdash W \wedge W'$ $\mathsf{fixInst} \dashv \Theta, {}^{\mathsf{d}}\hat{\Xi}_0, {}^{\mathsf{d}}\hat{\Xi}_0'$.

(4) *If* $\Theta, {}^{\mathsf{d}}\hat{\Xi}; \cdot \vdash W$ $\mathsf{fixInstChk} \dashv \Theta, \Omega$ *and* $\Theta, {}^{\mathsf{d}}\hat{\Xi}'; \cdot \vdash W'$ $\mathsf{fixInstChk} \dashv \Theta, \Omega'$
and $[{}^{\mathsf{d}}\hat{\Xi}]^2 W = W$ *and* $[{}^{\mathsf{d}}\hat{\Xi}']^2 W' = W'$ *and* $dom({}^{\mathsf{d}}\hat{\Xi}) \cap dom({}^{\mathsf{d}}\hat{\Xi}') = \emptyset$
then $\Theta, {}^{\mathsf{d}}\hat{\Xi}, {}^{\mathsf{d}}\hat{\Xi}'; \cdot \vdash W \wedge W'$ $\mathsf{fixInstChk} \dashv \Theta, \Omega - \Theta, \Omega' - \Theta$.

(5) *If* $\Theta, {}^{\mathsf{d}}\hat{\Xi}; \Gamma \vdash \chi$ $\mathsf{fixInstChk} \dashv \Theta, \Omega$ *and* $\Theta, {}^{\mathsf{d}}\hat{\Xi}'; \Gamma \vdash \chi'$ $\mathsf{fixInstChk} \dashv \Theta, \Omega'$
and $[{}^{\mathsf{d}}\hat{\Xi}]^2 W = W$ *and* $[{}^{\mathsf{d}}\hat{\Xi}']^2 W' = W'$ *and* $dom({}^{\mathsf{d}}\hat{\Xi}) \cap dom({}^{\mathsf{d}}\hat{\Xi}') = \emptyset$
then $\Theta, {}^{\mathsf{d}}\hat{\Xi}, {}^{\mathsf{d}}\hat{\Xi}'; \Gamma \vdash \chi, \chi'$ $\mathsf{fixInstChk} \dashv \Theta, \Omega - \Theta, \Omega' - \Theta$.

Proof. (1)

$\Theta, \hat{\Xi}_1 \vdash {}^{(\forall)}W_1$ $\mathsf{Inst} \dashv \Theta, \hat{\Xi}_1'$	Given
$\Theta, \hat{\Xi}_1, \hat{\Xi}_2 \vdash {}^{(\forall)}W_1$ $\mathsf{Inst} \dashv \Theta, \hat{\Xi}_1', \hat{\Xi}_2$	By Lemma G.1 (Weaken Inst)
$\Theta, \hat{\Xi}_2 \vdash {}^{(\forall)}W_2$ $\mathsf{Inst} \dashv \Theta, \hat{\Xi}_2'$	Given
$\Theta, \hat{\Xi}_1', \hat{\Xi}_2 \vdash {}^{(\forall)}W_2$ $\mathsf{Inst} \dashv \Theta, \hat{\Xi}_1', \hat{\Xi}_2'$	By Lemma G.1 (Weaken Inst)
$\Theta, \hat{\Xi}_1, \hat{\Xi}_2 \vdash {}^{(\forall)}W_1 \wedge {}^{(\forall)}W_2$ $\mathsf{Inst} \dashv \Theta, \hat{\Xi}_1', \hat{\Xi}_2'$	By a rule

(2) Similar to previous part.

(3) By structural induction on the given instantiation derivations, using part (1) and Lemma G.1 (Weaken Inst) and Lemma F.11 (Inst. Extends) (with WF presuppositions and substitution properties) as needed.

(4) Straightforward. Use part (2) and part (3) and an unstated lemma about composing select.

(5) Similar to previous part. □

Appendix G.1 Intermediate Metatheory

Lemma G.3 (Semidecl. Sub. Sound)**.**

(1) If $\Theta \mathrel{\widetilde{\vdash}} R <:^{+} P \mathbin{/} {}^{(\forall)}W$ *and* $\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W$ *then* $\Theta \vdash R \leq^{+} P$.

(2) If $\Theta \mathrel{\widetilde{\vdash}} N <:^{-} L \mathbin{/} {}^{(\forall)}W$ *and* $\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W$ *then* $\Theta \vdash N \leq^{-} L$.

(3) If $\Xi \mathrel{\widetilde{\vdash}} \alpha; F <:_{\tau} \alpha'; F'$ *then* $\Xi \vdash \alpha; F \leq_{\tau} \alpha'; F'$.

(4) If $\Theta \mathrel{\widetilde{\vdash}} \mathscr{M}(F) \geq \mathscr{M}'(F') \mathbin{/} W$ *and* $\Theta \mathrel{\widetilde{\models}} W$ *then* $\Theta \vdash \mathscr{M}(F) \geq \mathscr{M}'(F')$.

Proof. By mutual induction on the structure of the given derivation. Straightforward. We show one case of part (3).

- **Case**

$$\frac{\Xi, \overrightarrow{\varphi} \mathrel{\widetilde{\vdash}} R <:^{+} Q' \mathbin{/} {}^{(\forall)}W \qquad \Xi, \overrightarrow{\varphi} \mathrel{\widetilde{\models}} {}^{(\forall)}W \qquad \Xi \mathrel{\widetilde{\vdash}} q \Rightarrow t; \hat{P} <:_{\tau} q' \Rightarrow t'; \hat{P}'}{\Xi \mathrel{\widetilde{\vdash}} (\top, q) \Rightarrow t; \underline{R \wedge \overrightarrow{\varphi}} \otimes \hat{P} <:_{\tau} (\top, q') \Rightarrow t'; \underline{Q'} \otimes \hat{P}'} \mathrel{\widetilde{<:_{\tau}}}\text{Const}$$

$\Xi, \overrightarrow{\varphi} \widetilde{\vdash} R <:^{+} Q' \,/\, {}^{(\forall)}W$	Subderivation
$\Xi, \overrightarrow{\varphi} \widetilde{\models} {}^{(\forall)}W$	Subderivation
$\Xi, \overrightarrow{\varphi} \vdash R \leq^{+} Q'$	By i.h.
$\Xi \vdash R \wedge \overrightarrow{\varphi} \leq^{+} Q'$	By $\leq^{+}\wedge$L
$\Xi \widetilde{\vdash} q \Rightarrow t; \hat{P} <:_{\tau} q' \Rightarrow t'; \hat{P}'$	Subderivation
$\Xi \vdash q \Rightarrow t; \hat{P} \leq_{\tau} q' \Rightarrow t'; \hat{P}'$	By i.h.
$\Xi \vdash (\top, q) \Rightarrow t; \underline{R \wedge \overrightarrow{\varphi}} \otimes \hat{P} \leq_{\tau} (\top, q') \Rightarrow t'; \underline{Q'} \otimes \hat{P}'$	By Meas$\leq$Const $\square$

Lemma G.4 (Semidecl. Sub. Complete)**.**

(1) If $\Theta \vdash R \leq^{+} P$ then $\Theta \widetilde{\vdash} R <:^{+} P \,/\, {}^{(\forall)}W$ and $\Theta \widetilde{\models} {}^{(\forall)}W$.

(2) If $\Theta \vdash N \leq^{-} L$ then $\Theta \widetilde{\vdash} N <:^{-} L \,/\, {}^{(\forall)}W$ and $\Theta \widetilde{\models} {}^{(\forall)}W$.

(3) If $\Xi \vdash \alpha; F \leq_{\tau} \beta; G$ then $\Xi \widetilde{\vdash} \alpha; F <:_{\tau} \beta; G$.

(4) If $\Theta \vdash \mathscr{M}'(F') \geq \mathscr{M}(F)$ then $\Theta \widetilde{\vdash} \mathscr{M}'(F') \geq \mathscr{M}(F) \,/\, W$ and $\Theta \widetilde{\models} W$.

Proof. By mutual induction on the structure of the given subtyping/submeasuring derivation. Straightforward. We show one case of part (3).

- **Case**
$$\frac{\Xi \vdash Q \leq^{+} Q' \qquad \Xi \vdash q \Rightarrow t; \hat{P} \leq_{\tau} q' \Rightarrow t'; \hat{P}'}{\Xi \vdash (\top, q) \Rightarrow t; \underline{Q} \otimes \hat{P} \leq_{\tau} (\top, q') \Rightarrow t'; \underline{Q'} \otimes \hat{P}'} \text{ Meas}\leq\text{Const}$$

$\Xi \vdash Q \leq^{+} Q'$ Subderivation

$Q = R \wedge \overrightarrow{\varphi}$ Canonical form of Q

$\Xi \vdash R \wedge \overrightarrow{\varphi} \leq^{+} Q'$ By equality

$\Xi, \overrightarrow{\varphi} \vdash R \leq^{+} Q'$ By inversion

$\Xi, \overrightarrow{\varphi} \widetilde{\vdash} R <:^{+} Q' \;/\; {}^{(\forall)}W_1$ By i.h.

$\Xi, \overrightarrow{\varphi} \widetilde{\models} {}^{(\forall)}W_1$ ″

$\Xi \vdash q \Rightarrow t; \hat{P} \leq_{\tau} q' \Rightarrow t'; \hat{P}'$ Subderivation

$\Xi \widetilde{\vdash} q \Rightarrow t; \hat{P} <:_{\tau} q' \Rightarrow t'; \hat{P}'$ By i.h.

☞ $\Xi \widetilde{\vdash} (\top, q) \Rightarrow t; \underline{R \wedge \overrightarrow{\varphi}} \otimes \hat{P} <:_{\tau} (\top, q') \Rightarrow t'; \underline{Q'} \otimes \hat{P}'$ $\widetilde{<:}_{\tau}$Const

□

Lemma G.5 (Alg. to Semidecl. Chk. (Sub.))**.**

(1) If $\Theta \models {}^{(\forall)}W$ *then* $\Theta \widetilde{\models} {}^{(\forall)}W$.

(2) If $\hat{\Theta}; \cdot \vdash W$ fixInstChk $\dashv \Omega'$ *then* $\hat{\Theta}; \cdot \widetilde{\vdash} W$ fixInstChk $\dashv \Omega'$.

Proof.

(1) By structural induction on the given derivation. In the case for literal subtyping constraints, use Theorem F.3 and Lemma G.4 (Semidecl. Sub. Complete).

(2) Follows from part (1). □

Lemma G.6 (Semidecl. Typing Sound)**.**

(1) If $\Theta; \Gamma \widetilde{\vdash} h \Rightarrow P$ *then* $\Theta; \Gamma \vdash h \Rightarrow P$.

(2) If $\Theta; \Gamma \widetilde{\vdash} g \Rightarrow \uparrow P$ *then* $\Theta; \Gamma \vdash g \Rightarrow \uparrow P$.

(3) *If* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow P \mathbin{/} \chi$ *and* $\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$ *then* $\Theta;\Gamma \vdash v \Leftarrow P$.

(4) *If* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} e \Leftarrow N$ *then* $\Theta;\Gamma \vdash e \Leftarrow N$.

(5) *If* $\Theta;\Gamma;[P] \mathrel{\widetilde{\vdash}} \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$ *then* $\Theta;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$.

(6) *If* $\Theta;\Gamma;[N] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \mathbin{/} \chi$ *and* $\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$ *then* $\Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P$.

Proof. By mutual induction on the structure of the given derivation(s).

Use Lemma G.3 (Semidecl. Sub. Sound). □

Lemma G.7 (Semidecl. Typing Complete)**.**

(1) *If* $\Theta;\Gamma \vdash h \Rightarrow P$ *then* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} h \Rightarrow P$.

(2) *If* $\Theta;\Gamma \vdash g \Rightarrow \uparrow P$ *then* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} g \Rightarrow \uparrow P$.

(3) *If* $\Theta;\Gamma \vdash v \Leftarrow P$ *then there exists* χ *such that* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow P \mathbin{/} \chi$ *and* $\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$.

(4) *If* $\Theta;\Gamma \vdash e \Leftarrow N$ *then* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} e \Leftarrow N$.

(5) *If* $\Theta;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$ *then* $\Theta;\Gamma;[P] \mathrel{\widetilde{\vdash}} \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$.

(6) *If* $\Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P$
then there exists χ *such that* $\Theta;\Gamma;[N] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \mathbin{/} \chi$ *and* $\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$.

Proof. By mutual induction on the structure of the given derivation. Use Lemma G.4 (Semidecl. Sub. Complete). □

Lemma G.8 (Alg. to Semidecl. Chk.)**.**

(1) *If* $\Theta;\Gamma \lhd \chi$ *then* $\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$.

(2) *If* $\hat{\Theta};\Gamma \vdash \chi\ \mathsf{fixInstChk} \dashv \Omega'$ *then* $\hat{\Theta};\Gamma \mathrel{\widetilde{\vdash}} \chi\ \mathsf{fixInstChk} \dashv \Omega'$.

Proof. Similar to proof of Lemma G.5 (Alg. to Semidecl. Chk. (Sub.)) but using Theorem F.4 and Lemma G.7 (Semidecl. Typing Complete) rather than Theorem F.3 and Lemma G.4 (Semidecl. Sub. Complete). □

Lemma G.9 (Equiv. Weakening)**.** *If* $\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W \leftrightarrow {}^{(\forall)}W'$ *and* $\Theta \subseteq \Theta'$ *then* $\Theta' \mathrel{\widetilde{\models}} {}^{(\forall)}W \leftrightarrow {}^{(\forall)}W'$.

Proof. By structural induction on the given constraint equivalence derivation, using Lemma C.41 (Ix.-Level Weakening) as needed. □

Lemma G.10 (Tp./Meas. Equiv. Syn. Subs.)**.** *Assume* $\Theta_0; \cdot \vdash \sigma : \Theta; \cdot$.

(1) If $\Theta \vdash A \equiv^{\pm} B$ *then* $\Theta_0 \vdash [\sigma]A \equiv^{\pm} [\sigma]B$.

(2) If $\Theta \vdash \mathscr{M}'(F) \equiv \mathscr{M}(F)$ *then* $\Theta_0 \vdash [\sigma]\mathscr{M}'(F) \equiv [\sigma]\mathscr{M}(F)$.

(3) If $\Theta \vdash \alpha; F \equiv_\tau \beta; G$, *then* $\Theta_0 \vdash [\sigma]\alpha; [\sigma]F \equiv_\tau [\sigma]\beta; [\sigma]G$.

Proof. Each part is proved separately by induction on the structure of the given equivalence derivation. Part (1) uses part (2). Part (3) uses part (1). Use Lemma C.30 (Prop. Truth Syn. Subs), Lemma C.58 (Ix. Equiv. Syn. Subs), Lemma C.11 (Ix. Id. Subs. Extension), and Lemma C.12 (Value-Det. Substitution) as needed. □

Lemma G.11 (Tp./Meas. Equiv. Reflexive)**.**

(1) If $\Xi \vdash A\ \mathsf{type}[\xi]$ *then* $\Xi \vdash A \equiv^{\pm} A$.

(2) If $\Xi \vdash \mathscr{M}(F)\ \mathsf{msmts}[\xi]$ *then* $\Xi \vdash \mathscr{M}(F) \equiv \mathscr{M}(F)$.

(3) If $\Xi \vdash \alpha : F(\tau) \Rightarrow \tau$, *then* $\Xi \vdash \alpha; F \equiv_\tau \alpha; F$.

Proof. By mutual induction on the structure of the given formation derivation. Use Lemma C.27 (Prop. Truth Equiv. Relation) and Lemma C.34 (Ix. Equiv. Reflexive) as needed. Parts (1) and (2) are mutually recursive. □

Lemma G.12 (Tp./Meas. Equiv. Symmetric)**.**

(1) If $\Theta \vdash A \equiv^{\pm} B$ *then* $\Theta \vdash B \equiv^{\pm} A$ *by a derivation of equal structure and height.*

(2) If $\Theta \vdash \mathscr{M}'(F) \equiv \mathscr{M}(F)$ *then* $\Theta \vdash \mathscr{M}(F) \equiv \mathscr{M}'(F)$
by a derivation of equal structure and height.

(3) If $\Xi \vdash \alpha; F \equiv_\tau \alpha'; F'$ *then* $\Xi \vdash \alpha'; F' \equiv_\tau \alpha; F$
by a derivation of equal structure and height.

Proof. By induction on the structure of the given type or measure equivalence derivation, using Lemma C.27 (Prop. Truth Equiv. Relation) and Lemma C.69 (Ix. Equiv. Symmetric) as needed. Parts (1) and (2) are mutually recursive. □

Lemma G.13 (Equiv. Transitive)**.**

(1) If $\Theta \vdash A \equiv^{\pm} C$ *and* $\Theta \vdash C \equiv^{\pm} B$ *then* $\Theta \vdash A \equiv^{\pm} B$.

(2) If $\Theta \vdash \mathscr{M}'(F) \equiv \mathscr{M}''(F)$ *and* $\Theta \vdash \mathscr{M}''(F) \equiv \mathscr{M}(F)$ *then* $\Theta \vdash \mathscr{M}'(F) \equiv \mathscr{M}(F)$.

(3) If $\Xi \vdash \alpha; F \equiv_\tau \alpha''; F''$ *and* $\Xi \vdash \alpha''; F'' \equiv_\tau \alpha'; F'$ *then* $\Xi \vdash \alpha; F \equiv_\tau \alpha'; F'$.

Proof. By induction on the structure of the derivations. Parts (1) and (2) are mutually recursive. Use Lemma C.27 (Prop. Truth Equiv. Relation) and Lemma C.35 (Ix. Equiv. Transitive). □

Lemma G.14 (Prob. Equiv. Reflexive)**.**
If Θ ctx *and* $\overline{\Theta} \vdash {}^{(\forall)}W$ wf *then* $\Theta \models\!\!\!\sim {}^{(\forall)}W \leftrightarrow {}^{(\forall)}W$.

Proof. By structural induction on ${}^{(\forall)}W$, using Lemma C.34 (Ix. Equiv. Reflexive) and Lemma C.34 (Ix. Equiv. Reflexive) and Lemma G.11 (Tp./Meas. Equiv. Reflexive) as needed. □

Lemma G.15 (Prob. Equiv. Symmetric)**.**
If $\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W_1 \leftrightarrow {}^{(\forall)}W_2$ *then* $\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W_2 \leftrightarrow {}^{(\forall)}W_1$.

Proof. By structural induction on ${}^{(\forall)}W_1$, using Lemma C.69 (Ix. Equiv. Symmetric) and Lemma C.69 (Ix. Equiv. Symmetric) and Lemma G.12 (Tp./Meas. Equiv. Symmetric) as needed. □

Lemma G.16 (Prob. Equiv. Transitive)**.**
If $\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W \leftrightarrow \widetilde{{}^{(\forall)}W}$ *and* $\Theta \mathrel{\widetilde{\models}} \widetilde{{}^{(\forall)}W} \leftrightarrow {}^{(\forall)}W'$ *then* $\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W \leftrightarrow {}^{(\forall)}W'$.

Proof. By structural induction on ${}^{(\forall)}W$, using Lemma C.35 (Ix. Equiv. Transitive) and Lemma G.13 (Equiv. Transitive) as needed. □

Lemma G.17 (Probs. Equiv. Reflexive)**.** *If* $\Theta;\Gamma \vdash \chi$ wf *then* $\Theta \mathrel{\widetilde{\lhd}} \chi \leftrightarrow \chi$.

Proof. By structural induction on χ, using Lemma G.14 (Prob. Equiv. Reflexive) and Lemma G.11 (Tp./Meas. Equiv. Reflexive) as needed. □

Lemma G.18 (Probs. Equiv. Symmetric)**.** *If* $\Theta \mathrel{\widetilde{\lhd}} \chi_1 \leftrightarrow \chi_2$ *then* $\Theta \mathrel{\widetilde{\lhd}} \chi_2 \leftrightarrow \chi_1$.

Proof. By structural induction on χ_1, using Lemma G.15 (Prob. Equiv. Symmetric) and Lemma G.12 (Tp./Meas. Equiv. Symmetric) as needed. □

Lemma G.19 (Probs. Equiv. Transitive)**.**
If $\Theta \mathrel{\widetilde{\lhd}} \chi \leftrightarrow \widetilde{\chi}$ *and* $\Theta \mathrel{\widetilde{\lhd}} \widetilde{\chi} \leftrightarrow \chi'$ *then* $\Theta \mathrel{\widetilde{\lhd}} \chi \leftrightarrow \chi'$.

Proof. By structural induction on χ, using Lemma G.13 (Equiv. Transitive) and Lemma G.16 (Prob. Equiv. Transitive) as needed. □

Lemma G.20 (Equiv. Implies Subtyping)**.**

(1) If $\Theta \vdash A \equiv^{\pm} B$ *then* $\Theta \vdash A \leq^{\pm} B$.

(2) If $\Theta \vdash \mathscr{M}'(F) \equiv \mathscr{M}(F)$ *then* $\Theta \vdash \mathscr{M}'(F) \geq \mathscr{M}(F)$.

Proof. By mutual induction on the height of the given equivalence derivation. Use Lemma C.41 (Ix.-Level Weakening), Lemma C.28 (Assumption), Lemma C.69 (Ix. Equiv. Symmetric), Lemma C.71 (Ctx. Equiv. Compat), and Lemma G.12 (Tp./Meas. Equiv. Symmetric). □

Lemma G.21 (Ctx. Equiv. Compat. (Prob.))**.** *If* $\Theta_1 \vdash \Theta \equiv \Theta'$ *and* $\Theta_1, \Theta, \Theta_2 \mathrel{\widetilde{\models}} {}^{(\forall)}W$ *then* $\Theta_1, \Theta', \Theta_2 \mathrel{\widetilde{\models}} {}^{(\forall)}W$.

Proof. By structural induction on $\Theta_1, \Theta, \Theta_2 \mathrel{\widetilde{\models}} {}^{(\forall)}W$. Use Lemma C.71 (Ctx. Equiv. Compat) as needed. Note $\overline{\Theta} = \overline{\Theta'}$. □

Lemma G.22 ($^{(\forall)}W$ Checking Respects Equiv.)**.**
If $\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W$ *and* $\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W \leftrightarrow {}^{(\forall)}W'$
then $\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W'$ *by a derivation of equal height and structure.*

Proof. By structural induction on $^{(\forall)}W$. We case analyze rules concluding $\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W \leftrightarrow {}^{(\forall)}W'$. The $\widetilde{\models}\leftrightarrow$Prp case uses Lemma C.70 (Equiv. Resp. Prp. Truth). The $\widetilde{\models}\leftrightarrow$IxEq and $\widetilde{\models}\leftrightarrow \boldsymbol{t}$Eq cases use Lemma C.35 (Ix. Equiv. Transitive). The $\widetilde{\models}\leftrightarrow\supset$ case uses Lemma G.21 (Ctx. Equiv. Compat. (Prob.)). The $\widetilde{\models}\forall$ and $\widetilde{\models}\wedge$ cases are straightforward.

- **Case**
$$\frac{\Theta \vdash R_1 \equiv^{+} R_1' \qquad \Theta \vdash P_2 \equiv^{+} P_2'}{\Theta \mathrel{\widetilde{\models}} \underline{R_1 <:^{+} P_2} \leftrightarrow \underline{R_1' <:^{+} P_2'}} \widetilde{\models}\leftrightarrow <:^{\pm}$$
The only possible rule concluding $\Theta \mathrel{\widetilde{\models}} \underline{R_1 <:^{+} P_2}$ is $\widetilde{\models}<:^{+}$.

– **Case** $\dfrac{\Theta \mathrel{\widetilde{\vdash}} R_1 <:^+ P_2 \mathbin{/} \widetilde{(\forall)W} \qquad \Theta \mathrel{\widetilde{\models}} \widetilde{(\forall)W}}{\Theta \mathrel{\widetilde{\models}} \underline{R_1 <:^+ P_2}} \mathrel{\widetilde{\models}}<:^{\pm}$

$\Theta \mathrel{\widetilde{\vdash}} R_1 <:^+ P_2 \mathbin{/} \widetilde{(\forall)W}$	Subderivation
$\Theta \mathrel{\widetilde{\models}} \widetilde{(\forall)W}$	Subderivation
$\Theta \vdash R_1 \leq^+ P_2$	By Lemma G.3 (Semidecl. Sub. Sound)
$\Theta \vdash R_1 \equiv^+ R_1'$	Subderivation
$\Theta \vdash R_1' \equiv^+ R_1$	By Lemma G.12 (Tp./Meas. Equiv. Symmetric)
$\Theta \vdash R_1' \leq^+ R_1$	By Lemma G.20 (Equiv. Implies Subtyping)
$\Theta \vdash P_2 \equiv^+ P_2'$	Subderivation
$\Theta \vdash P_2 \leq^+ P_2'$	By Lemma G.12 (Tp./Meas. Equiv. Symmetric)
$\Theta \vdash R_1' \leq^+ P_2'$	By two uses of Lemma C.63 (Subtyping Transitive)
$\Theta \mathrel{\widetilde{\vdash}} R_1' <:^+ P_2' \mathbin{/} \widetilde{(\forall)W}'$	By Lemma G.4 (Semidecl. Sub. Complete)
$\Theta \mathrel{\widetilde{\models}} \widetilde{(\forall)W}'$	$''$
$\Theta \mathrel{\widetilde{\models}} \underline{R_1' <:^+ P_2'}$	By $\mathrel{\widetilde{\models}}<:^{\pm}$

- **Case** $\mathrel{\widetilde{\models}}<:^-$: Similar to $\mathrel{\widetilde{\models}}<:^+$ case. □

Lemma G.23 (Subtyping Respects Equiv.)**.**

(1) If $\Theta \mathrel{\widetilde{\vdash}} R <:^+ P \mathbin{/} {}^{(\forall)}W$ *and* $\Theta \vdash P \equiv^+ P'$
then $\Theta \mathrel{\widetilde{\vdash}} R <:^+ P' \mathbin{/} {}^{(\forall)}W'$ *and* $\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W \leftrightarrow {}^{(\forall)}W'$ *for some* ${}^{(\forall)}W'$.

(2) If $\Xi \mathrel{\widetilde{\vdash}} \alpha; F <:_\tau \beta; G$ *and* $\Xi \vdash \beta; G \equiv_\tau \beta'; G'$ *then* $\Xi \mathrel{\widetilde{\vdash}} \alpha; F <:_\tau \beta'; G'$.

(3) If $\Theta \mathrel{\widetilde{\vdash}} N <:^{-} L \mathbin{/} {}^{(\forall)}W$ *and* $\Theta \vdash N \equiv^{-} N'$
then $\Theta \mathrel{\widetilde{\vdash}} N' <:^{-} L \mathbin{/} {}^{(\forall)}W'$ *and* $\Theta \mathrel{\widetilde{\models}} {}^{(\forall)}W \leftrightarrow {}^{(\forall)}W'$ *for some* ${}^{(\forall)}W'$.

(4) If $\Theta \mathrel{\widetilde{\vdash}} \mathscr{M}_0(F_0) \geq \mathscr{M}(F) \mathbin{/} W$ *and* $\Theta \vdash \mathscr{M}(F) \equiv \mathscr{M}'(F)$
then $\Theta \mathrel{\widetilde{\vdash}} \mathscr{M}_0(F_0) \geq \mathscr{M}'(F) \mathbin{/} W'$ *and* $\Xi \mathrel{\widetilde{\models}} W \leftrightarrow W'$ *for some* W'.

Moreover, the subtyping/submeasuring derivation does not change in height.

Proof. Each part by structural induction on the given semideclarative derivation. Part (1) uses part (4). Part (2) uses part (1). In some cases, use weakening, Lemma C.34 (Ix. Equiv. Reflexive), Lemma G.11 (Tp./Meas. Equiv. Reflexive), or Lemma G.10 (Tp./Meas. Equiv. Syn. Subs); also Lemma G.22 (${}^{(\forall)}W$ Checking Respects Equiv). □

Lemma G.24 (Equiv. liftapps)**.** *If* $\Xi \vdash \mathscr{M}(F) \equiv \mathscr{M}'(F)$*, then for all* $a_k, F_k, \alpha_k, \tau_k$*,*
$(a_k, (\mathsf{fold}_{F_k}\ \alpha_k)\, \nu\, \boldsymbol{t}_k =_{\tau_k} t_k) \in zip(\overrightarrow{a})(\mathscr{M}(F))$ *for some* $\boldsymbol{t}_k, t_k$ *if and only if*
$(a_k, (\mathsf{fold}_{F_k}\ \alpha_k)\, \nu\, \boldsymbol{t}'_k =_{\tau_k} t'_k) \in zip(\overrightarrow{a})(\mathscr{M}'(F))$ *for some* $\boldsymbol{t}'_k, t'_k$.

Proof. By structural induction on ${}^{\mathsf{d}}\Xi \vdash \mathscr{M}(F) \equiv \mathscr{M}'(F)$. Straightforward. Equivalence of measurements requires alpha-equivalent algebras and functors. The point is liftapps ignores the spines and right-hand side indices of measurements, and only uses the algebras, functors, and sorts of measurements. □

Lemma G.25 (Unroll to Equiv. Type)**.**
If ${}^{\mathsf{d}}\Xi, \Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}(F) \int \mathrel{\hat{=}} {}^{\mathsf{d}}\Theta; R$ *and* ${}^{\mathsf{d}}\Xi \vdash \mathscr{M}(F) \equiv \mathscr{M}'(F)$
then there exist ${}^{\mathsf{d}}\Theta'$ *and* R' *such that* ${}^{\mathsf{d}}\Xi, \Xi \vdash \wr \overrightarrow{\beta}; G; \mathscr{M}'(F) \int \mathrel{\hat{=}} {}^{\mathsf{d}}\Theta'; R'$
and ${}^{\mathsf{d}}\Xi, \Xi \vdash \exists {}^{\mathsf{d}}\Theta'.\, R' \wedge {}^{\mathsf{d}}\Theta' \equiv^{+} \exists {}^{\mathsf{d}}\Theta.\, R \wedge {}^{\mathsf{d}}\Theta$.

Proof. By structural induction on the given unrolling derivation. Lemma G.24 (Equiv. liftapps), Lemma C.58 (Ix. Equiv. Syn. Subs), and Lemma C.59 (Sub. Syn. Subs) in the

$\wr \mathrm{Id} \wr$ case; and Lemma C.37 (Ix. App. Respects Equivalence) and Lemma C.72 (Equal Ix. Equalities) in the $\wr I \wr$ case. □

Lemma G.26 (Typing Respects Equiv.)**.** *Assume* $\Theta \vdash \Gamma \equiv^{+} \Gamma'$.

(1) If $\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$ *and* $\Theta \mathrel{\widetilde{\lhd}} \chi \leftrightarrow \chi'$
then $\Theta;\Gamma' \mathrel{\widetilde{\lhd}} \chi'$ *by a derivation of equal height and structure.*

(2) If $\Theta;\Gamma \mathrel{\widetilde{\vdash}} h \Rightarrow P$ *then there exists* P' *such that* $\Theta \vdash P \equiv^{+} P'$
and $\Theta;\Gamma' \mathrel{\widetilde{\vdash}} h \Rightarrow P'$ *by a derivation of equal height and structure.*

(3) If $\Theta;\Gamma \mathrel{\widetilde{\vdash}} g \Rightarrow \uparrow P$ *then there exists* P' *such that* $\Theta \vdash \uparrow P \equiv^{-} \uparrow P'$
and $\Theta;\Gamma' \mathrel{\widetilde{\vdash}} g \Rightarrow \uparrow P'$ *by a derivation of equal height and structure.*

(4) If $\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow P \mathbin{/} \chi$ *and* $\Theta \vdash P \equiv^{+} P'$ *then there exists* χ' *such that* $\Theta \mathrel{\widetilde{\lhd}} \chi \leftrightarrow \chi'$
and $\Theta;\Gamma' \mathrel{\widetilde{\vdash}} v \Leftarrow P' \mathbin{/} \chi'$ *by a derivation of equal height and structure.*

(5) If $\Theta;\Gamma \mathrel{\widetilde{\vdash}} e \Leftarrow N$ *and* $\Theta \vdash N \equiv^{-} N'$
then $\Theta;\Gamma' \mathrel{\widetilde{\vdash}} e \Leftarrow M$ *by a derivation of equal height and structure.*

(6) If $\Theta;\Gamma;[P] \mathrel{\widetilde{\vdash}} \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$ *and* $\Theta \vdash P \equiv^{+} P'$ *and* $\Theta \vdash N \equiv^{-} N'$
then $\Theta;\Gamma';[P'] \mathrel{\widetilde{\vdash}} \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N'$
by a derivation of equal height and structure.

(7) If $\Theta;\Gamma;[N] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \mathbin{/} \chi$ *and* $\Theta \vdash N \equiv^{-} N'$
then there exist P' *and* χ' *such that* $\Theta \vdash \uparrow P \equiv^{-} \uparrow P'$ *and* $\Theta \mathrel{\widetilde{\lhd}} \chi \leftrightarrow \chi'$
and $\Theta;\Gamma';[N'] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P' \mathbin{/} \chi'$ *by a derivation of equal height and structure.*

Proof. By mutual induction on the structure of the given semideclarative typing derivation.

Use Lemma G.23 (Subtyping Respects Equiv), Lemma G.22 ($^{(\forall)}W$ Checking Respects Equiv), Lemma G.11 (Tp./Meas. Equiv. Reflexive), Lemma G.25 (Unroll to Equiv. Type), Lemma G.10 (Tp./Meas. Equiv. Syn. Subs) Lemma C.41 (Ix.-Level Weakening), Lemma C.71 (Ctx. Equiv. Compat) as needed. □

Lemma G.27 (Equiv. Solutions)**.** *Assume* $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$.

(1) If $\overline{\hat{\Theta}} \rhd t : \kappa$ *then* $\|\hat{\Theta}\| \vdash [\Omega]^2 t = [\Omega]^2[\hat{\Theta}]^2 t$ true.

(2) If $\overline{\hat{\Theta}} \rhd u : \tau$ *then* $\|\hat{\Theta}\| \vdash [\Omega]^2 u \equiv [\Omega]^2[\hat{\Theta}]^2 u : \tau$.

(3) If $\overline{\hat{\Theta}}; [\tau] \rhd \boldsymbol{t} : \kappa$ *then* $\|\hat{\Theta}\|; [\tau] \vdash [\Omega]^2 \boldsymbol{t} \equiv [\Omega]^2[\hat{\Theta}]^2 \boldsymbol{t} : \kappa$.

(4) If $\overline{\hat{\Theta}} \rhd A$ type$[_]$ *then* $\|\hat{\Theta}\| \vdash [\Omega]^2 A \equiv^{\pm} [\Omega]^2[\hat{\Theta}]^2 A$.

(5) If $\overline{\hat{\Theta}} \rhd \mathscr{M}(F)$ msmts$[_]$ *then* $\|\hat{\Theta}\| \vdash [\Omega]^2 \mathscr{M}([\Omega]^2 F) \equiv [\Omega]^2[\hat{\Theta}]^2 \mathscr{M}([\Omega]^2[\hat{\Theta}]^2 F)$.

(6) If $\overline{\hat{\Theta}} \rhd \alpha : F(\tau) \Rightarrow \tau$ *then* $\|\hat{\Theta}\| \vdash [\Omega]^2 \alpha; [\Omega]^2 F \equiv_\tau [\Omega]^2[\hat{\Theta}]^2 \alpha; [\Omega]^2[\hat{\Theta}]^2 F$.

(7) If $\overline{\hat{\Theta}} \vdash {}^{(\forall)}W$ wf *then* $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} [\Omega]^{2(\forall)} W \leftrightarrow [\Omega]^2[\hat{\Theta}]^{2(\forall)} W$.

(8) If $\overline{\hat{\Theta}} \vdash \chi$ Wf *then* $\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} [\Omega]^2 \chi \leftrightarrow [\Omega]^2[\hat{\Theta}]^2 \chi$.

Proof. (1) By structural induction on $\overline{\hat{\Theta}} \rhd t : \kappa$. This part is mutually recursive with part (2). Use Lemma C.27 (Prop. Truth Equiv. Relation) and Lemma C.37 (Ix. App. Respects Equivalence). We show the case where t is an evar $\hat{a}$ for some $\hat{a} \in dom(\hat{\Theta})$. Either $\hat{\Theta}$ does not solve $\hat{a}$ or it does. If it does not, then $[\Omega]^2[\hat{\Theta}]^2 t = [\Omega]^2 t$ and the goal follows by Lemma C.27 (Prop. Truth Equiv. Relation). Suppose $\hat{\Theta}$ does solve $\hat{a}$. Either $\blacktriangleright\hat{a} : \kappa{=}t \in \hat{\Theta}$ or $\hat{a} : \kappa{=}t \in \hat{\Theta}$. In either case, the goal follows by inversion on $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$ and properties of substitution.

(2) By structural induction on $\overline{\hat{\Theta}} \rhd u : \tau$. This part is mutually recursive with part (3). Use part (1).

(3) By structural induction on $\overline{\hat{\Theta}}; [\tau] \rhd t : \kappa$. This part is mutually recursive with part (2).

(4) By structural induction on the given well-formedness derivation using previous parts as needed. This part is mutually recursive with parts (5).

(5) By structural induction on the given well-formedness derivation using previous parts as needed. This part is mutually recursive with part (5).

(6) By structural induction on the given well-formedness derivation using previous parts as needed.

(7) By structural induction on $\overline{\hat{\Theta}} \vdash {}^{(\forall)}W\ \mathsf{wf}$, using previous parts as needed. Use Lemma G.9 (Equiv. Weakening) and Lemma F.12 (Alg. to Decl. WF) in the ${}^{(\forall)}W = \supset$ case.

(8) By structural induction on $\overline{\hat{\Theta}} \vdash \chi\ \mathsf{wf}$ using previous parts as needed. □

Lemma G.28 (Constraint Checking Sandwich)**.**

(1) If $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$ *and* $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} [\Omega]^2({}^{(\forall)}W)$ *and* $\overline{\hat{\Theta}} \vdash {}^{(\forall)}W\ \mathsf{wf}$
then $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} [\Omega]^2[\hat{\Theta}]^2({}^{(\forall)}W)$ *by a derivation of equal height and structure.*

(2) If $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$ *and* $\|\hat{\Theta}\|; \Gamma \mathrel{\widetilde{\lhd}} [\Omega]^2\chi$ *and* $\overline{\hat{\Theta}} \vdash \chi\ \mathsf{Wf}$
then $\|\hat{\Theta}\|; \Gamma \mathrel{\widetilde{\lhd}} [\Omega]^2[\hat{\Theta}]^2\chi$ *by a derivation of equal height and structure.*

Proof. By Lemma G.27 (Equiv. Solutions) and Lemma G.22 (${}^{(\forall)}W$ Checking Respects Equiv). or Lemma G.26 (Typing Respects Equiv). □

Lemma G.29 (Only Evars Solved Later Remain Unsolved)**.**

If $\overline{\hat{\Theta}} \vdash W\ \mathsf{wf}[\xi]$ *and* $[\hat{\Theta}]^2 W = W$

and $cl(\xi - \|[\xi]\hat{\Theta}\|)(\emptyset) = unsol([\xi]\hat{\Theta})$ *and* $[\xi]\hat{\Theta} \vdash W\ \mathsf{fixInst} \dashv \hat{\Theta}'$

then $unsol(\hat{\Theta}') = \blacktriangleright unsol([\xi]\hat{\Theta})$.

Proof. Under the given the conditions, there's a bijection

between newly solved evars in $\hat{\Theta}'$ relative to $\hat{\Theta}$ and non-$\blacktriangleright$ unsolved evars of $[\xi]\hat{\Theta}$:

$$dom(sol(\hat{\Theta}') - sol([\xi]\hat{\Theta})) = dom(unsol([\xi]\hat{\Theta}) - \blacktriangleright([\xi]\hat{\Theta}))$$

$$\begin{aligned}
unsol(\hat{\Theta}') &= unsol([\xi]\hat{\Theta}) - sol(\hat{\Theta}') && \text{As } \hat{\Theta} \longrightarrow \hat{\Theta}' \\
&= unsol([\xi]\hat{\Theta}) - (sol(\hat{\Theta}') - sol([\xi]\hat{\Theta})) && sol([\xi]\hat{\Theta}) \cap unsol([\xi]\hat{\Theta}) = \emptyset \\
&= unsol([\xi]\hat{\Theta}) - (unsol([\xi]\hat{\Theta}) - \blacktriangleright([\xi]\hat{\Theta})) && \text{By equality} \\
&= \blacktriangleright unsol([\xi]\hat{\Theta}) && \text{Straightforward}
\end{aligned}$$

□

Lemma G.30 (True Inst. Preserves Relaxed Ext.)**.**

Assume $\|\Omega\|, \blacktriangleright\Omega, \hat{\Theta} - \|\Omega\| - \blacktriangleright\Omega \xrightarrow{\mathsf{SMT}} \Omega'$.

(1) If $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} [\Omega']^2({}^{(\forall)}W)$ *and* $\hat{\Theta} \vdash {}^{(\forall)}W\ \mathsf{Inst} \dashv \hat{\Theta}''$

and $\hat{\Theta}'' \longrightarrow \Omega$ *and* $[\hat{\Theta}]({}^{(\forall)}W) = {}^{(\forall)}W$

then $\|\hat{\Theta}''\|, \blacktriangleright\Omega, \hat{\Theta}'' - \|\hat{\Theta}''\| - \blacktriangleright\hat{\Theta}'' \xrightarrow{\mathsf{SMT}} \Omega'$.

(2) If $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} [\Omega']^2 W$ *and* $\hat{\Theta} \vdash W\ \mathsf{fixInst} \dashv \hat{\Theta}_0$

and $\hat{\Theta}_0 \longrightarrow \Omega$ *and* $unsol(\hat{\Theta}_0) = \blacktriangleright unsol(\hat{\Theta})$ *and* $[\hat{\Theta}]^2 W = W$

then $\Omega \xrightarrow{\mathsf{SMT}} \Omega'$.

Proof. Each part by structural induction on the given instantiation derivation. Part (2) uses part (1). Use Lemma F.11 (Inst. Extends), Lemma F.6 (Ext. Transitive), Lemma C.27

(Prop. Truth Equiv. Relation), Lemma G.27 (Equiv. Solutions), Lemma G.28 (Constraint Checking Sandwich), and the fact that the Inst judgment does not touch later evars. □

Lemma G.31 ($\mathsf{fixInstChk}$ Unapply)**.** *If* $\hat{\Theta}$ $\mathsf{present}$ *and* $[\hat{\Theta}]\chi = \chi$
and $\overline{\hat{\Theta}} \vdash \chi\ \mathsf{Wf}[\xi]$ *and* $cl(\xi - \|[\xi]\hat{\Theta}\|)(\emptyset) = unsol([\xi]\hat{\Theta})$
and $\hat{\Theta} \longrightarrow \hat{\Theta}'$ *and* $[\xi]\hat{\Theta}'; \Gamma \mathrel{\widetilde{\vdash}} [\hat{\Theta}']\chi\ \mathsf{fixInstChk} \dashv \Omega'$
then there exists a derivation $[\xi]\hat{\Theta}; \Gamma \mathrel{\widetilde{\vdash}} \chi\ \mathsf{fixInstChk} \dashv \Omega$ *such that* $\Omega \xrightarrow{\mathsf{SMT}} \Omega'$.

Proof.

$[\xi]\hat{\Theta}'; \Gamma \mathrel{\widetilde{\vdash}} [\hat{\Theta}']\chi\ \mathsf{fixInstChk} \dashv \Omega'$	Given
$[\xi]\hat{\Theta}' \vdash \lfloor [\hat{\Theta}']\chi \rfloor\ \mathsf{fixInst} \dashv \hat{\Theta}''$	By inversion
$\mathsf{select}\ \overrightarrow{W_o'}\ \mathsf{from}\ [\hat{\Theta}'']^2 \lfloor [\hat{\Theta}']\chi \rfloor$	$''$
$\hat{\Theta}'' \vdash \mathbb{\Lambda}\overrightarrow{W_o'}\ \mathsf{Inst}\blacktriangleright \dashv \Omega'$	$''$
$\|\hat{\Theta}'\|; \Gamma \mathrel{\widetilde{\lhd}} [\Omega']^2[\hat{\Theta}']\chi$	$''$

$\hat{\Theta} \longrightarrow \hat{\Theta}'$	Given
$[\xi]\hat{\Theta} \longrightarrow [\xi]\hat{\Theta}'$	As $(\hat{\Theta}, \hat{\Theta}')$ $\mathsf{present}$ so all solutions ground
$[\xi]\hat{\Theta}' \longrightarrow \hat{\Theta}''$	By Lemma F.11 (Inst. Extends)
$\hat{\Theta}'' \longrightarrow \Omega'$	By Lemma F.11 (Inst. Extends)
$[\xi]\hat{\Theta}' \longrightarrow \Omega'$	By Lemma F.6 (Ext. Transitive)
$[\xi]\hat{\Theta} \longrightarrow \Omega'$	By Lemma F.6 (Ext. Transitive)
$[\xi]\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega'$	By Lemma F.4 (Extension Sound)
$\mathsf{select}\ \overrightarrow{W_o'}\ \mathsf{from}\ [\hat{\Theta}'']^2 \lfloor \chi \rfloor$	As $[\hat{\Theta}'']^2 \lfloor [\hat{\Theta}']\chi \rfloor = [\hat{\Theta}'']^2 \lfloor \chi \rfloor$
$\hat{\Theta}'' \vdash \mathbb{\Lambda}\overrightarrow{W_o'}\ \mathsf{Inst}\blacktriangleright \dashv \Omega'$	Above
$\|\hat{\Theta}\|; \Gamma \mathrel{\widetilde{\lhd}} [\Omega']^2\chi$	As $\|\hat{\Theta}\| = \|\hat{\Theta}'\|$ and $[\Omega']^2[\hat{\Theta}']\chi = [\Omega']^2\chi$

$[\xi]\hat{\Theta} \vdash \lfloor \chi \rfloor$ fixInst $\dashv \hat{\Theta}_0$ By Lemma F.20 (Inst. Succeeds)

$unsol(\hat{\Theta}_0) = \blacktriangleright unsol([\xi]\hat{\Theta})$ By Lemma G.29 (Only Evars Solved Later Remain Unsolved)

Let $\lfloor \chi \rfloor \rightsquigarrow X$ where $\rightsquigarrow$ is (re)defined locally in this proof as follows:

$$\frac{^{(\forall)}W \rightsquigarrow X \qquad ^{(\forall)}W' \rightsquigarrow X'}{^{(\forall)}W \wedge {}^{(\forall)}W' \rightsquigarrow X \cup X'} \qquad \frac{(_ \wedge t \equiv_\kappa \hat{a}) \in \overrightarrow{W}}{\bigvee \overrightarrow{W} \rightsquigarrow \{\bigvee \overrightarrow{W}\}}$$

$$\frac{^{(\forall)}W \neq - \wedge - \text{ and } {}^{(\forall)}W \neq \bigvee \overrightarrow{W} \text{ with } _ \wedge _ \equiv__ \hat{a} \in \overrightarrow{W}}{^{(\forall)}W \rightsquigarrow \emptyset}$$

For each $\hat{a} \in unsol(\hat{\Theta}_0)$,
pick exactly one $\bigvee \overrightarrow{W} \in X$ and exactly one $\boldsymbol{u} \equiv_{[\tau]} \boldsymbol{t} \wedge u \equiv_\kappa \hat{a}$ in $\overrightarrow{W}$
such that $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} \boldsymbol{u} \equiv_{[\tau]} [\Omega']^2 \boldsymbol{t} \wedge u \equiv_\kappa [\Omega']^2 \hat{a}$
(at least one such constraint exists, by inversion on $\|\hat{\Theta}\|; \Gamma \mathrel{\widetilde{\lhd}} [\Omega']^2 \chi$)
and collect each $u \equiv_\kappa \hat{a}$ in $\overrightarrow{W_o}$. By construction, the following three lines hold:

$$\text{select } \overrightarrow{W_o} \text{ from } [\hat{\Theta}_0]^2 \lfloor \chi \rfloor$$

$$\hat{\Theta}_0 \vdash \wedge \overrightarrow{W_o} \text{ Inst}\blacktriangleright \dashv \Omega$$

$$\|\hat{\Theta}\|, \blacktriangleright \Omega \xrightarrow{\text{SMT}} \|\hat{\Theta}\|, \blacktriangleright \Omega'$$

☞ $\Omega \xrightarrow{\text{SMT}} \Omega'$ By Lemma G.30 (True Inst. Preserves Relaxed Ext)

$\|\hat{\Theta}\|; \Gamma \mathrel{\widetilde{\lhd}} [\Omega']^2 [\Omega]^2 \chi$ By Lemma G.28 (Constraint Checking Sandwich)

$\|\hat{\Theta}\|; \Gamma \mathrel{\widetilde{\lhd}} [\Omega]^2 \chi$ Solutions already applied

☞ $\hat{\Theta}; \Gamma \mathrel{\widetilde{\vdash}} \chi$ fixInstChk $\dashv \Omega$ By rule

□

Appendix G.2 Dependency Mediation

(We don't need most of the machinery here for algorithmic subtyping completeness, it's mainly for algorithmic typing completeness.)

$\boxed{\Delta \vdash \xi' \angle \xi}$ Presupposes $FV(\xi') \cup FV(\xi)$ is a set of evars and $FV(\xi') \cap dom(\Delta) = \emptyset$

$$\frac{\begin{array}{c} dom(\Delta) \subseteq cl(\xi)(FV(\xi) - \Delta) \\ \text{for all } \widetilde{\mathfrak{D}} \to \hat{c} \in \xi' \text{ there exists } \widetilde{\mathfrak{B}} \to \hat{c} \in \xi \text{ such that } \widetilde{\mathfrak{B}} \subseteq \widetilde{\mathfrak{D}} \cup \Delta \text{ and } \widetilde{\mathfrak{B}} \cap \Delta \subseteq cl(\xi)(\widetilde{\mathfrak{D}}) \end{array}}{\Delta \vdash \xi' \angle \xi}$$

Figure G.1: Dependency mediation

Lemma G.32 (Admissible Premise)**.** *If $\Delta \vdash \xi' \angle \xi$ then $cl(\xi')(\emptyset) \subseteq cl(\xi)(\emptyset)$.*

Proof. It suffices to show (by induction on n) $cl^n(\xi')(\emptyset) \subseteq cl(\xi)(\emptyset)$ for all $n \in \mathbb{N}$.

If $n = 0$ then $cl^n(\xi')(\emptyset) = cl^0(\xi')(\emptyset) = \emptyset \subseteq cl(\xi)(\emptyset)$.

Assume $n = m + 1$.

Suppose $\hat{a} \in cl^n(\xi')(\emptyset)$.

By definition of cl^{m+1}, we know $\hat{a} \in cl^m(\xi')(\emptyset) \cup \{c \mid \mathfrak{A} \subseteq cl^m(\xi')(\emptyset)\}$.

If $\hat{a} \in cl^m(\xi')(\emptyset)$ then $\hat{a} \in cl(\xi)(\emptyset)$ by the induction hypothesis.

Assume $\hat{a} \notin cl^m(\xi')(\emptyset)$.

Then there exists $\mathfrak{D} \to \hat{a} \in \xi'$ such that $\mathfrak{D} \subseteq cl^m(\xi')(\emptyset)$.

By inversion on $\Delta \vdash \xi' \angle \xi$,

$$\Delta \subseteq cl(\xi)(FV(\xi) - \Delta)$$

and (*):

for all $\widetilde{\mathfrak{D}} \to \hat{c} \in \xi'$ there exists $\widetilde{\mathfrak{B}} \to \hat{c} \in \xi$ such that $\widetilde{\mathfrak{B}} \subseteq \widetilde{\mathfrak{D}} \cup \Delta$ and $\widetilde{\mathfrak{B}} \cap \Delta \subseteq cl(\xi)(\widetilde{\mathfrak{D}})$.

$$
\begin{array}{ll}
\mathfrak{D}'{\to}\hat{a} \in \xi & \text{By (*)} \\
\mathfrak{D}' \subseteq \mathfrak{D} \cup \Delta & '' \\
\mathfrak{D}' \cap \Delta \subseteq cl(\xi)(\mathfrak{D}) & ''
\end{array}
$$

$$
\begin{array}{ll}
\mathfrak{D}' \cap \mathfrak{D} \subseteq \mathfrak{D} & \\
\quad \subseteq cl^m(\xi')(\emptyset) & \text{Above} \\
\quad \subseteq cl(\xi)(\emptyset) & \text{By i.h.}
\end{array}
$$

$$
\begin{array}{ll}
\mathfrak{D}' \cap \Delta \subseteq cl(\xi)(\mathfrak{D}) & \text{Above} \\
\quad = cl(\xi)(\emptyset) & \text{By Lemma B.5 (Weaken } cl\text{)}
\end{array}
$$

$$
\begin{array}{ll}
\mathfrak{D}' = \mathfrak{D}' \cap (\mathfrak{D} \cup \Delta) & \text{As } \mathfrak{D}' \subseteq \mathfrak{D} \cup \Delta \\
\quad = (\mathfrak{D}' \cap \mathfrak{D}) \cup (\mathfrak{D}' \cap \Delta) & \\
\quad \subseteq cl(\xi)(\emptyset) &
\end{array}
$$

$$
\begin{array}{ll}
\hat{a} \in cl(\xi \cup \mathfrak{D}'{\to}\hat{a})(\emptyset) & \text{By Lemma B.6 (} cl \text{ transitive)} \\
\quad = cl(\xi)(\emptyset) & \text{As } \mathfrak{D}'{\to}\hat{a} \in \xi
\end{array}
$$

□

Lemma G.33 (No Δ Mediates Reflexive)**.** *If $FV(\xi)$ is a set of evars then $\cdot \vdash \xi \angle \xi$.*

Proof. Straightforward. □

Lemma G.34 (Compose Mediates)**.**

If $dom(\Delta_1) \cap dom(\Delta_2) = \emptyset$

and $dom(\Delta_1, \Delta_2) \cap FV(\xi_1' \cup \xi_2') = \emptyset$ and $\Delta_1 \cap FV(\xi_2) = \emptyset = \Delta_2 \cap FV(\xi_1)$

and $\Delta_1 \vdash \xi_1' \angle \xi_1$ *and* $\Delta_2 \vdash \xi_2' \angle \xi_2$

then $\Delta_1, \Delta_2 \vdash \xi_1' \cup \xi_2' \angle \xi_1 \cup \xi_2$.

Proof. For all $k \in \{1,2\}$, by inversion on $\Delta_k \vdash \xi_k' \angle \xi_k$, we know that

$dom(\Delta_k) \subseteq cl(\xi_k)(FV(\xi_k) - \Delta_k)$

and (*): for all $\widetilde{\mathfrak{D}} \to \hat{c} \in \xi_k'$ there exists $\widetilde{\mathfrak{B}} \to \hat{c} \in \xi_k$ such that $\widetilde{\mathfrak{B}} \subseteq \widetilde{\mathfrak{D}} \cup \Delta_k$ and $\widetilde{\mathfrak{B}} \cap \Delta_k \subseteq cl(\xi_k)(\widetilde{\mathfrak{D}})$

We show the two conditions

needed to apply the rule to obtain the goal $\Delta_1, \Delta_2 \vdash \xi_1' \cup \xi_2' \angle \xi_1 \cup \xi_2$.

Suppose $\mathfrak{D} \to c \in (\xi_1' \cup \xi_2')$.

Then $\mathfrak{D} \to c \in \xi_j'$ for some $j \in \{1,2\}$.

By (*), there exists $\mathfrak{D}' \to c \in \xi_j$ such that $\mathfrak{D}' \subseteq \mathfrak{D} \cup \Delta_j$ and $\mathfrak{D}' \cap \Delta_j \subseteq cl(\xi_j)(\mathfrak{D})$.

$$\begin{aligned}
\mathfrak{D}' \to c &\in \xi_j && \text{Above} \\
&\subseteq \xi_1 \cup \xi_2
\end{aligned}$$

$$\begin{aligned}
\mathfrak{D}' &\subseteq \mathfrak{D} \cup \Delta_j && \text{Above} \\
&\subseteq \mathfrak{D} \cup (\Delta_1, \Delta_2)
\end{aligned}$$

$$\begin{aligned}
\mathfrak{D}' \cap \Delta_{3-j} &= \emptyset && \text{As } \mathfrak{D}' \subseteq \mathfrak{D} \cup \Delta_j \text{ and } (\mathfrak{D} \cup \Delta_j) \cap \Delta_{3-j} = \emptyset \\
\mathfrak{D}' \cap (\Delta_1, \Delta_2) &= \mathfrak{D}' \cap \Delta_j \\
&\subseteq cl(\xi_j)(\mathfrak{D}) && \text{Above} \\
&\subseteq cl(\xi_1 \cup \xi_2)(\mathfrak{D}) && cl(-)(\mathscr{O}) \text{ monotone}
\end{aligned}$$

Therefore,

$\forall \widetilde{\mathfrak{D}} \to \hat{c} \in \xi_1' \cup \xi_2', \exists \widetilde{\mathfrak{B}} \to \hat{c} \in \xi_1 \cup \xi_2$ s.t. $\widetilde{\mathfrak{B}} \subseteq \widetilde{\mathfrak{D}} \cup (\Delta_1, \Delta_2)$ and $\widetilde{\mathfrak{B}} \cap (\Delta_1, \Delta_2) \subseteq cl(\xi_1 \cup \xi_2)(\widetilde{\mathfrak{D}})$

- We show the other condition:

$$\begin{aligned} FV(\xi_1) - \Delta_1 &= FV(\xi_1) - (\Delta_1, \Delta_2) && FV(\xi_1) \cap \Delta_2 = \emptyset \\ &\subseteq (FV(\xi_1) - (\Delta_1, \Delta_2)) \cup (FV(\xi_2) - (\Delta_1, \Delta_2)) \\ &= (FV(\xi_1) \cup FV(\xi_2)) - (\Delta_1, \Delta_2) \\ &= FV(\xi_1 \cup \xi_2) - (\Delta_1, \Delta_2) \end{aligned}$$

$$FV(\xi_2) - \Delta_2 \subseteq FV(\xi_1 \cup \xi_2) - (\Delta_1, \Delta_2) \qquad \text{Similarly}$$

$$\begin{aligned} (\Delta_1, \Delta_2) &\subseteq cl(\xi_1)(FV(\xi_1) - \Delta_1) \cup cl(\xi_2)(FV(\xi_2) - \Delta_2) && \Delta_k \subseteq cl(\xi_k)(FV(\xi_k) - \Delta_k) \\ &\subseteq cl(\xi_1 \cup \xi_2)(FV(\xi_1) - \Delta_1) \cup cl(\xi_1 \cup \xi_2)(FV(\xi_2) - \Delta_2) && cl(-)(\mathscr{O}) \text{ monotone (x2)} \\ &\subseteq cl(\xi_1 \cup \xi_2)(FV(\xi_1 \cup \xi_2) - (\Delta_1, \Delta_2)) \\ &\quad \cup cl(\xi_1 \cup \xi_2)(FV(\xi_1 \cup \xi_2) - (\Delta_1, \Delta_2)) && cl(\mathscr{O})(-) \text{ monotone (x2)} \\ &= cl(\xi_1 \cup \xi_2)(FV(\xi_1 \cup \xi_2) - (\Delta_1, \Delta_2)) && \cup \text{ idempotent} \end{aligned}$$

□

Lemma G.35 (Main Complete)**.**

(1) *If* $\hat{\Theta} \vdash R' <:^{+} Q \;/\; W$ *and* R' ground *and* $\overline{\hat{\Theta}} \vdash Q$ type$[\xi_Q]$ *and* $[\hat{\Theta}]Q = Q$
then $\overline{\hat{\Theta}} \vdash W$ wf$[\xi_W]$ *and* $\cdot \vdash \xi_Q - \|\hat{\Theta}\| \angle \xi_W - \|\hat{\Theta}\|$*;*
moreover, if $Q = R$ *then* $\blacktriangleright(pos(\xi_Q - \|\hat{\Theta}\|)) \subseteq \xi_W - \|\hat{\Theta}\|$.

(2) *If* $\overline{\hat{\Theta}} \rhd \mathscr{M}(F)$ type$[\xi']$ *and* $\nexists x.\, v = \overrightarrow{\text{inj}_{k_i}}^{i}\left(\overrightarrow{\langle -_j, - \rangle}^{j} x\right)$
and $\overset{\mathsf{d}}{\div}\hat{\Theta} \rhd \overrightarrow{\wr \beta}; G; \mathscr{M}(F) \wr \triangleq {}^{\mathsf{d}}\Theta; R$ *and* $\hat{\Theta}; \Gamma \vdash v \Leftarrow \exists {}^{\mathsf{d}}\Theta.\, R \wedge {}^{\mathsf{d}}\Theta \;/\; \chi \dashv \Delta$
and $[\hat{\Theta}](F, \mathscr{M}, G, \overrightarrow{\beta}) = (F, \mathscr{M}, G, \beta)$
then $\overline{\hat{\Theta}}, \Delta \vdash \lfloor \chi \rfloor$ wf$[\xi_\chi]$ *and* $\Delta \vdash \xi' - \|\hat{\Theta}\| \angle \xi_\chi - \|\hat{\Theta}\|$.

(3) *If* $\hat{\Theta}; \Gamma \vdash v \Leftarrow Q \;/\; \chi \dashv \Delta$ *and* $\overline{\hat{\Theta}} \rhd Q$ type$[\xi_Q]$ *and* $[\hat{\Theta}]Q = Q$
then $\overline{\hat{\Theta}}, \Delta \vdash \lfloor \chi \rfloor$ wf$[\xi_\chi]$ *and* $\Delta \vdash \xi_Q - \|\hat{\Theta}\| \angle \xi_\chi - \|\hat{\Theta}\|$.

(4) *If* $\hat{\Theta};\Gamma;[M] \vdash s \Rightarrow \uparrow P \mathbin{/} \chi \dashv \Delta$
and $\overline{\hat{\Theta}} \rhd M\ \mathsf{type}[\xi_M]$ *and* $[\hat{\Theta}]M = M$
then $\overline{\hat{\Theta}}, \Delta \vdash \lfloor \chi \rfloor\ \mathsf{wf}[\xi_\chi]$ *and* $\Delta \vdash \xi_M - \|\hat{\Theta}\| \angle \xi_\chi - \|\hat{\Theta}\|$.

Proof. Parts (2) and (3) are mutually recursive. It is easy to check the presuppositions of $\Delta \vdash _ \angle _$.

(1) By induction on the structure of the given subtyping derivation. Straightforward. Use Lemma G.34 in the $<:^+\times$ case, similar to Alg$\Leftarrow\times$ case of part (3). In the $<:^+\wedge$R case, use Lemma G.34 (Compose Mediates) with Lemma G.33 (No Δ Mediates Reflexive).

(2) By lexicographic induction, first on v structure, second on unrolling height.

Let $\Xi = \overline{{}^{\mathsf{d}}\Theta}$.

Let $\hat{\Xi} = \widehat{\Xi}$.

Let $\Psi = [\hat{\Xi}/\Xi]({}^{\mathsf{d}}\Theta - \overline{{}^{\mathsf{d}}\Theta})$.

- **Case**

$$\frac{\overrightarrow{\beta} \circ \mathsf{inj}_1 \doteq \overrightarrow{\beta_1} \qquad {}^{\mathsf{d}}\!\div\hat{\Theta} \rhd \wr \overrightarrow{\beta_1}; G_1; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta_1; R_1 \qquad \overrightarrow{\beta} \circ \mathsf{inj}_2 \doteq \overrightarrow{\beta_2} \qquad {}^{\mathsf{d}}\!\div\hat{\Theta} \rhd \wr \overrightarrow{\beta_2}; G_2; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta_2; R_2}{{}^{\mathsf{d}}\!\div\hat{\Theta} \rhd \wr \overrightarrow{\beta}; G_1 \oplus G_2; \mathscr{M}(F) \int \doteq \cdot; (\exists^{\mathsf{d}}\Theta_1.\,(R_1 \wedge {}^{\mathsf{d}}\Theta_1)) + (\exists^{\mathsf{d}}\Theta_2.\,(R_2 \wedge {}^{\mathsf{d}}\Theta_2))}\ \mathrm{Alg}\wr\oplus\int$$

  - **Case**

$$\frac{\hat{\Theta};\Gamma \vdash v_0 \Leftarrow \exists^{\mathsf{d}}\Theta_k.\,(R_k \wedge {}^{\mathsf{d}}\Theta_k) \mathbin{/} \chi \dashv \Delta}{\hat{\Theta};\Gamma \vdash \mathsf{inj}_k\, v_0 \Leftarrow (\exists^{\mathsf{d}}\Theta_1.\,(R_1 \wedge {}^{\mathsf{d}}\Theta_1)) + (\exists^{\mathsf{d}}\Theta_2.\,(R_2 \wedge {}^{\mathsf{d}}\Theta_2)) \mathbin{/} \chi \dashv \Delta}\ \mathrm{Alg}\Leftarrow +_k$$

$\overset{\mathsf{d}}{\div}\hat{\Theta} \rhd \wr \overrightarrow{\beta_k}; G_k; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta_k; R_k$ Subderivation

$\hat{\Theta}; \Gamma \vdash v_0 \Leftarrow \exists {}^{\mathsf{d}}\Theta_k.\, (R_k \wedge {}^{\mathsf{d}}\Theta_k) \,/\, \chi \dashv \Delta$ Premise

$\nexists x.\, \mathrm{inj}_k\, v_0 = \overrightarrow{\mathrm{inj}_{k_i}}^i \left(\overrightarrow{\langle __j, - \rangle}^j x \right)$ Given

$\nexists x.\, v_0 = \overrightarrow{\mathrm{inj}_{k_i}}^i \left(\overrightarrow{\langle __j, - \rangle}^j x \right)$ Straightforward

☞ $\overline{\hat{\Theta}}, \Delta \vdash \lfloor \chi \rfloor\ \mathsf{wf}[\xi_\chi]$ By i.h.

☞ $\Delta \vdash \xi' - \|\hat{\Theta}\| \angle \xi_\chi - \|\hat{\Theta}\|$ ″

- **Case**

$$\frac{\begin{array}{c}({}^{\mathsf{d}}\Xi' \text{ may be } \cdot \text{ and } \overrightarrow{\varphi} \text{ may be } \cdot) \\ \overrightarrow{\beta} \rightsquigarrow \overrightarrow{\beta'} \\ {}^{\mathsf{d}}\Xi', \overset{\mathsf{d}}{\div}\hat{\Theta} \vdash \wr \overrightarrow{\beta'}; \hat{P}; \mathscr{M}(F) \int \doteq {}^{\mathsf{d}}\Theta'_0; R'\end{array}}{\overset{\mathsf{d}}{\div}\hat{\Theta} \rhd \wr \overrightarrow{\beta}; \underline{\exists {}^{\mathsf{d}}\Xi'.\, R_Q \wedge \overrightarrow{\varphi}} \otimes \hat{P}; \mathscr{M}(F) \int \doteq \underbrace{{}^{\mathsf{d}}\Xi', {}^{\mathsf{d}}\Theta'_0, \overrightarrow{\varphi}}_{{}^{\mathsf{d}}\Theta}; \underbrace{R_Q \times R'}_{R}} \mathrm{Alg}\wr\mathrm{Const}\int$$

We assume $({}^{\mathsf{d}}\Xi', \overline{{}^{\mathsf{d}}\Theta'_0}) \neq \cdot$. The empty case is similar.

Let ${}^{\mathsf{d}}\Theta_0 = {}^{\mathsf{d}}\Theta'_0, \overrightarrow{\varphi}$.

- **Case**

$$\frac{\hat{\Theta}, \widehat{{}^{\mathsf{d}}\Xi', \overline{{}^{\mathsf{d}}\Theta_0}}; \Gamma \vdash v \Leftarrow [\widehat{{}^{\mathsf{d}}\Xi', \overline{{}^{\mathsf{d}}\Theta_0}} / {}^{\mathsf{d}}\Xi', \overline{{}^{\mathsf{d}}\Theta_0}](R \wedge ({}^{\mathsf{d}}\Xi', {}^{\mathsf{d}}\Theta_0)) \,/\, \chi \dashv \Delta_0}{\hat{\Theta}; \Gamma \vdash v \Leftarrow \exists {}^{\mathsf{d}}\Xi', {}^{\mathsf{d}}\Theta_0.\, R \wedge ({}^{\mathsf{d}}\Xi', {}^{\mathsf{d}}\Theta_0) \,/\, \chi \dashv \underbrace{\widehat{{}^{\mathsf{d}}\Xi', \overline{{}^{\mathsf{d}}\Theta_0}}, \Delta_0}_{\Delta}} \mathrm{Alg}\Leftarrow\exists$$

$\hat{\Theta}, \widehat{{}^{\mathsf{d}}\Xi', \overline{{}^{\mathsf{d}}\Theta_0}}; \Gamma \vdash v \Leftarrow [\widehat{{}^{\mathsf{d}}\Xi', \overline{{}^{\mathsf{d}}\Theta_0}} / {}^{\mathsf{d}}\Xi', \overline{{}^{\mathsf{d}}\Theta_0}](R \wedge ({}^{\mathsf{d}}\Xi', {}^{\mathsf{d}}\Theta_0)) \,/\, \chi \dashv \Delta_0$ Premise

$\hat{\Theta}, \widehat{{}^{\mathsf{d}}\Xi', \overline{{}^{\mathsf{d}}\Theta_0}}; \Gamma \vdash v \Leftarrow [\widehat{{}^{\mathsf{d}}\Xi', \overline{{}^{\mathsf{d}}\Theta_0}} / {}^{\mathsf{d}}\Xi', \overline{{}^{\mathsf{d}}\Theta_0}](R \wedge {}^{\mathsf{d}}\Theta_0) \,/\, \chi \dashv \Delta_0$ By def. $\wedge$

$\hat{\Theta}, \widehat{{}^{\mathsf{d}}\Xi', \overline{{}^{\mathsf{d}}\Theta_0}}; \Gamma \vdash v \Leftarrow [\widehat{{}^{\mathsf{d}}\Xi', \overline{{}^{\mathsf{d}}\Theta_0}} / {}^{\mathsf{d}}\Xi', \overline{{}^{\mathsf{d}}\Theta_0}] R \,/\, \chi_0 \dashv \Delta_0$ By inversion on Alg$\Leftarrow\wedge$

$\chi = \Psi, \chi_0$ ″

By equality and def. $[-]-$ and $FV(R_Q) \cap dom({}^{\mathsf{d}}\Theta_0) = \emptyset$,

$\hat{\Theta}, {}^{\mathsf{d}}\widehat{\Xi', {}^{\mathsf{d}}\Theta_0}; \Gamma \vdash v \Leftarrow [\widehat{{}^{\mathsf{d}}\Xi'}/{}^{\mathsf{d}}\Xi]R_Q \times [\widehat{{}^{\mathsf{d}}\Xi', {}^{\mathsf{d}}\Theta_0}/{}^{\mathsf{d}}\Xi', \overline{{}^{\mathsf{d}}\Theta_0}]R' \;/\; \chi_0 \dashv \Delta_0$

$\nexists x.\, v = \overrightarrow{\mathrm{inj}_{k_i}}^{i}\left(\overrightarrow{\langle __j, - \rangle}^{j}\, x\right)$	Given
v not a var.	Straightforward
$\hat{\Theta}, {}^{\mathsf{d}}\widehat{\Xi', {}^{\mathsf{d}}\Theta_0}; \Gamma \vdash v_1 \Leftarrow [\widehat{{}^{\mathsf{d}}\Xi'}/{}^{\mathsf{d}}\Xi']R_Q \;/\; \chi_1 \dashv \Delta_1$	By inversion on Alg$\Leftarrow\times$
$\hat{\Theta}, {}^{\mathsf{d}}\widehat{\Xi', {}^{\mathsf{d}}\Theta_0}; \Gamma \vdash v_2 \Leftarrow [\widehat{{}^{\mathsf{d}}\Xi', {}^{\mathsf{d}}\Theta_0}/{}^{\mathsf{d}}\Xi', \overline{{}^{\mathsf{d}}\Theta_0}]R' \;/\; \chi_2 \dashv \Delta_2$	$''$
$v = \langle v_1, v_2 \rangle$	$''$
$\chi_0 = \chi_1, \chi_2$	$''$
$\Delta_0 = \Delta_1, \Delta_2$	$''$

$\hat{\Theta}, {}^{\mathsf{d}}\widehat{\Xi', {}^{\mathsf{d}}\Theta_0}; \Gamma \vdash v_1 \Leftarrow [\widehat{{}^{\mathsf{d}}\Xi'}/{}^{\mathsf{d}}\Xi']R_Q \wedge [\widehat{{}^{\mathsf{d}}\Xi'}/{}^{\mathsf{d}}\Xi']\overrightarrow{\varphi} \;/\; [\widehat{{}^{\mathsf{d}}\Xi'}/{}^{\mathsf{d}}\Xi']\overrightarrow{\varphi}, \chi_1 \dashv \Delta_1$ Alg$\Leftarrow\wedge$

$\hat{\Theta}, {}^{\mathsf{d}}\widehat{\Xi', {}^{\mathsf{d}}\Theta_0}; \Gamma \vdash v_1 \Leftarrow [\widehat{{}^{\mathsf{d}}\Xi'}/{}^{\mathsf{d}}\Xi']Q \;/\; [\widehat{{}^{\mathsf{d}}\Xi'}/{}^{\mathsf{d}}\Xi']\overrightarrow{\varphi}, \chi_1 \dashv \Delta_1$ By def. $[-]-$ and equality

$\Delta_1 \vdash \xi_{\widehat{[{}^{\mathsf{d}}\Xi'/{}^{\mathsf{d}}\Xi']Q}} - \|\hat{\Theta}\| \angle \xi_{\widehat{[{}^{\mathsf{d}}\Xi'/{}^{\mathsf{d}}\Xi']\overrightarrow{\varphi}, \chi_1}} - \|\hat{\Theta}\|$ By i.h. (3), definitions $\widehat{[{}^{\mathsf{d}}\Xi'/{}^{\mathsf{d}}\Xi']\overrightarrow{\varphi}} \subseteq \Psi$

$\Delta_1 \vdash \xi_{\widehat{[{}^{\mathsf{d}}\Xi'/{}^{\mathsf{d}}\Xi']Q}} - \|\hat{\Theta}\| \angle \xi_{\Psi, \chi_1, \chi_2} - \|\hat{\Theta}\|$ $\mathscr{O} \vdash \mathscr{O}' \angle -$ monotone

$\Delta_1 \subseteq cl(\xi_\chi - \|\hat{\Theta}\|)(FV(\xi_\chi - \|\hat{\Theta}\|) - \Delta_1)$	By inversion
$\widehat{{}^{\mathsf{d}}\Xi'} \subseteq cl(\xi_{\widehat{[{}^{\mathsf{d}}\Xi'/{}^{\mathsf{d}}\Xi']Q}} - \|\hat{\Theta}\|)(\emptyset)$	By type WF, Lemma B.4
$\subseteq cl(\xi_\chi - \|\hat{\Theta}\|)(\emptyset)$	By Lemma G.32
$\subseteq cl(\xi_\chi - \|\hat{\Theta}\|)(FV(\xi_\chi - \|\hat{\Theta}\|) - \Delta)$	$cl(\mathscr{O})(-)$ monotone

Let $\chi_2' = [\widehat{\overline{{}^{\mathsf{d}}\Theta_0'}}/\overline{{}^{\mathsf{d}}\Theta_0'}][\widehat{{}^{\mathsf{d}}\Xi'}/{}^{\mathsf{d}}\Xi']{}^{\mathsf{d}}\Theta_0', \chi_2$.

$$\hat{\Theta}, {}^{\mathsf{d}}\Xi', \overline{{}^{\mathsf{d}}\Theta_0}; \Gamma \vdash v_2 \Leftarrow [{}^{\mathsf{d}}\Xi', \overline{{}^{\mathsf{d}}\Theta_0}/{}^{\mathsf{d}}\Xi', {}^{\mathsf{d}}\Theta_0]R' \;/\; \chi_2 \dashv \Delta_2$$

$$\hat{\Theta}, {}^{\mathsf{d}}\Xi', \overline{{}^{\mathsf{d}}\Theta_0'}; \Gamma \vdash v_2 \Leftarrow [{}^{\mathsf{d}}\Xi', \overline{{}^{\mathsf{d}}\Theta_0'}/{}^{\mathsf{d}}\Xi', {}^{\mathsf{d}}\Theta_0']R' \;/\; \chi_2 \dashv \Delta_2$$

$$\hat{\Theta}, \widehat{{}^{\mathsf{d}}\Xi'}, \overline{{}^{\mathsf{d}}\Theta_0'}; \Gamma \vdash v_2 \Leftarrow [\widehat{{}^{\mathsf{d}}\Xi'}, \overline{{}^{\mathsf{d}}\Theta_0'}/{}^{\mathsf{d}}\Xi', {}^{\mathsf{d}}\Theta_0']R' \;/\; \chi_2 \dashv \Delta_2$$

$$\hat{\Theta}, \widehat{{}^{\mathsf{d}}\Xi'}, \overline{{}^{\mathsf{d}}\Theta_0'}; \Gamma \vdash v_2 \Leftarrow [\widehat{\overline{{}^{\mathsf{d}}\Theta_0'}}/\overline{{}^{\mathsf{d}}\Theta_0'}][\widehat{{}^{\mathsf{d}}\Xi'}/{}^{\mathsf{d}}\Xi'](R' \wedge {}^{\mathsf{d}}\Theta_0') \;/\; \chi_2' \dashv \Delta_2$$

$$\hat{\Theta}, \widehat{{}^{\mathsf{d}}\Xi'}; \Gamma \vdash v_2 \Leftarrow \exists \overline{{}^{\mathsf{d}}\Theta_0'}.\, ([\widehat{{}^{\mathsf{d}}\Xi'}/{}^{\mathsf{d}}\Xi]R' \wedge [\widehat{{}^{\mathsf{d}}\Xi'}/{}^{\mathsf{d}}\Xi]{}^{\mathsf{d}}\Theta_0') \;/\; \chi_2' \dashv \widehat{\overline{{}^{\mathsf{d}}\Theta_0'}}, \Delta_2$$

By exchange, renaming on $\mathscr{D}_0 :: {}^{\mathsf{d}}\Xi', \dot{\div}\hat{\Theta} \vdash \wr \overrightarrow{\beta'}; \hat{P}; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Theta_0'; R'$,

$\widehat{{}^{\mathsf{d}}\Xi'}, \dot{\div}\hat{\Theta} \vdash \wr [\widehat{{}^{\mathsf{d}}\Xi'}/{}^{\mathsf{d}}\Xi']\overrightarrow{\beta'}; \hat{P}; \mathscr{M}(F) \wr \doteq [\widehat{{}^{\mathsf{d}}\Xi'}/{}^{\mathsf{d}}\Xi']{}^{\mathsf{d}}\Theta_0'; [\widehat{{}^{\mathsf{d}}\Xi'}/{}^{\mathsf{d}}\Xi']R'$

by a derivation of height equal to that of $\mathscr{D}_0$.

	$\nexists x.\, \langle v_1, v_2 \rangle = \overrightarrow{\mathrm{inj}_{k_i}}^i \left(\overrightarrow{\langle __j, - \rangle}^j x \right)$	Given
	$\nexists x.\, v_2 = \overrightarrow{\mathrm{inj}_{k_i}}^i \left(\overrightarrow{\langle __j, - \rangle}^j x \right)$	Straightforward
$\widehat{\overline{{}^{\mathsf{d}}\Theta_0'}}, \Delta_2 \vdash \xi' - \|\hat{\Theta}, \widehat{{}^{\mathsf{d}}\Xi'}\| \angle \xi_{\chi_2'} - \|\hat{\Theta}, \widehat{{}^{\mathsf{d}}\Xi'}\|$		By i.h.
$\widehat{\overline{{}^{\mathsf{d}}\Theta_0'}}, \Delta_2 \vdash \xi' - \|\hat{\Theta}\| \angle \xi_{\chi_2'} - \|\hat{\Theta}\|$		By definitions
$\widehat{\overline{{}^{\mathsf{d}}\Theta_0'}}, \Delta_2 \vdash \xi' - \|\hat{\Theta}\| \angle \xi_{\chi} - \|\hat{\Theta}\|$		$\mathscr{O}' \vdash \mathscr{O} \angle -$ monotone
$\Delta_1, \widehat{{}^{\mathsf{d}}\Xi'} \subseteq cl\,(\xi_\chi - \|\hat{\Theta}\|)(FV(\xi_\chi - \|\hat{\Theta}\|) - \Delta)$		Above
	$\widehat{{}^{\mathsf{d}}\Xi'}, \widehat{\overline{{}^{\mathsf{d}}\Theta_0'}}, \Delta_1, \Delta_2 \vdash \xi' - \|\hat{\Theta}\| \angle \xi_{\chi} - \|\hat{\Theta}\|$	Inversion, set theory, rule
	${}^{\mathsf{d}}\Xi', \widehat{{}^{\mathsf{d}}\Theta_0'}, \Delta_1, \Delta_2 \vdash \xi' - \|\hat{\Theta}\| \angle \xi_{\chi} - \|\hat{\Theta}\|$	$\widehat{-}$, append commute
	${}^{\mathsf{d}}\Xi', \widehat{{}^{\mathsf{d}}\Theta_0}, \Delta_1, \Delta_2 \vdash \xi' - \|\hat{\Theta}\| \angle \xi_{\chi} - \|\hat{\Theta}\|$	def., equality
☞	$\Delta \vdash \xi' - \|\hat{\Theta}\| \angle \xi_{\chi} - \|\hat{\Theta}\|$	By equalities

- **Case** $\mathrm{Alg}\wr \mathrm{Id} \wr$

$$\overrightarrow{a \stackrel{\mathsf{d}\cdot}{\div} \tau} = \overrightarrow{a} \stackrel{\mathsf{d}\cdot}{\div} \mathscr{M}(F)$$

$$\overrightarrow{a \stackrel{\mathsf{d}\cdot}{\div} \tau, a\,\mathrm{Id}}, \stackrel{\mathsf{d}\cdot}{\div}\hat{\Theta} \rhd \wr \overrightarrow{q \Rightarrow t'}; \hat{I}; \mathscr{M}(F) \wr \stackrel{\circ}{=} {}^{\mathsf{d}}\Xi', \overrightarrow{\psi}; R'$$

$$\stackrel{\mathsf{d}\cdot}{\div}\hat{\Theta}; {}^{\mathsf{d}}\Xi'; zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash \overrightarrow{\psi} \rightsquigarrow \check{\Xi}_1; \mathscr{M}_1(F); \overrightarrow{\psi'}$$

$$\stackrel{\mathsf{d}\cdot}{\div}\hat{\Theta}; {}^{\mathsf{d}}\Xi'; zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash R'' \rightsquigarrow \check{\Xi}_2; \mathscr{M}_2(F); R''$$

$$\check{\Xi}'' = \check{\Xi}_1 \cup \check{\Xi}_2 \qquad \mathscr{M}'(F) = \mathscr{M}_1(F) \cup \mathscr{M}_2(F)$$

$$\frac{dom({}^{\mathsf{d}}\Xi'') \cap dom(\stackrel{\mathsf{d}\cdot}{\div}\hat{\Theta}, \overrightarrow{a \stackrel{\mathsf{d}\cdot}{\div} \tau}, {}^{\mathsf{d}}\Xi', \check{\Xi}'') = \emptyset \qquad \rho \text{ is the variable renaming } {}^{\mathsf{d}}\Xi''/\check{\Xi}''}{\stackrel{\mathsf{d}\cdot}{\div}\hat{\Theta} \rhd \wr \overrightarrow{(a,q) \Rightarrow t'}; \mathrm{Id} \otimes \hat{I}; \mathscr{M}(F) \wr \stackrel{\circ}{=} \underbrace{{}^{\mathsf{d}}\Xi'', {}^{\mathsf{d}}\Xi', [\rho]\overrightarrow{\psi'}}_{{}^{\mathsf{d}}\Theta}; \underbrace{\{\nu : \mu F \mid [\rho]\mathscr{M}'(F)\} \times [\rho]R''}_{R}}$$

Assume $({}^{\mathsf{d}}\Xi'', {}^{\mathsf{d}}\Xi') \neq \cdot$. The empty case is similar.

$$\overrightarrow{a \stackrel{\mathsf{d}\cdot}{\div} \tau, a\,\mathrm{Id}}, \hat{\Xi} \rhd \wr \overrightarrow{q \Rightarrow t'}; \hat{I}; \mathscr{M}(F) \wr \stackrel{\circ}{=} {}^{\mathsf{d}}\Xi', \overrightarrow{\psi}; R' \quad \text{Subderivation}$$

$$\stackrel{\mathsf{d}\cdot}{\div}\hat{\Theta}; {}^{\mathsf{d}}\Xi'; zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash \overrightarrow{\psi} \rightsquigarrow \check{\Xi}_1; \mathscr{M}_1(F); \overrightarrow{\psi'} \quad \text{Premise}$$

$$\stackrel{\mathsf{d}\cdot}{\div}\hat{\Theta}; {}^{\mathsf{d}}\Xi'; zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash R' \rightsquigarrow \check{\Xi}_2; \mathscr{M}_2(F); R'' \quad \text{Premise}$$

$$cl((\xi_{[\rho]\mathscr{M}'} - \stackrel{\mathsf{d}\cdot}{\div}\hat{\Theta}) - {}^{\mathsf{d}}\Xi')(\emptyset) = dom({}^{\mathsf{d}}\Xi'') \quad \text{Lemma C.54 (liftapps WF)}$$

(algo. version)

$$\overrightarrow{\psi'} = \overrightarrow{t = \langle t' \mid \boldsymbol{t} \rangle} \quad \text{By inversion}$$

$$\mathscr{M}(F) = \overrightarrow{(\mathsf{fold}_F\ \alpha)\, \nu\, \boldsymbol{t} =_\tau t} \quad ''$$

$\hat{\Theta};\Gamma \vdash v \Leftarrow \exists^{\mathsf{d}}\Theta.R \wedge {}^{\mathsf{d}}\Theta \;/\; \chi \dashv \Delta$ Given

$\hat{\Theta},\hat{\Xi};\Gamma \vdash v \Leftarrow [\hat{\Xi}/\overline{{}^{\mathsf{d}}\Theta}]R \wedge \Psi \;/\; \chi \dashv \Delta'$ By inversion on Alg$\Leftarrow\exists$, definitions

$\Delta = \hat{\Xi},\Delta'$ $''$

$\hat{\Xi} = \widehat{{}^{\mathsf{d}}\Xi''},\widehat{{}^{\mathsf{d}}\Xi'}$ By equality, $\hat{}$ property

$\Psi = [\hat{\Xi}/\Xi][\rho]\overrightarrow{\psi'}$ By equalities

By inversion on Alg$\Leftarrow\wedge$, definitions, substitution properties, $FV(F) = \emptyset$,

$$\hat{\Theta},\widehat{{}^{\mathsf{d}}\Xi''},\widehat{{}^{\mathsf{d}}\Xi'};\Gamma \vdash v \Leftarrow \left\{\nu : \mu F \;\middle|\; [\hat{\Xi}/\Xi][\rho].\mathscr{M}'\right\} \times [\hat{\Xi}/\Xi][\rho]R'' \;/\; \chi_0 \dashv \Delta'$$

$$\chi = \Psi,\chi_0$$

$$\begin{aligned}\Psi &= [\hat{\Xi}/\Xi][\rho]\overrightarrow{\psi'}\\ &= [\hat{\Xi}/\Xi][\rho]\left(\overrightarrow{t = \langle t' \mid \boldsymbol{t}\rangle}\right)\\ &= \overrightarrow{t = [\hat{\Xi}/\Xi][\rho]\langle t' \mid \boldsymbol{t}\rangle} \qquad \text{By def. subst.}\\ &\qquad\qquad\qquad\qquad\qquad\qquad \text{and } FV(t) \cap dom(\Xi) = \emptyset\end{aligned}$$

Let $R_0 = \left\{\nu : \mu F \;\middle|\; [\hat{\Xi}/\Xi][\rho].\mathscr{M}'\right\} \times [\hat{\Xi}/\Xi][\rho]R''$.

Let $\mathfrak{D}' = FV([\hat{\Xi}/\Xi][\rho]\overrightarrow{\langle t' \mid \boldsymbol{t}\rangle}) - \|\hat{\Theta}\|$.

Let $\mathfrak{D} = FV(\overrightarrow{\boldsymbol{t}}) - \|\hat{\Theta}\|$.

$$
\begin{aligned}
\mathfrak{D}' &= FV([\hat{\Xi}/\Xi]\overrightarrow{\langle t' \mid t \rangle}) - \|\hat{\Theta}\| \\
&\subseteq (FV([\hat{\Xi}/\Xi]\overrightarrow{t'}) \cup FV([\hat{\Xi}/\Xi]\overrightarrow{t})) - \|\hat{\Theta}\| \\
&\subseteq (FV([\hat{\Xi}/\Xi]\overrightarrow{t'}) \cup FV(\overrightarrow{t})) - \|\hat{\Theta}\| && \text{As } FV(\overrightarrow{t}) \cap dom({}^{\mathsf{d}}\Xi'', {}^{\mathsf{d}}\Xi') = \emptyset \\
&= (FV([\hat{\Xi}/\Xi]\overrightarrow{t'}) - \|\hat{\Theta}\|) \cup \mathfrak{D} \\
&\subseteq dom(\hat{\Xi}) \cup \mathfrak{D} \\
&\subseteq \mathfrak{D} \cup (\hat{\Xi}, \Delta') \\
&= \mathfrak{D} \cup \Delta
\end{aligned}
$$

To apply the rule for the goal, we need to show (SG1) and (SG2), respectively:

$$\Delta \subseteq cl(\xi_\chi - \|\hat{\Theta}\|)(FV(\xi_\chi - \|\hat{\Theta}\|) - \Delta)$$

and

$$\forall \widetilde{\mathfrak{D}} \to \hat{c} \in \xi' - \|\hat{\Theta}\|, \exists \widetilde{\mathfrak{B}} \to \hat{c} \in \xi_\chi - \|\hat{\Theta}\| \text{ s.t. } \widetilde{\mathfrak{B}} \subseteq \widetilde{\mathfrak{D}} \cup \Delta \text{ and } \widetilde{\mathfrak{B}} \cap \Delta \subseteq cl(\xi_\chi - \|\hat{\Theta}\|)(\widetilde{\mathfrak{D}})$$

$\nexists x.\, v = \overrightarrow{\mathrm{inj}_{k_i}}^{i}\left(\overrightarrow{\langle __j, - \rangle}^{j} x\right)$	Given
$\hat{\Theta}, \widehat{{}^{\mathsf{d}}\Xi''}, \widehat{{}^{\mathsf{d}}\Xi'}; \Gamma \vdash v_1 \Leftarrow \{\nu : \mu F \mid [\hat{\Xi}/\Xi][\rho]\mathscr{M}'\} \,/\, \chi_1 \dashv \Delta_1'$	By inversion on Alg$\Leftarrow\times$
$\hat{\Theta}, \widehat{{}^{\mathsf{d}}\Xi''}, \widehat{{}^{\mathsf{d}}\Xi'}; \Gamma \vdash v_2 \Leftarrow [\hat{\Xi}/\Xi][\rho]R'' \,/\, \chi_2 \dashv \Delta_2'$	$''$
$v = \langle v_1, v_2 \rangle$	$''$
$\chi_0 = \chi_1, \chi_2$	$''$
$\Delta' = \Delta_1', \Delta_2'$	$''$

Extend the liftapps judgment in the obvious way to perform the substitution on lambda abstraction indices. Then we can instead perform the substitution on the principal input measures of unrolling subderivation

$$\mathscr{D}_0 :: \overrightarrow{a \mathbin{\dot{\div}} \tau, a\,\mathrm{Id}}, \dot{\div}\hat{\Theta} \rhd \wr \overrightarrow{q \Rightarrow t'}; \hat{I}; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Xi', \overrightarrow{\psi'}; R'$$

to get the result on the outputs by an unrolling derivation of the same height. Different variables can be generated (the input has not been hereditarily applied like in the output so Id), but we can rename them (ρ') to the ones corresponding to the unrolling output:

$$\dot{\div}\hat{\Theta}; \cdot; zip(\overrightarrow{a})(\mathscr{M}(F)) \vdash \overrightarrow{t'} \rightsquigarrow \breve{\Xi}'''; _; \overrightarrow{t''}$$

$$\rho' = \breve{\Xi}''/\breve{\Xi}'''$$

$$\mathscr{D}_0' :: {}^{\mathsf{d}}\Xi'', \dot{\div}\hat{\Theta} \rhd \wr \overrightarrow{q \Rightarrow [\rho][\rho']t''}; \hat{I}; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Xi', [\rho]\overrightarrow{\psi'}; [\rho]R''$$

$$hgt(\mathscr{D}_0') = hgt(\mathscr{D}_0)$$

By renaming, exchange, and $FV(\mathscr{M}) \cap dom(\widehat{{}^{\mathsf{d}}\Xi''}) = \emptyset$,

$$\mathscr{D}_0'' :: \dot{\div}\hat{\Theta}, \widehat{{}^{\mathsf{d}}\Xi''} \rhd \wr \overrightarrow{q \Rightarrow t'''}; \hat{I}; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Xi', [\widehat{{}^{\mathsf{d}}\Xi''}/{}^{\mathsf{d}}\Xi'']\overrightarrow{\psi'}; [\widehat{{}^{\mathsf{d}}\Xi''}/{}^{\mathsf{d}}\Xi''][\rho]R''$$

$$\overrightarrow{t'''} = \overrightarrow{[\widehat{{}^{\mathsf{d}}\Xi''}/{}^{\mathsf{d}}\Xi''][\rho][\rho']t''}$$

$$hgt(\mathscr{D}_0'') = hgt(\mathscr{D}_0')$$

By Alg$\Leftarrow\wedge$ and def. of substitution,

$$\hat{\Theta}, \widehat{{}^{\mathsf{d}}\Xi''}, \widehat{{}^{\mathsf{d}}\Xi'}; \Gamma \vdash v_2 \Leftarrow [\widehat{{}^{\mathsf{d}}\Xi'}/{}^{\mathsf{d}}\Xi']Q' \;/\; \Psi, \chi_2 \dashv \Delta_2'$$

$$Q' = [\widehat{{}^{\mathsf{d}}\Xi''}/{}^{\mathsf{d}}\Xi'']R'' \wedge [\widehat{{}^{\mathsf{d}}\Xi''}/{}^{\mathsf{d}}\Xi'']\overrightarrow{\psi'}$$

Assume ${}^{\mathsf{d}}\Xi'' \neq \cdot$. The empty case is similar. By Alg$\Leftarrow\exists$,

$$\hat{\Theta}, \widehat{\mathsf{d}\,\Xi''}; \Gamma \vdash v_2 \Leftarrow \exists^{\mathsf{d}}\Xi'.\, Q' \;/\; \Psi, \chi_2 \dashv \widehat{\mathsf{d}\,\Xi'}, \Delta_2'$$

$$\nexists x.\, v_2 = \overrightarrow{\mathrm{inj}_{k_i}}^{i}\left(\overrightarrow{\langle __j, - \rangle}^{j}\, x\right) \qquad \text{Straightforward}$$

$$\widehat{\mathsf{d}\,\Xi'}, \Delta_2' \vdash \xi' - \|\hat{\Theta}\| \angle \xi_{\Psi,\chi_2} - \|\hat{\Theta}\| \qquad \text{By i.h.}$$

By inversion,

$$\widehat{\mathsf{d}\,\Xi'}, \Delta_2' \subseteq cl(\xi_{\Psi,\chi_2} - \|\hat{\Theta}\|)(FV(\xi_{\Psi,\chi_2} - \|\hat{\Theta}\|) - (\widehat{\mathsf{d}\,\Xi'}, \Delta_2'))$$

$$\forall \widetilde{\mathfrak{D}} \to \hat{c} \in \xi' - \|\hat{\Theta}\|, \exists \widetilde{\mathfrak{B}} \to \hat{c} \in \xi_{\Psi,\chi_2} - \|\hat{\Theta}\| \text{ s.t. } \widetilde{\mathfrak{B}} \subseteq \widetilde{\mathfrak{D}} \cup (\widehat{\mathsf{d}\,\Xi'}, \Delta_2') \text{ and } \widetilde{\mathfrak{B}} \cap (\widehat{\mathsf{d}\,\Xi'}, \Delta_2') \subseteq cl(\xi_{\Psi,\chi_2} - \|\hat{\Theta}\|)(\widetilde{\mathfrak{D}}) \text{ (*)}$$

– **Case** $v_1 = x$

$$\hat{\Theta}, \widehat{\mathsf{d}\,\Xi''}, \widehat{\mathsf{d}\,\Xi'}; \Gamma \vdash x \Leftarrow \left\{ \nu : \mu F \;\middle|\; [\hat{\Xi}/\Xi][\rho].\mathscr{M}' \right\} \;/\; \chi_1 \dashv \Delta_1' \qquad \text{By equality}$$

$$\Delta_1' = \cdot \qquad \text{By inversion}$$

$$\blacktriangleright(\widehat{\mathsf{d}\,\Xi''}) = \blacktriangleright(pos(\xi_{[\hat{\Xi}/\Xi][\rho].\mathscr{M}'} - \|\hat{\Theta}\|)) \qquad ''$$

$$\subseteq \xi_{\chi_1} - \|\hat{\Theta}\| \qquad \text{By part (1)}$$

$$\subseteq \xi_{\Psi,\chi_1,\chi_2} - \|\hat{\Theta}\| \qquad \text{By set theory}$$

$$= \xi_{\chi} - \|\hat{\Theta}\| \qquad \text{By equalities}$$

$$\widehat{\mathsf{d}\,\Xi''} \subseteq cl(\xi_{\chi} - \|\hat{\Theta}\|)(\emptyset) \qquad \blacktriangleright(\widehat{\mathsf{d}\,\Xi''}) \subseteq \xi_{\chi} - \|\hat{\Theta}\|$$

$$\subseteq cl(\xi_{\chi} - \|\hat{\Theta}\|)(FV(\xi_{\chi} - \|\hat{\Theta}\|) - \Delta) \qquad cl(\mathscr{O})(-) \text{ monotone}$$

$$\widehat{\mathsf{d}\,\Xi'}, \Delta_2' \vdash \xi' - \|\hat{\Theta}\| \angle \xi_{\Psi,\chi_2} - \|\hat{\Theta}\| \qquad \text{Above}$$

$$\widehat{\mathsf{d}\,\Xi'}, \Delta_2' \vdash \xi' - \|\hat{\Theta}\| \angle \xi_{\Psi,\chi_1,\chi_2} - \|\hat{\Theta}\| \qquad \mathscr{O} \vdash \mathscr{O}' \angle - \text{ monotone}$$

$$\widehat{\mathsf{d}\,\Xi'}, \Delta_2' \vdash \xi' - \|\hat{\Theta}\| \angle \xi_{\Psi,\chi_0} - \|\hat{\Theta}\| \qquad \text{By equality}$$

$$\begin{aligned}
\Delta &= \widehat{{}^{\mathsf{d}}\Xi''}, \widehat{{}^{\mathsf{d}}\Xi'}, \Delta_1', \Delta_2' && \text{By equalities} \\
&= \widehat{{}^{\mathsf{d}}\Xi''}, \widehat{{}^{\mathsf{d}}\Xi'}, \Delta_2' && \text{By equality} \\
&\subseteq cl(\xi_\chi - \|\hat{\Theta}\|)(FV(\xi_\chi - \|\hat{\Theta}\|) - \Delta) && \text{By property of union}
\end{aligned}$$

That establishes (SG1).

Assume $\mathfrak{D}{\rightarrow}t \in \xi' - \|\hat{\Theta}\|$. Goal (SG2) reduces to

$$\exists \widetilde{\mathfrak{B}}{\rightarrow}t \in \xi_\chi - \|\hat{\Theta}\|.\, \widetilde{\mathfrak{B}} \subseteq \mathfrak{D} \cup \Delta \text{ and } \widetilde{\mathfrak{B}} \cap \Delta \subseteq cl(\xi_\chi - \|\hat{\Theta}\|)(\mathfrak{D}).$$

$$\begin{aligned}
\mathfrak{D}''{\rightarrow}t &\in \xi_{\Psi,\chi_2} - \|\hat{\Theta}\| && \text{By (*)} \\
\mathfrak{D}'' &\subseteq \mathfrak{D} \cup (\widehat{{}^{\mathsf{d}}\Xi'}, \Delta_2') && '' \\
\mathfrak{D}'' \cap (\widehat{{}^{\mathsf{d}}\Xi'}, \Delta_2') &\subseteq cl(\xi_{\Psi,\chi_2} - \|\hat{\Theta}\|)(\mathfrak{D}) && ''
\end{aligned}$$

$$\begin{aligned}
\mathfrak{D}''{\rightarrow}t &\in \xi_{\Psi,\chi_2} - \|\hat{\Theta}\| && \text{Above} \\
&\subseteq \xi_{\Psi,\chi_1,\chi_2} - \|\hat{\Theta}\| \\
&= \xi_\chi - \|\hat{\Theta}\| && \text{By equalities}
\end{aligned}$$

$$\begin{aligned}
\mathfrak{D}'' &\subseteq \mathfrak{D} \cup (\widehat{{}^{\mathsf{d}}\Xi'}, \Delta_2') && \text{Above} \\
&\subseteq \mathfrak{D} \cup \Delta
\end{aligned}$$

$$\begin{aligned}
\mathfrak{D}'' \cap (\widehat{\mathsf{d}\Xi'}, \Delta_2') &\subseteq cl(\xi_{\Psi,\chi_2} - \|\hat{\Theta}\|)(\mathfrak{D}) && \text{Above}\\
&\subseteq cl(\xi_{\Psi,\chi_1,\chi_2} - \|\hat{\Theta}\|)(\mathfrak{D}) && cl(-)(\mathscr{O}) \text{ monotone}\\
&= cl(\xi_{\chi} - \|\hat{\Theta}\|)(\mathfrak{D}) && \text{By equalities}\\
\widehat{\mathsf{d}\Xi''} &\subseteq cl(\xi_{\chi} - \|\hat{\Theta}\|)(\emptyset) && \text{Above}\\
&\subseteq cl(\xi_{\chi} - \|\hat{\Theta}\|)(\mathfrak{D}) && cl(\mathscr{O})(-) \text{ monotone}\\
\mathfrak{D}'' \cap \Delta &\subseteq cl(\xi_{\chi} - \|\hat{\Theta}\|)(\mathfrak{D}) && \text{By properties of union}\\
& && \text{and intersection}
\end{aligned}$$

☞ $\Delta \vdash \xi' - \|\hat{\Theta}\| \angle \xi_{\chi} - \|\hat{\Theta}\|$ By rule

– **Case** $v_1 = \mathtt{into}(v_1')$

Let $X' = FV(\xi_{\Psi,\chi_2} - \|\hat{\Theta}\|) - (\widehat{\mathsf{d}\Xi'}, \Delta_2')$.

Let $X = FV(\xi_{\Psi,\chi_2} - \|\hat{\Theta}\|) - (\widehat{\mathsf{d}\Xi'}, \Delta_1', \Delta_2')$.

Let $Y = FV(\xi_{\underbrace{\Psi,\chi_1,\chi_2}_{\chi}} - \|\hat{\Theta}\|) - \underbrace{(\widehat{\mathsf{d}\Xi''}, \widehat{\mathsf{d}\Xi'}, \Delta_1', \Delta_2')}_{\Delta}$.

$$\begin{aligned}
X - \widehat{\mathsf{d}\Xi''} &\subseteq Y && \text{Straightforward}\\
X &\subseteq Y \cup \widehat{\mathsf{d}\Xi''} && \text{Set theory}\\
X \cup Y &\subseteq Y \cup \widehat{\mathsf{d}\Xi''} && \text{Set theory}
\end{aligned}$$

$$
\begin{array}{lll}
\widehat{\mathsf{d}\Xi'}, \Delta_2' \subseteq cl(\xi_{\Psi,\chi_2} - \|\hat{\Theta}\|)(X') & & \text{Above} \\
\quad = cl(\xi_{\Psi,\chi_2} - \|\hat{\Theta}\|)(X) & & FV(\xi_{\Psi,\chi_2} - \|\hat{\Theta}\|) \cap \Delta_1' = \emptyset \\
\quad \subseteq cl(\xi_{\chi} - \|\hat{\Theta}\|)(X) & & cl(-)(\mathscr{O}) \text{ monotone} \\
\quad \subseteq cl(\xi_{\chi} - \|\hat{\Theta}\|)(X \cup Y) & & cl(\mathscr{O})(-) \text{ monotone} \\
\quad \subseteq cl(\xi_{\chi} - \|\hat{\Theta}\|)(Y \cup \widehat{\mathsf{d}\Xi''}) & & cl(\mathscr{O})(-) \text{ monotone} \\
\quad = cl(\xi_{\chi} - \|\hat{\Theta}\|)(\underbrace{FV(\xi_{\chi} - \|\hat{\Theta}\|) - \Delta}_{Y}) & & \text{By Lemma B.5, equalities}
\end{array}
$$

$$
\hat{\Theta}, \widehat{\mathsf{d}\Xi''}, \widehat{\mathsf{d}\Xi'}; \Gamma \vdash \mathtt{into}(v_1') \Leftarrow \left\{ v : \mu F \;\middle|\; [\hat{\Xi}/\Xi][\rho]\mathscr{M}' \right\} \,/\, \chi_1 \dashv \Delta_1' \quad \text{By equality}
$$

$$
\begin{array}{lll}
[\hat{\Xi}/\Xi][\rho]\mathscr{M}' \rightsquigarrow \overrightarrow{\alpha} & & \text{By inversion on Alg}\Leftarrow\mu \\
\dot{\div}\hat{\Theta}, \widehat{\mathsf{d}\Xi''}, \widehat{\mathsf{d}\Xi'} \rhd \wr \overrightarrow{\alpha}; F; [\hat{\Xi}/\Xi][\rho]\mathscr{M}'(F) \wr \doteq {}^{\mathsf{d}}\Theta_2; R_2 & & '' \\
\hat{\Theta}, \widehat{\mathsf{d}\Xi''}, \widehat{\mathsf{d}\Xi'}; \Gamma \vdash v_1' \Leftarrow \exists {}^{\mathsf{d}}\Theta_2. R_2 \wedge {}^{\mathsf{d}}\Theta_2 \,/\, \chi_1 \dashv \Delta_1' & & '' \\
\quad \nexists x. v_1' = \overrightarrow{\mathrm{inj}_{k_i}}^{i}\left(\overrightarrow{\langle __j, - \rangle}^{j} x \right) & & ''
\end{array}
$$

$$
\Delta_1' \vdash \xi_{[\hat{\Xi}/\Xi][\rho].\mathscr{M}'} - \|\hat{\Theta}\| \angle \xi_{\chi_1} - \|\hat{\Theta}\| \quad \text{By i.h.}
$$

By inversion,

$$
\Delta_1' \subseteq cl(\xi_{\chi_1} - \|\hat{\Theta}\|)(FV(\xi_{\chi_1} - \|\hat{\Theta}\|) - \Delta_1')
$$

$$
\forall \widetilde{\mathfrak{D}} \to \hat{c} \in \xi_{[\hat{\Xi}/\Xi][\rho].\mathscr{M}'} - \|\hat{\Theta}\|, \exists \widetilde{\mathfrak{B}} \to \hat{c} \in \xi_{\chi_1} - \|\hat{\Theta}\| \text{ s.t. } \widetilde{\mathfrak{B}} \subseteq \widetilde{\mathfrak{D}} \cup \Delta_1' \text{ and } \widetilde{\mathfrak{B}} \cap \Delta_1' \subseteq cl(\xi_{\chi_1} - \|\hat{\Theta}\|)(\widetilde{\mathfrak{D}})
$$

$$
\begin{array}{lll}
\Delta_1' \subseteq cl(\xi_{\chi_1} - \|\hat{\Theta}\|)(FV(\xi_{\chi_1} - \|\hat{\Theta}\|) - \Delta_1') & & \text{Above} \\
\quad = cl(\xi_{\chi_1} - \|\hat{\Theta}\|)(FV(\xi_{\chi_1} - \|\hat{\Theta}\|) - (\widehat{\mathsf{d}\,\Xi'}, \Delta_1', \Delta_2')) & & FV(\xi_{\chi_1} - \|\hat{\Theta}\|) \text{ and} \\
& & (\widehat{\mathsf{d}\,\Xi'}, \Delta_2') \text{ disjoint} \\
\quad = cl(\xi_{\chi_1} - \|\hat{\Theta}\|)(FV(\xi_{\chi_1} - \|\hat{\Theta}\|) - \Delta) & & \text{By Lemma B.5} \\
\quad \subseteq cl(\xi_{\Psi,\chi_1,\chi_2} - \|\hat{\Theta}\|)(FV(\xi_{\chi_1} - \|\hat{\Theta}\|) - \Delta) & & cl(-)(\mathscr{O}) \text{ monotone} \\
\quad \subseteq cl(\xi_{\Psi,\chi_1,\chi_2} - \|\hat{\Theta}\|)(FV(\xi_{\Psi,\chi_1,\chi_2} - \|\hat{\Theta}\|) - \Delta) & & cl(\mathscr{O})(-) \text{ monotone} \\
\quad = cl(\xi_{\chi} - \|\hat{\Theta}\|)(FV(\xi_{\chi} - \|\hat{\Theta}\|) - \Delta) & & \text{By equalities}
\end{array}
$$

$$
\Delta_1' \vdash \xi_{[\hat{\Xi}/\Xi][\rho].\mathscr{M}'} - \|\hat{\Theta}\| - \widehat{\mathsf{d}\,\Xi'} \angle \xi_{\chi_1} - \|\hat{\Theta}\| - \widehat{\mathsf{d}\,\Xi'}
$$

$$
\begin{array}{lll}
cl((\xi_{[\rho].\mathscr{M}'} - \dot{\div}\hat{\Theta}) - \mathsf{d}\,\Xi')(\emptyset) = dom(\mathsf{d}\,\Xi'') & & \text{Above} \\
cl((\xi_{[\hat{\Xi}/\Xi][\rho].\mathscr{M}'} - \dot{\div}\hat{\Theta}) - \widehat{\mathsf{d}\,\Xi'})(\emptyset) = dom(\widehat{\mathsf{d}\,\Xi''}) & & \text{By renaming}
\end{array}
$$

$$
\begin{array}{lll}
\widehat{\mathsf{d}\,\Xi''} \subseteq cl((\xi_{[\hat{\Xi}/\Xi][\rho].\mathscr{M}'} - \dot{\div}\hat{\Theta}) - \widehat{\mathsf{d}\,\Xi'})(\emptyset) & & \\
\quad \subseteq cl(\xi_{[\hat{\Xi}/\Xi][\rho].\mathscr{M}'} - \|\hat{\Theta}\| - \widehat{\mathsf{d}\,\Xi'})(FV(\xi_{\chi_1} - \|\hat{\Theta}\|) - \Delta_1') & & \\
\quad = cl(\xi_{[\hat{\Xi}/\Xi][\rho].\mathscr{M}'} - \|\hat{\Theta}\| - \widehat{\mathsf{d}\,\Xi'})(FV(\xi_{\chi_1} - \|\hat{\Theta}\|) - \Delta) & & \\
\quad \subseteq cl(\xi_{\chi_1} - \|\hat{\Theta}\| - \widehat{\mathsf{d}\,\Xi'})(FV(\xi_{\chi_1} - \|\hat{\Theta}\|) - \Delta) & & \\
\quad \subseteq cl(\xi_{\chi} - \|\hat{\Theta}\| - \widehat{\mathsf{d}\,\Xi'})(FV(\xi_{\chi} - \|\hat{\Theta}\|) - \Delta) & & \\
\quad \subseteq cl(\xi_{\chi} - \|\hat{\Theta}\|)(FV(\xi_{\chi} - \|\hat{\Theta}\|) - \Delta) & & \text{By Lemma B.9}
\end{array}
$$

$$
\Delta \subseteq cl(\xi_{\chi} - \|\hat{\Theta}\|)(FV(\xi_{\chi} - \|\hat{\Theta}\|) - \Delta) \quad \text{By property of union}
$$

That establishes (SG1).

Assume $\mathfrak{D}{\rightarrow}t \in \xi' - \|\hat{\Theta}\|$; (SG2) reduces to

$$\exists \widetilde{\mathfrak{B}}{\rightarrow}t \in \xi_\chi - \|\hat{\Theta}\|.\, \widetilde{\mathfrak{B}} \subseteq \mathfrak{D} \cup \Delta \text{ and } \widetilde{\mathfrak{B}} \cap \Delta \subseteq cl(\xi_\chi - \|\hat{\Theta}\|)(\mathfrak{D}).$$

$$\begin{array}{rll}
\mathfrak{D}'''{\rightarrow}t \in \xi_{\Psi,\chi_2} - \|\hat{\Theta}\| & & \text{By (*)} \\
\mathfrak{D}''' \subseteq \mathfrak{D} \cup (\widehat{\mathsf{d}\Xi'}, \Delta_2') & & '' \\
\mathfrak{D}''' \cap (\widehat{\mathsf{d}\Xi'}, \Delta_2') \subseteq cl(\xi_{\Psi,\chi_2} - \|\hat{\Theta}\|)(\mathfrak{D}) & & '' \\
\subseteq cl(\xi_\chi - \|\hat{\Theta}\|)(\mathfrak{D}) & & cl(-)(\mathscr{O}) \text{ monotone}
\end{array}$$

$$\begin{array}{rll}
\mathfrak{D}'''{\rightarrow}t \in \xi_{\Psi,\chi_2} - \|\hat{\Theta}\| & & \text{Above} \\
\subseteq \xi_\chi - \|\hat{\Theta}\| & &
\end{array}$$

$$\begin{array}{rll}
\mathfrak{D}''' \subseteq \mathfrak{D} \cup (\widehat{\mathsf{d}\Xi'}, \Delta_2') & & \text{Above} \\
\subseteq \mathfrak{D} \cup \Delta & &
\end{array}$$

$$\begin{array}{rll}
\mathfrak{D}''' \cap \widehat{\mathsf{d}\Xi''} = \emptyset & & \mathfrak{D}''' \subseteq \mathfrak{D} \cup (\widehat{\mathsf{d}\hat{\Xi}'}, \Delta_2') \\
& & \text{and } (\mathfrak{D} \cup (\widehat{\mathsf{d}\Xi'}, \Delta_2')) \cap \widehat{\mathsf{d}\Xi''} = \emptyset \\
\mathfrak{D}''' \cap \Delta_1' = \emptyset & & \mathfrak{D}''' \subseteq \mathfrak{D} \cup (\widehat{\mathsf{d}\Xi'}, \Delta_2') \\
& & \text{and } (\mathfrak{D} \cup (\widehat{\mathsf{d}\Xi'}, \Delta_2')) \cap \Delta_1' = \emptyset \\
\mathfrak{D}''' \cap \Delta \subseteq cl(\xi_\chi - \|\hat{\Theta}\|)(\mathfrak{D}) & & \text{By properties of } \cap \text{ and } \cup \\
& & \text{and } \Delta = \widehat{\mathsf{d}\Xi''}, \widehat{\mathsf{d}\Xi'}, \Delta_1', \Delta_2'
\end{array}$$

☞ $\Delta \vdash \xi' - \|\hat{\Theta}\| \angle \xi_\chi - \|\hat{\Theta}\|$ By rule

- **Case**

$$\frac{\overrightarrow{t'} \mathbin{@} \mathscr{M}(F) \doteq \overrightarrow{\varphi}}{\hat{\Xi} \rhd \wr \overrightarrow{() \Rightarrow t'}; I; \mathscr{M}(F) \wr \doteq \overrightarrow{\varphi}; 1}\ \mathrm{Alg}\wr I \wr$$

By inversion $\overrightarrow{\varphi} = \overrightarrow{t = \langle t' \mid \boldsymbol{t} \rangle}$.

Given $\nexists x.\, v = \overrightarrow{\mathrm{inj}_{k_i}}^{i}\left(\overrightarrow{\langle __j, - \rangle}^{j} x\right)$, we know v cannot be a variable.

  - **Case**

$$\frac{\dfrac{}{\hat{\Theta}; \Gamma \vdash v \Leftarrow 1 \mathbin{/} \mathtt{tt} \dashv \cdot}\ \mathrm{Alg}{\Leftarrow}1}{\hat{\Theta}; \Gamma \vdash v \Leftarrow (1 \wedge \overrightarrow{\varphi}) \mathbin{/} \underbrace{\overrightarrow{\varphi}, \mathtt{tt}}_{\chi} \dashv \underbrace{\cdot}_{\Delta}}\ \mathrm{Alg}{\Leftarrow}\wedge$$

Let $\mathfrak{A} = FV(\overrightarrow{\boldsymbol{t}}) - \|\hat{\Theta}\|$.

Let $\mathfrak{A}' = (FV(\overrightarrow{\langle t' \mid \boldsymbol{t} \rangle}) - \|\hat{\Theta}\|)$.

$\|\dot{-}\hat{\Theta}\| \vdash \overrightarrow{() \Rightarrow t'} : I(\tau) \Rightarrow \tau$	Presupposed derivations
$\dot{-}\|\dot{-}\hat{\Theta}\| \vdash \overrightarrow{t'} : \tau$	By inversion
$\|\hat{\Theta}\| \vdash \overrightarrow{t'} : \tau$	By definitions and weakening

$FV(\overrightarrow{t'}) \subseteq dom(\|\hat{\Theta}\|)$	Straightforward

$\mathfrak{A}' = FV(\overrightarrow{\langle t' \mid \boldsymbol{t} \rangle}) - \|\hat{\Theta}\|$	By definition of $\mathfrak{A}'$
$\subseteq (FV(\overrightarrow{t'}) \cup FV(\overrightarrow{\boldsymbol{t}})) - \|\hat{\Theta}\|$	Straightforward
$= FV(\overrightarrow{\boldsymbol{t}}) - \|\hat{\Theta}\|$	As $FV(\overrightarrow{t'}) \subseteq dom(\|\hat{\Theta}\|)$
$= \mathfrak{A}$	By definition of $\mathfrak{A}$

The goal follows by the rule (its two premises are straightforward, the latter

using $\mathfrak{A}' \subseteq \mathfrak{A}$).

(3) By structural induction on v.

- **Case**

$$\frac{(x : R') \in \Gamma \qquad \hat{\Theta} \vdash R' <:^{+} R \mathbin{/} W}{\hat{\Theta}; \Gamma \vdash x \Leftarrow \underbrace{R}_{Q} \mathbin{/} \underbrace{W}_{\chi} \dashv \underbrace{\cdot}_{\Delta}} \text{Alg}\Leftarrow\text{Var}$$

	$\hat{\Theta} \vdash R' <:^{+} R \mathbin{/} W$	Premise
	$\overline{\hat{\Theta}} \rhd Q\ \mathsf{type}[\xi_Q]$	Given
	$R'\ \mathsf{ground}$	By inversion on presupposed ctx. WF
	$[\hat{\Theta}]Q = Q$	Given
☞	$\overline{\hat{\Theta}} \vdash W\ \mathsf{wf}[\xi]$	By part (1)
	$\cdot \vdash \xi_R - \|\hat{\Theta}\| \angle \xi - \|\hat{\Theta}\|$	$''$
☞	$\Delta \vdash \xi_Q - \|\hat{\Theta}\| \angle \xi - \|\hat{\Theta}\|$	By equalities

- **Case**

$$\frac{\hat{\Theta}; \Gamma \vdash v_1 \Leftarrow R_1 \mathbin{/} \chi_1 \dashv \Delta_1 \qquad \hat{\Theta}; \Gamma \vdash v_2 \Leftarrow R_2 \mathbin{/} \chi_2 \dashv \Delta_2}{\hat{\Theta}; \Gamma \vdash \langle v_1, v_2 \rangle \Leftarrow (R_1 \times R_2) \mathbin{/} \underbrace{\chi_1, \chi_2}_{\chi} \dashv \underbrace{\Delta_1, \Delta_2}_{\Delta}} \text{Alg}\Leftarrow\times$$

$\hat{\Theta};\Gamma \vdash v_1 \Leftarrow R_1 \,/\, \chi_1 \dashv \Delta_1$	Subderivation
$\overline{\hat{\Theta}}, \Delta_1 \vdash \lfloor \chi_1 \rfloor \ \mathsf{wf}[\xi_1]$	By i.h.
$\Delta_1 \vdash \xi_{R_1} - \lVert\hat{\Theta}\rVert \angle \xi_1 - \lVert\hat{\Theta}\rVert$	$''$
$\hat{\Theta};\Gamma \vdash v_2 \Leftarrow R_2 \,/\, \chi_2 \dashv \Delta_2$	Subderivation
$\overline{\hat{\Theta}}, \Delta_2 \vdash \lfloor \chi_2 \rfloor \ \mathsf{wf}[\xi_2]$	By i.h.
$\Delta_2 \vdash \xi_{R_2} - \lVert\hat{\Theta}\rVert \angle \xi_2 - \lVert\hat{\Theta}\rVert$	$''$
$\Delta_1, \Delta_2 \vdash (\xi_{R_1} - \lVert\hat{\Theta}\rVert) \cup (\xi_{R_2} - \lVert\hat{\Theta}\rVert) \angle (\xi_1 - \lVert\hat{\Theta}\rVert) \cup (\xi_2 - \lVert\hat{\Theta}\rVert)$	By Lemma G.34
☞ $\Delta_1, \Delta_2 \vdash (\xi_{R_1} \cup \xi_{R_2}) - \lVert\hat{\Theta}\rVert \angle (\xi_1 \cup \xi_2) - \lVert\hat{\Theta}\rVert$	Subtraction distributes over $\cup$

- **Case**

$$\frac{\begin{array}{c} \nexists x.\, v_0 = \overrightarrow{\mathrm{inj}_{k_i}}^{i}\left(\overrightarrow{\langle __j, - \rangle}^{j} x\right) \\ \mathscr{M}(F) \rightsquigarrow \overrightarrow{\alpha}; \overrightarrow{\tau} \\ \overset{\mathsf{d}}{\div}\hat{\Theta} \rhd \wr \overrightarrow{\alpha}; F; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Theta; R' \\ \hat{\Theta};\Gamma \vdash v_0 \Leftarrow \exists^{\mathsf{d}}\Theta.\, R' \wedge {}^{\mathsf{d}}\Theta \,/\, \chi \dashv \Delta \end{array}}{\hat{\Theta};\Gamma \vdash \mathtt{into}(v_0) \Leftarrow \underbrace{\{\nu : \mu F \mid \mathscr{M}(F)\}}_{Q} \,/\, \chi \dashv \Delta}\ \mathrm{Alg}{\Leftarrow}\mu$$

$\hat{\Theta};\Gamma \vdash v_0 \Leftarrow \exists^{\mathsf{d}}\Theta.\, R' \wedge {}^{\mathsf{d}}\Theta \,/\, \chi \dashv \Delta$	Subderivation
$\overset{\mathsf{d}}{\div}\hat{\Theta} \rhd \wr \overrightarrow{\alpha}; F; \mathscr{M}(F) \wr \doteq {}^{\mathsf{d}}\Theta; R'$	Premise
$\nexists x.\, v_0 = \overrightarrow{\mathrm{inj}_{k_i}}^{i}\left(\overrightarrow{\langle __j, - \rangle}^{j} x\right)$	Premise
☞ $\overline{\hat{\Theta}}, \Delta \vdash \lfloor \chi \rfloor \ \mathsf{wf}[\xi]$	By i.h., part (2)
☞ $\Delta \vdash \xi_Q - \lVert\hat{\Theta}\rVert \angle \xi - \lVert\hat{\Theta}\rVert$	$''$

- **Case** Alg$\Leftarrow\wedge$: Similar to Alg$\Leftarrow\times$ case.

- **Cases** Alg⇐$+_k$, Alg⇐1, Alg⇐↓:

  In these cases, $\xi_Q = \cdot$ and $\Delta = \cdot$.

  By a rule, $\cdot \vdash \cdot \angle \xi_\chi - \|\hat{\Theta}\|$.

  By equalities and def. of subtraction, $\Delta \vdash \xi_Q - \|\hat{\Theta}\| \angle \xi_\chi - \|\hat{\Theta}\|$.

- **Case** Alg⇐∃:

  Impossible because $Q \neq \exists$.

(4) Similar to part (3). □

Lemma G.36 (cl to Mediated cl)**.**

If $\Delta \vdash \xi' \angle \xi$ *and* $FV(\xi') \subseteq cl(\xi')(\emptyset)$ *and* $FV(\xi) \subseteq FV(\xi') \cup dom(\Delta)$

then $FV(\xi'), \Delta \subseteq cl(\xi)(\emptyset)$.

Proof.

$\Delta \vdash \xi' \angle \xi$ Given

$\emptyset = FV(\xi') \cap dom(\Delta)$ By inversion

$FV(\xi') \subseteq cl(\xi')(\emptyset)$ Given

$\subseteq cl(\xi)(\emptyset)$ By Lemma G.32 (Admissible Premise)

$FV(\xi) \subseteq FV(\xi') \cup dom(\Delta)$ Given

$FV(\xi) - \Delta \subseteq FV(\xi')$ Subtraction monotone and $FV(\xi') \cap dom(\Delta) = \emptyset$

$\Delta \subseteq cl(\xi)(FV(\xi) - \Delta)$ By inversion on $\Delta \vdash \xi' \angle \xi$

$\subseteq cl(\xi)(FV(\xi'))$ $cl(\mathscr{O})(-)$ monotone (with $FV(\xi) - \Delta \subseteq FV(\xi')$)

$= cl(\xi)(\emptyset)$ By Lemma B.5 (Weaken cl) with $FV(\xi') \subseteq cl(\xi)(\emptyset)$

☞ $FV(\xi'), \Delta \subseteq cl(\xi)(\emptyset)$ By property of union □

The following lemma is never directly used and we tend to use *implicitly* the lemma stating the equivalence of *cl* and det.

Lemma G.37 (Det. to Mediated Det.)**.**
If $\xi' \vdash FV(\xi')$ det *and* $\Delta \vdash \xi' \angle \xi$ *and* $FV(\xi) \subseteq FV(\xi') \cup dom(\Delta)$
then $\xi \vdash FV(\xi'), \Delta$ det.

Proof.

$\xi' \vdash FV(\xi')$ det	Given
$FV(\xi') \subseteq cl(\xi')(\emptyset)$	By Lemma B.4 (Equivalence of *cl* and det)
$FV(\xi) \subseteq FV(\xi') \cup dom(\Delta)$	Given
$\Delta \vdash \xi' \angle \xi$	Given
$FV(\xi'), \Delta \subseteq cl(\xi)(\emptyset)$	By Lemma G.36 (cl to Mediated cl)
☞ $\xi \vdash FV(\xi'), \Delta$ det	By Lemma B.4 (Equivalence of *cl* and det) □

Appendix G.3 Algorithmic Subtyping

Lemma G.38 (Aux. Alg. Sub. Complete)**.**

(1) If $\Theta \mathrel{\widetilde{\models}} W$ *then* $\Theta \models W$.

(2) If $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$ *and* $\|\hat{\Theta}\| \mathrel{\widetilde{\vdash}} R <:^{+} [\Omega]Q \;/\; W$ *and* $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W$
and $\hat{\Theta}$ present *and* $[\hat{\Theta}]Q = Q$ *and* R ground *and* $\overline{\hat{\Theta}} \triangleright Q$ type$[\xi]$
then there exists W' *such that* $\hat{\Theta} \vdash R <:^{+} Q \;/\; W'$
and $\|\hat{\Theta}\|; \cdot \vdash [\Omega]W'$ fixInstChk $\dashv \|\hat{\Theta}\|$ *and* $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W \leftrightarrow [\Omega]W'$.

(3) *If* $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$ *and* $\|\hat{\Theta}\| \mathrel{\widetilde{\vdash}} [\Omega]M <:^{-} L \mathbin{/} W$ *and* $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W$
and $\hat{\Theta}\,\mathsf{present}$ *and* $[\hat{\Theta}]M = M$ *and* $L\,\mathsf{ground}$ *and* $\overline{\hat{\Theta}} \rhd M\ \mathsf{type}[\xi]$
then there exists W' *such that* $\hat{\Theta} \vdash L <:^{-} M \mathbin{/} W'$
and $\|\hat{\Theta}\|;\cdot \vdash [\Omega]W'\ \mathsf{fixInstChk} \dashv \|\hat{\Theta}\|$ *and* $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W \leftrightarrow [\Omega]W'$.

(4) *If* $\Xi \mathrel{\widetilde{\vdash}} \alpha;F <:_{\tau} \alpha';F'$ *then* $\Xi \rhd \alpha;F <:_{\tau} \alpha';F'$.

(5) *If* $\Theta \mathrel{\widetilde{\vdash}} R <:^{+} P \mathbin{/} W$ *and* $\Theta \mathrel{\widetilde{\models}} W$ *then* $\Theta \vdash R <:^{+} P$.

(6) *If* $\Theta \mathrel{\widetilde{\vdash}} N <:^{-} L \mathbin{/} W$ *and* $\Theta \mathrel{\widetilde{\models}} W$ *then* $\Theta \vdash N <:^{-} L$.

(7) *If* $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$ *and* $\|\hat{\Theta}\| \mathrel{\widetilde{\vdash}} \mathscr{M}'(F') \geq [\Omega](\mathscr{M}(F)) \mathbin{/} W$ *and* $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W$
and $\hat{\Theta}\,\mathsf{present}$ *and* $[\hat{\Theta}](\mathscr{M}(F)) = \mathscr{M}(F)$ *and* $\mathscr{M}'(F')\,\mathsf{ground}$ *and* $\overline{\hat{\Theta}} \rhd \mathscr{M}(F)\ \mathsf{msmts}[\xi]$
then there exists W' *such that* $\hat{\Theta} \vdash \mathscr{M}'(F') \geq \mathscr{M}(F) \mathbin{/} W'$
and $\|\hat{\Theta}\|;\cdot \vdash [\Omega]W'\ \mathsf{fixInstChk} \dashv \|\hat{\Theta}\|$ *and* $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W \leftrightarrow [\Omega]W'$.

Proof. By induction on the sum of the height of the given semideclarative subtyping/submeasuring derivation and the height of the given semideclarative (subtyping) constraint checking derivation. All parts are mutually recursive.

(1) Straightforward

(2) • **Case**

$$\frac{}{\|\hat{\Theta}\| \mathrel{\widetilde{\vdash}} 0 <:^{+} \underbrace{0}_{[\Omega]0} \mathbin{/} \mathsf{tt}}\ \widetilde{<:}^{+}0$$

☞ $\hat{\Theta} \vdash 0 <:^{+} 0 \mathbin{/} \mathsf{tt}$ By $<:^{+}0$

☞ $\|\hat{\Theta}\| \mathrel{\widetilde{\models}} \mathsf{tt} \leftrightarrow \underbrace{\mathsf{tt}}_{[\Omega]\mathsf{tt}}$ By Lemma G.14 (Prob. Equiv. Reflexive)

☞ $\|\hat{\Theta}\|;\cdot \vdash \underbrace{\mathsf{tt}}_{[\Omega]\mathsf{tt}}\ \mathsf{fixInstChk} \dashv \|\hat{\Theta}\|$ Straightforward

- **Case** $\widetilde{<:^+}1$: Similar to $\widetilde{<:^+}0$ case.

- **Case**
$$\frac{\|\hat{\Theta}\| \mathrel{\widetilde{\vdash}} R_1 <:^+ [\Omega]R_1' \;/\; W_1 \qquad \|\hat{\Theta}\| \mathrel{\widetilde{\vdash}} R_2 <:^+ [\Omega]R_2' \;/\; W_2}{\|\hat{\Theta}\| \mathrel{\widetilde{\vdash}} R_1 \times R_2 <:^+ [\Omega]R_1' \times [\Omega]R_2' \;/\; W_1 \wedge W_2}\,\widetilde{<:^+}\times$$

$\|\hat{\Theta}\| \mathrel{\widetilde{\vdash}} R_1 <:^+ [\Omega]R_1' \;/\; W_1$	Subderivation
$\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$	Given
$\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W_1 \wedge W_2$	Given
$\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W_1$	By inversion on $\widetilde{\models}\leftrightarrow\wedge$
$\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W_2$	$''$
$\hat{\Theta} \vdash R_1 <:^+ R_1' \;/\; W_1'$	By i.h.
$\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W_1 \leftrightarrow [\Omega]W_1'$	$''$
$\|\hat{\Theta}\|;\cdot \vdash [\Omega]W_1' \;\mathsf{fixInstChk} \dashv \|\hat{\Theta}\|$	$''$
$\overline{\hat{\Theta}} \rhd R_1' \times R_2' \;\mathsf{type}[_]$	Given
$\overline{\hat{\Theta}} \rhd R_2' \;\mathsf{type}[_]$	By inversion
$\|\hat{\Theta}\| \mathrel{\widetilde{\vdash}} R_2 <:^+ [\Omega]R_2' \;/\; W_2$	Subderivation
$\hat{\Theta} \vdash R_2 <:^+ R_2' \;/\; W_2'$	By i.h.
$\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W_2 \leftrightarrow [\Omega]W_2'$	$''$
$\|\hat{\Theta}\|;\cdot \vdash [\Omega]W_2' \;\mathsf{fixInstChk} \dashv \|\hat{\Theta}\|$	$''$
☞ $\hat{\Theta} \vdash R_1 \times R_2 <:^+ R_1' \times R_2' \;/\; W_1' \wedge W_2'$	By $<:^+\times$

$$\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W_1 \wedge W_2 \leftrightarrow [\Omega]W_1' \wedge [\Omega]W_2' \quad \text{By } \widetilde{\models}\leftrightarrow\wedge$$

☞ $$\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W_1 \wedge W_2 \leftrightarrow [\Omega](W_1' \wedge W_2') \quad \text{By def. of } [-]-$$

☞ $$\|\hat{\Theta}\|;\cdot \vdash \underbrace{[\Omega]W_1' \wedge [\Omega]W_2'}_{[\Omega](W_1' \wedge W_2')} \mathsf{fixInstChk} \dashv \|\hat{\Theta}\| \quad \text{By Lemma G.2 (Inst. Compose)}$$

- **Case** $\widetilde{<:^+}+:$

$$\frac{P_1 = \exists^{\mathsf{d}} \Xi_1.\, R_1 \wedge \overrightarrow{\varphi_1} \qquad P_2 = \exists^{\mathsf{d}} \Xi_2.\, R_2 \wedge \overrightarrow{\varphi_2} \qquad W = (\forall^{\mathsf{d}} \Xi_1.\, \overrightarrow{\varphi_1} \supset \underline{R_1 <:^+ [\Omega]P_1'}) \wedge (\forall^{\mathsf{d}} \Xi_2.\, \overrightarrow{\varphi_2} \supset \underline{R_2 <:^+ [\Omega]P_2'})}{\Theta \mathrel{\widetilde{\vdash}} P_1 + P_2 <:^+ [\Omega]P_1' + [\Omega]P_2' \,/\, W}$$

By $<:^+ +$,

☞ $$\hat{\Theta} \vdash P_1 + P_2 <:^+ P_1' + P_2' \,/\, \forall^{\mathsf{d}} \Xi_1.\, \overrightarrow{\varphi_1} \supset \underline{R_1 <:^+ P_1'} \wedge \forall^{\mathsf{d}} \Xi_2.\, \overrightarrow{\varphi_2} \supset \underline{P_2 <:^+ P_2'}$$

The constraint equivalence follows from Lemma G.14 (Prob. Equiv. Reflexive), the definition of substitution, and R_1, R_2 ground.

The fixInstChk goal is easy (like in the $\widetilde{<:^+}1$ case, nothing is instantiated).

- **Case**

$$\frac{\|\hat{\Theta}\| \mathrel{\widetilde{\vdash}} R <:^+ [\Omega]R' \,/\, W_0}{\|\hat{\Theta}\| \mathrel{\widetilde{\vdash}} R <:^+ [\Omega]R' \wedge [\Omega]\overrightarrow{\varphi} \,/\, W_0 \wedge [\Omega]\overrightarrow{\varphi}} \widetilde{<:^+}\wedge\mathrm{R}$$

	$\|\hat{\Theta}\| \mathrel{\widetilde{\vdash}} R <:^+ [\Omega]R' \mathbin{/} W_0$	Subderivation
	$\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$	Given
	$\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W_0 \wedge [\Omega]\overrightarrow{\varphi}$	Given
	$\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W_0$	By inversion
	$\|\hat{\Theta}\| \vdash [\Omega]\overrightarrow{\varphi}\,\mathsf{true}$	$''$
	$\hat{\Theta} \vdash R <:^+ R' \mathbin{/} W_0'$	By i.h.
	$\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W_0 \leftrightarrow [\Omega]W_0'$	$''$
	$\|\hat{\Theta}\|;\cdot \vdash [\Omega]W_0'\ \mathsf{fixInstChk} \dashv \|\hat{\Theta}\|$	$''$
☞	$\hat{\Theta} \vdash R <:^+ R' \wedge \overrightarrow{\varphi} \mathbin{/} W_0' \wedge [\hat{\Theta}']\overrightarrow{\varphi}$	By $<:^+\wedge$R
	$\|\hat{\Theta}\| \vdash [\Omega]\overrightarrow{\varphi} \equiv [\Omega]\overrightarrow{\varphi} : \mathbb{B}$	By repeated Lemma C.34
	$\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W_0 \wedge [\Omega]\overrightarrow{\varphi} \leftrightarrow [\Omega]W_0' \wedge [\Omega]\overrightarrow{\varphi}$	By repeated $\mathrel{\widetilde{\models}}\leftrightarrow$Prp and $\mathrel{\widetilde{\models}}\leftrightarrow\wedge$
☞	$\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W_0 \wedge [\Omega]\overrightarrow{\varphi} \leftrightarrow [\Omega](W_0' \wedge \overrightarrow{\varphi})$	By def. of $[-]-$
☞	$\|\hat{\Theta}\|;\cdot \vdash [\Omega](W_0' \wedge \overrightarrow{\varphi})\ \mathsf{fixInstChk} \dashv \|\hat{\Theta}\|$	As $[\Omega]\overrightarrow{\varphi}$ is ground

- **Case** $\widetilde{<:^+}\mu$: Follows from i.h. part (7).
- **Case**

$$\frac{}{\|\hat{\Theta}\| \mathrel{\widetilde{\vdash}} \downarrow N <:^+ \downarrow \forall^{\mathsf{d}}\Xi.\,[\Omega]\overrightarrow{\varphi} \supset [\Omega]L \mathbin{/} \forall^{\mathsf{d}}\Xi.\,[\Omega]\overrightarrow{\varphi} \supset \underline{N <:^- [\Omega]L}}\ \widetilde{<:^+}\downarrow$$

☞	$\hat{\Theta} \vdash \downarrow N <:^+ \downarrow\underbrace{\forall^{\mathsf{d}}\Xi.\,\overrightarrow{\varphi} \supset L}_{N'} \mathbin{/} \forall^{\mathsf{d}}\Xi.\,\overrightarrow{\varphi} \supset \underline{N <:^- L}$	By $<:^+\downarrow$
	$\|\hat{\Theta}\| \mathrel{\widetilde{\models}} \forall^{\mathsf{d}}\Xi.\,[\Omega]\overrightarrow{\varphi} \supset \underline{N <:^- [\Omega]L} \leftrightarrow \forall^{\mathsf{d}}\Xi.\,[\Omega]\overrightarrow{\varphi} \supset \underline{N <:^- [\Omega]L}$	Lemma G.14
	$\|\hat{\Theta}\| \mathrel{\widetilde{\models}} \forall^{\mathsf{d}}\Xi.\,[\Omega]\overrightarrow{\varphi} \supset \underline{N <:^- [\Omega]L} \leftrightarrow \forall^{\mathsf{d}}\Xi.\,[\Omega]\overrightarrow{\varphi} \supset \underline{[\Omega]N <:^- [\Omega]L}$	As N ground
☞	$\|\hat{\Theta}\| \mathrel{\widetilde{\models}} \forall^{\mathsf{d}}\Xi.\,[\Omega]\overrightarrow{\varphi} \supset \underline{N <:^- [\Omega]L} \leftrightarrow [\Omega](\forall^{\mathsf{d}}\Xi.\,\overrightarrow{\varphi} \supset \underline{N <:^- L})$	$[-]-$ def.

The fixInstChk goal is easy (like in the $\widetilde{<:}^{+}1$ case, nothing is instantiated).

(3)
- **Case** $\widetilde{<:}^{-}\uparrow$: Similar to $\widetilde{<:}^{+}\downarrow$ case of part (2).
- **Case** $\widetilde{<:}^{-}\supset$L: Similar to $\widetilde{<:}^{+}\wedge$R case of part (2).
- **Case** $\widetilde{<:}^{-}\rightarrow$: Similar to $\widetilde{<:}^{+}\times$ case of part (2).

(4)
- **Case**

$$\frac{\begin{array}{lll}\alpha\circ\mathsf{inj}_1 \doteq \alpha_1 & \beta\circ\mathsf{inj}_1 \doteq \beta_1 & \Xi \widetilde{\vdash} \alpha_1;F_1 <:_\tau \beta_1;G_1\\ \alpha\circ\mathsf{inj}_2 \doteq \alpha_2 & \beta\circ\mathsf{inj}_2 \doteq \beta_2 & \Xi \widetilde{\vdash} \alpha_2;F_2 <:_\tau \beta_2;G_2\end{array}}{\Xi \widetilde{\vdash} \alpha;F_1\oplus F_2 <:_\tau \beta;G_1\oplus G_2}\ \widetilde{<:}_\tau\oplus$$

$\alpha\circ\mathsf{inj}_k \doteq \alpha_k$ Premises

$\beta\circ\mathsf{inj}_k \doteq \beta_k$ Premises

$\Xi \widetilde{\vdash} \alpha_k;F_k <:_\tau \beta_k;G_k$ Subderivations

$\Xi \rhd \alpha_k;F_k <:_\tau \beta_k;G_k$ By i.h. (twice)

☞ $\Xi \rhd \alpha;F <:_\tau \beta;G$ By Meas$\rhd<:/\oplus$

- **Case** $\widetilde{<:}_\tau\exists$L

$$\frac{\Xi,{}^{\mathsf{d}}\Xi' \widetilde{\vdash} (\top,q)\Rightarrow t;\underline{Q}\otimes\hat{P} <:_\tau ((o',q')\Rightarrow t');(\underline{P}\otimes\hat{P}')}{\Xi \widetilde{\vdash} (\mathtt{pk}({}^{\mathsf{d}}\Xi',\top),q)\Rightarrow t;\underline{\exists^{\mathsf{d}}\Xi'.Q}\otimes\hat{P} <:_\tau ((o',q')\Rightarrow t');(\underline{P}\otimes\hat{P}')}$$

$\Xi,{}^{\mathsf{d}}\Xi' \widetilde{\vdash} (\top,q)\Rightarrow t;\underline{Q}\otimes\hat{P} <:_\tau ((o',q')\Rightarrow t');(\underline{P}\otimes\hat{P}')$ Subder.

$\Xi,{}^{\mathsf{d}}\Xi' \rhd (\top,q)\Rightarrow t;\underline{Q}\otimes\hat{P} <:_\tau (o',q')\Rightarrow t';\underline{P}\otimes\hat{P}'$ By i.h.

☞ $\Xi \rhd (\mathtt{pk}({}^{\mathsf{d}}\Xi',\top),q)\Rightarrow t;\underline{\exists^{\mathsf{d}}\Xi'.Q}\otimes\hat{P} <:_\tau (o',q')\Rightarrow t';\underline{P}\otimes\hat{P}'$ Meas$\rhd<:/\exists$L

- **Case** $\widetilde{<:_\tau}\exists\mathrm{R}$

$$\frac{\overset{\mathsf{d}}{\div}\Xi \vdash \overrightarrow{t}/{}^{\mathsf{d}}\Xi' : {}^{\mathsf{d}}\Xi' \qquad \Xi \mathrel{\widetilde{\vdash}} (\top,q) \Rightarrow t;\underline{Q}\otimes\hat{P} <:_\tau (\top,q') \Rightarrow [\overrightarrow{t}/{}^{\mathsf{d}}\Xi']t';\underline{[\overrightarrow{t}/{}^{\mathsf{d}}\Xi']Q'}\otimes\hat{P}'}{\Xi \mathrel{\widetilde{\vdash}} (\top,q) \Rightarrow t;\underline{Q}\otimes\hat{P} <:_\tau (\mathtt{pk}({}^{\mathsf{d}}\Xi',\top),q') \Rightarrow t';\underline{\exists{}^{\mathsf{d}}\Xi'.Q'}\otimes\hat{P}'}$$

$\overset{\mathsf{d}}{\div}\Xi \vdash \overrightarrow{t}/{}^{\mathsf{d}}\Xi' : {}^{\mathsf{d}}\Xi'$ Premise

$\Xi \mathrel{\widetilde{\vdash}} (\top,q) \Rightarrow t;\underline{Q}\otimes\hat{P} <:_\tau (\top,q') \Rightarrow [\overrightarrow{t}/{}^{\mathsf{d}}\Xi']t';\underline{[\overrightarrow{t}/{}^{\mathsf{d}}\Xi']Q'}\otimes\hat{P}'$ Subderivation

$Q = R\wedge\overrightarrow{\varphi}$ Canonical form of Q

$\Xi \mathrel{\widetilde{\vdash}} (\top,q) \Rightarrow t;\underline{R\wedge\overrightarrow{\varphi}}\otimes\hat{P} <:_\tau (\top,q') \Rightarrow [\overrightarrow{t}/{}^{\mathsf{d}}\Xi']t';\underline{[\overrightarrow{t}/{}^{\mathsf{d}}\Xi']Q'}\otimes\hat{P}'$ By equality

$\Xi,\overrightarrow{\varphi} \mathrel{\widetilde{\vdash}} R <:^+ [\overrightarrow{t}/{}^{\mathsf{d}}\Xi']Q' \,/\, {}^{(\forall)}W$ By inversion on $\widetilde{<:_\tau}$Const

$\Xi,\overrightarrow{\varphi} \mathrel{\widetilde{\models}} {}^{(\forall)}W$ ″

$\Xi \mathrel{\widetilde{\vdash}} q \Rightarrow t;\hat{P} <:_\tau q' \Rightarrow [\overrightarrow{t}/{}^{\mathsf{d}}\Xi']t';\hat{P}'$ ″

$\Xi,\overrightarrow{\varphi} \mathrel{\widetilde{\vdash}} R <:^+ [\Xi,\overrightarrow{\varphi},\widehat{{}^{\mathsf{d}}\Xi'=\overrightarrow{t}}][\widehat{{}^{\mathsf{d}}\Xi'}/{}^{\mathsf{d}}\Xi']Q' \,/\, {}^{(\forall)}W$ By properties of $[-]-$

$\Xi,\overrightarrow{\varphi},\widehat{{}^{\mathsf{d}}\Xi'} \vdash R <:^+ [\widehat{{}^{\mathsf{d}}\Xi'}/{}^{\mathsf{d}}\Xi']Q' \,/\, {}^{(\forall)}W'$ By i.h.

$\|\Xi,\overrightarrow{\varphi},\widehat{{}^{\mathsf{d}}\Xi'}\| \mathrel{\widetilde{\models}} {}^{(\forall)}W \leftrightarrow [\Xi,\overrightarrow{\varphi},\widehat{{}^{\mathsf{d}}\Xi'}=\overrightarrow{t}]{}^{(\forall)}W'$ ″

$\Xi,\overrightarrow{\varphi};\cdot \vdash [\Xi,\overrightarrow{\varphi},\widehat{{}^{\mathsf{d}}\Xi'}=\overrightarrow{t}]{}^{(\forall)}W'\ \mathsf{fixInstChk} \dashv \Xi,\overrightarrow{\varphi}$ ″

$\Xi,\overrightarrow{\varphi},[\xi_{(\forall)W'}]\widehat{{}^{\mathsf{d}}\Xi'}=\overrightarrow{t};\cdot \vdash [\Xi,\overrightarrow{\varphi},\widehat{{}^{\mathsf{d}}\Xi'}=\overrightarrow{t}]{}^{(\forall)}W'\ \mathsf{fixInstChk} \dashv \Xi,\overrightarrow{\varphi},[\xi_{(\forall)W'}]\widehat{{}^{\mathsf{d}}\Xi'}=\overrightarrow{t}$ Stfd.

$\Xi,\overrightarrow{\varphi},[\xi_{(\forall)W'}]\widehat{{}^{\mathsf{d}}\Xi'}=\overrightarrow{t};\cdot \mathrel{\widetilde{\vdash}} [\Xi,\overrightarrow{\varphi},\widehat{{}^{\mathsf{d}}\Xi'}=\overrightarrow{t}]{}^{(\forall)}W'\ \mathsf{fixInstChk} \dashv \Xi,\overrightarrow{\varphi},[\xi_{(\forall)W'}]\widehat{{}^{\mathsf{d}}\Xi'}=\overrightarrow{t}$ Lemma G.5

$\Xi, \overrightarrow{\varphi}, [\xi_{(\forall)W'}]^{\widehat{\mathsf{d}\,\Xi'}}; \cdot \mathrel{\widetilde{\vdash}} {}^{(\forall)}W'\ \mathsf{fixInstChk} \dashv \Omega'$ By Lemma G.31 (fixInstChk Unapply) with Lemma G.35 and Lemma G.36

$\Omega' \xrightarrow{\mathsf{SMT}} \Xi, \overrightarrow{\varphi}, [\xi_{(\forall)W'}]^{\widehat{\mathsf{d}\,\Xi'} = \overrightarrow{t}}$ ″

☞ $\Xi, \overrightarrow{\varphi} \mathrel{\widetilde{\models}} {}^{(\forall)}W \leftrightarrow [\Omega']^{2\,(\forall)}W'$ By Lemma G.27 (Equiv. Solutions) and Lemma G.16 (Prob. Equiv. Transitive) and subst. props.

$\Xi, \overrightarrow{\varphi} \mathrel{\widetilde{\models}} [\Omega']^{2\,(\forall)}W'$ By Lemma G.22 (${}^{(\forall)}W$ Checking Respects Equiv)

at same height

$\Xi, \overrightarrow{\varphi} \models [\Omega']^{2\,(\forall)}W'$ By i.h.

$\Xi, \overrightarrow{\varphi}, [\xi_{(\forall)W'}]^{\widehat{\mathsf{d}\,\Xi'}}; \cdot \vdash {}^{(\forall)}W'\ \mathsf{fixInstChk} \dashv \Omega'$ By inversion and then by the rule

$\Xi \mathrel{\widetilde{\vdash}} q \Rightarrow t; \hat{P} <:_\tau q' \Rightarrow [\Omega']^2[\widehat{\mathsf{d}\,\Xi'}/{}^{\mathsf{d}}\Xi']t'; \hat{P}'$ By Lemma G.23 (Subtyping Respects Equiv) with Lemma G.27 (Equiv. Solutions) and properties of substitution

at same height

$\Xi \rhd q \Rightarrow t; \hat{P} <:_\tau q' \Rightarrow [\Omega']^2[\widehat{\mathsf{d}\,\Xi'}/{}^{\mathsf{d}}\Xi']t'; \hat{P}'$ By i.h.

☞ $\Xi \rhd (\top, q) \Rightarrow t; \underline{Q} \otimes \hat{P} <:_\tau (\mathtt{pk}({}^{\mathsf{d}}\Xi', \top), q') \Rightarrow t'; \underline{\exists {}^{\mathsf{d}}\Xi'.\, Q'} \otimes \hat{P}'$ Meas$\rhd$<:/$\exists$R

- **Case**

$$\frac{\Xi, \overrightarrow{\varphi} \mathrel{\widetilde{\vdash}} R <:^{+} Q' \mathbin{/} {}^{(\forall)}W \qquad \Xi, \overrightarrow{\varphi} \mathrel{\widetilde{\models}} {}^{(\forall)}W \qquad \Xi \mathrel{\widetilde{\vdash}} q \Rightarrow t; \hat{P} <:_{\tau} q' \Rightarrow t'; \hat{P}'}{\Xi \mathrel{\widetilde{\vdash}} (\top, q) \Rightarrow t; \underline{R \wedge \overrightarrow{\varphi}} \otimes \hat{P} <:_{\tau} (\top, q') \Rightarrow t'; \underline{Q'} \otimes \hat{P}'} \mathrel{\widetilde{<:}}_{\tau}\text{Const}$$

$\Xi, \overrightarrow{\varphi} \mathrel{\widetilde{\vdash}} R <:^{+} Q' \mathbin{/} {}^{(\forall)}W$ Subderivation

$\Xi, \overrightarrow{\varphi} \mathrel{\widetilde{\models}} {}^{(\forall)}W$ Subderivation

$\Xi, \overrightarrow{\varphi} \vdash R <:^{+} Q'$ By i.h.

$\Xi \mathrel{\widetilde{\vdash}} q \Rightarrow t; \hat{P} <:_{\tau} q' \Rightarrow t'; \hat{P}'$ Subderivation

$\Xi \rhd q \Rightarrow t; \hat{P} <:_{\tau} q' \Rightarrow t'; \hat{P}'$ By i.h.

☞ $\Xi \rhd (\top, q) \Rightarrow t; \underline{R \wedge \overrightarrow{\varphi}} \otimes \hat{P} <:_{\tau} (\top, q') \Rightarrow t'; \underline{Q'} \otimes \hat{P}'$ By Meas$\rhd$<:/Const

- **Case**

$$\frac{\Xi, a \mathrel{\overset{\mathsf{d}}{\div}} \tau, a\,\mathrm{Id} \mathrel{\widetilde{\vdash}} q \Rightarrow t; \hat{I} <:_{\tau} q \Rightarrow t'; \hat{I}}{\Xi \mathrel{\widetilde{\vdash}} (a, q) \Rightarrow t; \mathrm{Id} \otimes \hat{I} <:_{\tau} (a, q) \Rightarrow t'; \mathrm{Id} \otimes \hat{I}} \mathrel{\widetilde{<:}}_{\tau}\mathrm{Id}$$

Straightforward.

- **Case**

$$\frac{\overset{\mathsf{d}}{\div} \Xi \vdash u \equiv t : \tau}{\Xi \mathrel{\widetilde{\vdash}} () \Rightarrow u; I <:_{\tau} () \Rightarrow t; I} \mathrel{\widetilde{<:}}_{\tau} I$$

Straightforward.

(5) - **Case**

$$\frac{\overset{\mathsf{d}}{\div} \Theta \vdash \overrightarrow{t} \mathbin{/}^{\mathsf{d}} \Xi : {}^{\mathsf{d}}\Xi \qquad \Theta \mathrel{\widetilde{\vdash}} R <:^{+} [\overrightarrow{t} \mathbin{/}^{\mathsf{d}} \Xi] Q \mathbin{/} W}{\Theta \mathrel{\widetilde{\vdash}} R <:^{+} \exists^{\mathsf{d}} \Xi. Q \mathbin{/} W} \mathrel{\widetilde{<:}}^{+}\exists\mathrm{R}$$

$\Theta, \widehat{{}^{\mathsf{d}}\Xi} \vdash R <:^{+} [\widehat{{}^{\mathsf{d}}\Xi}/{}^{\mathsf{d}}\Xi]Q \;/\; W'$ — By i.h. part (2)

$\Theta; \cdot \vdash [\Theta, \widehat{{}^{\mathsf{d}}\Xi} = \overrightarrow{t}]W'\ \mathsf{fixInstChk} \dashv \Theta$ — ″

$\Theta \mathrel{\widetilde{\models}} W \leftrightarrow [\Theta, \widehat{{}^{\mathsf{d}}\Xi} = \overrightarrow{t}]W'$ — ″

$\Theta, [\xi_{W'}]\widehat{{}^{\mathsf{d}}\Xi} = \overrightarrow{t}; \cdot \vdash [\Theta, \widehat{{}^{\mathsf{d}}\Xi} = \overrightarrow{t}]W'\ \mathsf{fixInstChk} \dashv \Theta, [\xi_{W'}]\widehat{{}^{\mathsf{d}}\Xi} = \overrightarrow{t}$ — Weakening (stfd.)

$\Theta, [\xi_{W'}]\widehat{{}^{\mathsf{d}}\Xi} = \overrightarrow{t}; \cdot \mathrel{\widetilde{\vdash}} [\Theta, \widehat{{}^{\mathsf{d}}\Xi} = \overrightarrow{t}]W'\ \mathsf{fixInstChk} \dashv \Theta, [\xi_{W'}]\widehat{{}^{\mathsf{d}}\Xi} = \overrightarrow{t}$ — Lemma G.5

$\Theta, [\xi_{W'}]\widehat{{}^{\mathsf{d}}\Xi}; \cdot \mathrel{\widetilde{\vdash}} W'\ \mathsf{fixInstChk} \dashv \Theta, [\xi_{W'}]\widehat{{}^{\mathsf{d}}\Xi} = \overrightarrow{u}$ — By Lemma G.31 with Lemma G.35 and Lemma G.36

$\Theta, [\xi_{W'}]\widehat{{}^{\mathsf{d}}\Xi} = \overrightarrow{u} \xrightarrow{\mathsf{SMT}} \Theta, [\xi_{W'}]\widehat{{}^{\mathsf{d}}\Xi} = \overrightarrow{t}$ — ″

$\Theta \mathrel{\widetilde{\models}} W$ — Given

$\Theta \mathrel{\widetilde{\models}} [\Theta, \widehat{{}^{\mathsf{d}}\Xi} = \overrightarrow{t}]W' \leftrightarrow [\Theta, \widehat{{}^{\mathsf{d}}\Xi} = \overrightarrow{u}]W'$ — By Lemma G.27 and subst. prop.

$\Theta \mathrel{\widetilde{\models}} W \leftrightarrow [\Theta, \widehat{{}^{\mathsf{d}}\Xi} = \overrightarrow{u}]W'$ — By Lemma G.16

$\Theta \mathrel{\widetilde{\models}} [\Theta, \widehat{{}^{\mathsf{d}}\Xi} = \overrightarrow{u}]W'$ — By Lemma G.22 at same height

$\Theta \models [\Theta, \widehat{{}^{\mathsf{d}}\Xi} = \overrightarrow{u}]W'$ — By i.h. part (1)

$\Theta, [\xi_{W'}]\widehat{{}^{\mathsf{d}}\Xi}; \cdot \vdash W'\ \mathsf{fixInstChk} \dashv \Theta, [\xi_{W'}]\widehat{{}^{\mathsf{d}}\Xi} = \overrightarrow{u}$ — By inversion and by rule

☞ $\Theta \vdash R <:^{+} \exists {}^{\mathsf{d}}\Xi.\, Q$ — By $<:^{+}$

Note that this is a similar pattern as the $\widetilde{<:}_{\tau}\exists$R case of part (4) of the current proof and the more complicated Decl$\Leftarrow\exists$ case of part (4) of Lemma G.39 (Aux.

Alg. Typing Complete).

- The case where $P \neq \exists$ is similar but simpler (no need to use Lemma G.31).

(6) Similar to part (5).

(7) Straightforward. Use Lemma C.27 (Prop. Truth Equiv. Relation) and Lemma C.34 (Ix. Equiv. Reflexive). □

Appendix G.4 Algorithmic Typing

Lemma G.39 (Aux. Alg. Typing Complete)**.**

(1) *If* $\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$ *then* $\Theta;\Gamma \lhd \chi$.

(2) *If* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} h \Rightarrow P$ *then* $\Theta;\Gamma \rhd h \Rightarrow P$.

(3) *If* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} g \Rightarrow \uparrow P$
then there exists P' *such that* $\Theta;\Gamma \rhd g \Rightarrow \uparrow P'$ *and* $\Theta \vdash P \equiv^{+} P'$.

(4) *If* $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$ *and* $\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow [\Omega]P \mathbin{/} \chi$ *and* $\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\lhd}} \chi$
and $\hat{\Theta}$ present *and* $[\hat{\Theta}]P = P$ *and* $\overline{\hat{\Theta}} \rhd P$ type$[\xi]$
then there exist χ', Δ', *and* Ω' *such that* $\hat{\Theta};\Gamma \vdash v \Leftarrow P \mathbin{/} \chi' \dashv \Delta'$
and $\overline{\hat{\Theta}}, \Delta' \vdash \chi'$ Wf$[\xi']$ *and* $\|\hat{\Theta}\|, [\xi']\Delta';\Gamma \vdash [\Omega]\chi'$ fixInstChk $\dashv \|\hat{\Theta}\|, \Omega'$
and $\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} \chi \leftrightarrow [\Omega, \Omega']^2\chi'$.

(5) *If* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow P \mathbin{/} \chi$ *and* $\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$ *then* $\Theta;\Gamma \rhd v \Leftarrow P$.

(6) *If* $\Theta;\Gamma \mathrel{\widetilde{\vdash}} e \Leftarrow N$ *then* $\Theta;\Gamma \rhd e \Leftarrow N$.

(7) *If* $\Theta;\Gamma;[P] \mathrel{\widetilde{\vdash}} \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$ *then* $\Theta;\Gamma;[P] \rhd \{r_i \Rightarrow e_i\}_{i \in I} \Leftarrow N$.

(8) *If* $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$ *and* $\|\hat{\Theta}\|;\Gamma;[[\Omega]M] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \mathbin{/} \chi$ *and* $\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\lhd}} \chi$
and $\hat{\Theta}$ present *and* $\overline{\hat{\Theta}} \rhd M$ type$[\xi]$ *and* $[\hat{\Theta}]M = M$
then there exist P', χ', Δ', *and* Ω' *such that* $\hat{\Theta};\Gamma;[M] \vdash s \Rightarrow \uparrow P' \mathbin{/} \chi' \dashv \Delta'$
and $\overline{\hat{\Theta}}, \Delta' \vdash \chi'$ Wf$[\xi']$ *and* $\|\hat{\Theta}\|, [\xi']\Delta'; \Gamma \vdash [\Omega]\chi'$ fixInstChk $\dashv \|\hat{\Theta}\|, \Omega'$
and $\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} \chi \leftrightarrow [\Omega, \Omega']^2\chi'$ *and* $\|\hat{\Theta}\| \vdash P \equiv^+ [\Omega, \Omega']^2 P'$.

(9) *If* $\Theta;\Gamma;[N] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \mathbin{/} \chi$ *and* $\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$
then there exists P' *such that* $\Theta;\Gamma;[N] \rhd s \Rightarrow \uparrow P'$ *and* $\Theta \vdash P \equiv^+ P'$.

Proof. By induction on the sum of the height of the given semideclarative typing derivation and the height of the given semideclarative constraint checking derivation. Every part is mutually recursive.

(1)

- **Case**

$$\dfrac{}{\Theta;\Gamma \mathrel{\widetilde{\lhd}} \cdot}\ \widetilde{\lhd}\text{Empty}$$

$\Theta;\Gamma \lhd \cdot$ By $\lhd$Empty

- **Case**

$$\dfrac{\Theta;\Gamma \mathrel{\widetilde{\vdash}} e \Leftarrow N \qquad \Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi_0}{\Theta;\Gamma \mathrel{\widetilde{\lhd}} \underbrace{(e \Leftarrow N), \chi_0}_{\chi}}\ \widetilde{\lhd}\text{NegChk}$$

$\Theta;\Gamma \mathrel{\widetilde{\vdash}} e \Leftarrow N$	Subderivation
$\Theta;\Gamma \rhd e \Leftarrow N$	By i.h.
$\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi_0$	Subderivation
$\Theta;\Gamma \lhd \chi_0$	By i.h.
$\Theta;\Gamma \lhd (e \Leftarrow N), \chi_0$	By $\lhd$NegChk

- **Case**
$$\frac{\Theta \mathrel{\widetilde{\models}} W \qquad \Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi_0}{\Theta;\Gamma \mathrel{\widetilde{\lhd}} W, \chi_0}\ \widetilde{\lhd} W$$

  $\Theta \mathrel{\widetilde{\models}} W$ Subderivation

  $\Theta \models W$ By Lemma G.38 (Aux. Alg. Sub. Complete)

  $\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi_0$ Subderivation

  $\Theta;\Gamma \lhd \chi_0$ By i.h.

  $\Theta;\Gamma \lhd W, \chi_0$ By $\lhd W$

(2) - **Case**
$$\frac{(x : R) \in \Gamma}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} x \Rightarrow R}\ \widetilde{\Rightarrow}\text{Var}$$

  $(x : R) \in \Gamma$ Premise

  $\Theta;\Gamma \rhd x \Rightarrow R$ By Alg$\Rightarrow$Var

- **Case**
$$\frac{\overline{\Theta} \vdash P\ \mathsf{type}[\xi] \qquad \Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow P \mid \chi \qquad \Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} (v : P) \Rightarrow P}\ \widetilde{\Rightarrow}\text{ValAnnot}$$

  $\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow P \mid \chi$ Subderivation

  $\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$ Subderivation

  $\Theta;\Gamma \rhd v \Leftarrow P$ By i.h.

  ☞ $\Theta;\Gamma \rhd (v : P) \Rightarrow P$ By Alg$\Rightarrow$ValAnnot

(3) - **Case**
$$\frac{\Theta;\Gamma \mathrel{\widetilde{\vdash}} h \Rightarrow \downarrow N \qquad \Theta;\Gamma;[N] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \mid \chi \qquad \Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} h(s) \Rightarrow \uparrow P}\ \widetilde{\Rightarrow}\text{App}$$

$$\Theta;\Gamma \mathrel{\widetilde{\vdash}} h \Rightarrow \downarrow N \qquad \text{Subderivation}$$

$$\Theta;\Gamma \rhd h \Rightarrow \downarrow N \qquad \text{By i.h.}$$

$$\Theta;\Gamma;[N] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \mathbin{/} \chi \qquad \text{Subderivation}$$

$$\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi \qquad \text{Subderivation}$$

$$\Theta;\Gamma;[N] \rhd s \Rightarrow \uparrow P' \qquad \text{By i.h.}$$

☞ $$\Theta \vdash P \equiv^{+} P' \qquad ''$$

☞ $$\Theta;\Gamma \rhd h(s) \Rightarrow \uparrow P' \qquad \text{By Alg}\Rightarrow\text{App}$$

- **Case** $\widetilde{\Rightarrow}$ExpAnnot: Similar to $\widetilde{\Rightarrow}$ValAnnot case of part (2).

(4) - **Case**

$$\frac{(x : R') \in \Gamma \qquad \|\hat{\Theta}\| \mathrel{\widetilde{\vdash}} R' <:^{+} [\Omega]R \mathbin{/} W}{\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\vdash}} x \Leftarrow [\Omega]R \mathbin{/} W}\ \widetilde{\Leftarrow}\text{Var}$$

	$\|\hat{\Theta}\| \mathrel{\widetilde{\vdash}} R' <:^+ [\Omega]R \,/\, W$	Subderivation
	$\hat{\Theta}$ present	Given
	$\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$	Given
	$\overline{\hat{\Theta}} \rhd R$ type$[\xi]$	Given
	$[\hat{\Theta}]R = R$	Given
	$\overline{\|\hat{\Theta}\|} \vdash \Gamma$ ctx	Presupposed derivation
	$(x : R') \in \Gamma$	Premise
	R' ground	By inversion on Γ WF
	$\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\lhd}} W$	Given
	$\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W$	By inversion
	$\hat{\Theta} \vdash R' <:^+ R \,/\, W'$	By Lemma G.38
	$\|\hat{\Theta}\| \mathrel{\widetilde{\models}} W \leftrightarrow [\Omega]W'$	$''$
	$\|\hat{\Theta}\|;\cdot \vdash [\Omega]W'$ fixInstChk $\dashv \|\hat{\Theta}\|$	$''$
	$\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} \cdot \leftrightarrow \cdot$	By $\widetilde{\lhd}\leftrightarrow$Empty
☞	$\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} W \leftrightarrow [\Omega]^2 W'$	By $\widetilde{\lhd}\leftrightarrow W$, Ω present
	$\hat{\Theta};\Gamma \vdash x \Leftarrow R \,/\, W' \dashv \cdot$	By Alg$\Leftarrow$Var
	Let $\Delta' = \cdot$.	
	Let $\Omega' = \cdot$.	
☞	$\|\hat{\Theta}\|, [\xi']\Delta';\Gamma \vdash [\Omega]W'$ fixInstChk $\dashv \|\hat{\Theta}\|, \Omega'$	By equalities
☞	$\hat{\Theta};\Gamma \vdash x \Leftarrow R \,/\, W' \dashv \Delta'$	By equality

- **Case**

$$\frac{}{\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\vdash}} \langle\rangle \Leftarrow 1 \mathbin{/} \cdot}\ \widetilde{\Leftarrow}1$$

☞	$\hat{\Theta};\Gamma \vdash \langle\rangle \Leftarrow 1 \mathbin{/} \cdot \dashv \cdot$	By Alg⇐1
	$\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$	Given
☞	$\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} \cdot \leftrightarrow \cdot$	By $\widetilde{\lhd}\leftrightarrow$Empty
	Let $\Delta' = \cdot$.	
	Let $\Omega' = \Omega$.	
☞	$\|\hat{\Theta}\|, [\xi']\Delta';\Gamma \vdash [\Omega]\cdot \ \mathsf{fixInstChk} \dashv \|\hat{\Theta}\|, \Omega'$	By rules, equalities

- **Case**

$$\frac{\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\vdash}} v_1 \Leftarrow [\Omega]R_1 \mathbin{/} \chi_1 \qquad \|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\vdash}} v_2 \Leftarrow [\Omega]R_2 \mathbin{/} \chi_2}{\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\vdash}} \langle v_1, v_2\rangle \Leftarrow [\Omega]R_1 \times [\Omega]R_2 \mathbin{/} \chi_1, \chi_2}\ \widetilde{\Leftarrow}\times$$

$\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\vdash}} v_1 \Leftarrow [\Omega]R_1 \mathbin{/} \chi_1$	Subderivation
$\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\lhd}} \chi_1, \chi_2$	Given
$\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\lhd}} \chi_1$	By inversion
$\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\lhd}} \chi_2$	$''$
$\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$	Given
$\hat{\Theta};\Gamma \vdash v_1 \Leftarrow R_1 \mathbin{/} \chi_1' \dashv \Delta_1$	By i.h.
$\|\hat{\Theta}\|, [\xi_{\chi_1'}]\Delta_1;\Gamma \vdash [\Omega]\chi_1' \ \mathsf{fixInstChk} \dashv \|\hat{\Theta}\|, \Omega_1'$	$''$
$\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} \chi_1 \leftrightarrow [\Omega_1']^2\chi_1'$	$''$

$$\overline{\hat{\Theta}} \rhd R_1 \times R_2 \;\mathsf{type}[\xi] \qquad \text{Given}$$

$$\overline{\hat{\Theta}} \rhd R_2 \;\mathsf{type}[_] \qquad \text{By inversion}$$

$$\|\hat{\Theta}\|; \Gamma \mathrel{\widetilde{\vdash}} v_2 \Leftarrow [\Omega]R_2 \,/\, \chi_2 \qquad \text{Subderivation}$$

$$\hat{\Theta}; \Gamma \vdash v_2 \Leftarrow R_2 \,/\, \chi_2' \dashv \Delta_2 \qquad \text{By i.h.}$$

$$\|\hat{\Theta}\|, [\xi_{\chi_2'}]\Delta_2; \Gamma \vdash [\Omega]\chi_2' \;\mathsf{fixInstChk} \dashv \|\hat{\Theta}\|, \Omega_2' \qquad ''$$

$$\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} \chi_2 \leftrightarrow [\Omega_2']^2\chi_2' \qquad ''$$

☞ $\hat{\Theta}; \Gamma \vdash \langle v_1, v_2\rangle \Leftarrow R_1 \times R_2 \,/\, \chi_1', \chi_2' \dashv \Delta_1, \Delta_2$ By Alg$\Leftarrow\times$

$\|\hat{\Theta}\|, [\xi_{\chi_1'}]\Delta_1, [\xi_{\chi_2'}]\Delta_2; \Gamma \vdash [\Omega]\chi_1', [\Omega]\chi_2' \;\mathsf{fIC} \dashv \|\hat{\Theta}\|, \Omega_1', \Omega_2'$ By Lemma G.2

☞ $\|\hat{\Theta}\|, [\xi_{\chi_1', \chi_2'}](\Delta_1, \Delta_2); \Gamma \vdash [\Omega](\chi_1', \chi_2') \;\mathsf{fIC} \dashv \|\hat{\Theta}\|, \Omega_1', \Omega_2'$ Distribute

☞ $\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} \chi_1, \chi_2 \leftrightarrow [\Omega, \Omega_1, \Omega_2]^2(\chi_1', \chi_2')$ Straightforward

- **Case** $\mathrel{\widetilde{\Leftarrow}}+_k$: Straightforward.

- **Case**

$$\frac{\|\hat{\Theta}\|; \Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow [\Omega]R \,/\, \chi_0}{\|\hat{\Theta}\|; \Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow [\Omega]R \wedge [\Omega]\overrightarrow{\varphi} \,/\, [\Omega]\overrightarrow{\varphi}, \chi_0} \mathrel{\widetilde{\Leftarrow}}\wedge$$

	$\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow [\Omega]R \,/\, \chi_0$	Subderivation
	$\hat{\Theta};\Gamma \vdash v \Leftarrow R \,/\, \chi_0' \dashv \Delta'$	By i.h.
	$\|\hat{\Theta}\|,\Delta' \vdash \lfloor [\Omega]\chi_0' \rfloor$ fixInst $\dashv \|\hat{\Theta}\|,\Omega'$	$''$
	$\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} \chi_0 \leftrightarrow [\Omega,\Omega']^2\chi_0'$	$''$
☞	$\hat{\Theta};\Gamma \vdash v \Leftarrow R \wedge \overrightarrow{\varphi} \,/\, \overrightarrow{\varphi},\chi_0' \dashv \Delta'$	By Alg$\Leftarrow\wedge$
☞	$\|\hat{\Theta}\|,\Delta' \vdash \lfloor [\Omega](\overrightarrow{\varphi},\chi_0') \rfloor$ fixInst $\dashv \|\hat{\Theta}\|,\Omega'$	As $[\Omega]\overrightarrow{\varphi}$ground
	$\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} [\Omega]\overrightarrow{\varphi} \leftrightarrow [\Omega]\overrightarrow{\varphi}$	By reflexivity
	$\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} [\Omega]\overrightarrow{\varphi} \leftrightarrow [\Omega,\Omega']\overrightarrow{\varphi}$	As $dom(\Omega') \cap FV(\overrightarrow{\varphi}) = \emptyset$
	$\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} [\Omega]\overrightarrow{\varphi},\chi_0 \leftrightarrow [\Omega,\Omega']\overrightarrow{\varphi},[\Omega,\Omega']^2\chi_0'$	Append
☞	$\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} [\Omega]\overrightarrow{\varphi},\chi_0 \leftrightarrow [\Omega,\Omega']^2(\overrightarrow{\varphi},\chi_0')$	Property of subst.

- **Case**

$$\frac{\overset{\mathsf{d}\cdot}{\div}\|\hat{\Theta}\| \vdash \overrightarrow{t}/^{\mathsf{d}}\Xi : {}^{\mathsf{d}}\Xi \qquad \|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow [\overrightarrow{t}/^{\mathsf{d}}\Xi][\Omega]Q \,/\, \chi}{\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow (\exists^{\mathsf{d}}\Xi.\,[\Omega]Q) \,/\, \chi}\,{}^{\sim}\!\Leftarrow\!\exists$$

$\overset{\mathsf{d}\cdot}{\div}\|\hat{\Theta}\| \vdash \overrightarrow{t}/^{\mathsf{d}}\Xi : {}^{\mathsf{d}}\Xi$	Premise
$\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow [\overrightarrow{t}/^{\mathsf{d}}\Xi][\Omega]Q \,/\, \chi$	Subderivation
$\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow [\Omega,\widehat{{}^{\mathsf{d}}\Xi}=\overrightarrow{t}]([\widehat{{}^{\mathsf{d}}\Xi}/^{\mathsf{d}}\Xi]Q) \,/\, \chi$	By properties of subst.
$\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\lhd}} \chi$	Given
$\hat{\Theta}$ present	Given
$\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$	Given
$\hat{\Theta},\widehat{{}^{\mathsf{d}}\Xi} \xrightarrow{\mathsf{SMT}} \Omega,\widehat{{}^{\mathsf{d}}\Xi}=\overrightarrow{t}$	By repeated $\xrightarrow{\sim}$Solve

$$\hat{\Theta}, \widehat{{}^{\mathsf{d}}\Xi}; \Gamma \vdash v \Leftarrow [\widehat{{}^{\mathsf{d}}\Xi}/{}^{\mathsf{d}}\Xi]Q \;/\; \chi' \dashv \Delta'' \qquad \text{By i.h.}$$

$$\|\hat{\Theta}\|, [\xi_{\chi'}]\Delta''; \Gamma \vdash [\Omega, \widehat{{}^{\mathsf{d}}\Xi = \overrightarrow{t}}]\chi' \;\mathsf{fixInstChk} \dashv \|\hat{\Theta}\|, \Omega'' \qquad ''$$

$$\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} \chi \leftrightarrow [\Omega, \widehat{{}^{\mathsf{d}}\Xi = \overrightarrow{t}}, \Omega'']^2\chi' \qquad ''$$

☞ $$\hat{\Theta}; \Gamma \vdash v \Leftarrow \exists\,{}^{\mathsf{d}}\Xi.\, Q \;/\; \chi' \dashv \widehat{{}^{\mathsf{d}}\Xi}, \Delta'' \qquad \text{By Alg}\Leftarrow\exists$$

$\|\hat{\Theta}\|; \Gamma \vdash v \Leftarrow [\Omega](\exists\,{}^{\mathsf{d}}\Xi.\, Q) \;/\; [\Omega]\chi' \dashv \widehat{{}^{\mathsf{d}}\Xi}, \Delta''$ Substitution lemma

$\widehat{{}^{\mathsf{d}}\Xi}, \Delta'' \vdash \xi_{[\Omega](\exists\,{}^{\mathsf{d}}\Xi.\, Q)} - \|\hat{\Theta}\| \angle \xi_{[\Omega]\chi'} - \|\hat{\Theta}\|$ By Lemma G.35 (Main Complete)

$$\begin{aligned}\widehat{{}^{\mathsf{d}}\Xi}, \Delta'' &\subseteq cl(\xi_{[\Omega]\chi'} - \|\hat{\Theta}\|)((FV(\xi_{[\Omega]\chi'} - \|\hat{\Theta}\|)) - (\widehat{{}^{\mathsf{d}}\Xi}, \Delta'')) \qquad \text{By inversion}\\ &= cl(\xi_{[\Omega]\chi'} - \|\hat{\Theta}\|)(\emptyset)\\ &\qquad\qquad \text{As } FV(\xi_{[\Omega]\chi'} - \|\hat{\Theta}\|) \subseteq \widehat{{}^{\mathsf{d}}\Xi}, \Delta''\end{aligned}$$

$\widehat{{}^{\mathsf{d}}\Xi}, \Delta'' = cl(\xi_{[\Omega]\chi'} - \|\hat{\Theta}\|)(\emptyset)$ Also as $FV(\xi_{[\Omega]\chi'} - \|\hat{\Theta}\|) \subseteq \widehat{{}^{\mathsf{d}}\Xi}, \Delta''$

$$\|\hat{\Theta}\|, [\xi_{[\Omega]\chi'}]\Delta''; \Gamma \mathrel{\widetilde{\vdash}} [\Omega, \widehat{{}^{\mathsf{d}}\Xi = \overrightarrow{t}}]\chi' \;\mathsf{fIC} \dashv \|\hat{\Theta}\|, \Omega'' \qquad \text{Lemma G.8}$$

$$\|\hat{\Theta}\|, [\xi_{[\Omega]\chi'}]\widehat{{}^{\mathsf{d}}\Xi = \overrightarrow{t}}, [\xi_{[\Omega]\chi'}]\Delta''; \Gamma \mathrel{\widetilde{\vdash}} [\Omega, \widehat{{}^{\mathsf{d}}\Xi = \overrightarrow{t}}]\chi' \;\mathsf{fIC} \dashv \|\hat{\Theta}\|, [\xi_{[\Omega]\chi'}]\widehat{{}^{\mathsf{d}}\Xi = \overrightarrow{t}}, \Omega'' \qquad \text{Stfd.}$$

$\|\hat{\Theta}\|, [\xi_{[\Omega]\chi'}](\widehat{{}^{\mathsf{d}}\Xi}, \Delta''); \Gamma \mathrel{\widetilde{\vdash}} [\Omega]\chi' \;\mathsf{fixInstChk} \dashv \Omega_0'$ By Lemma G.31 (fixInstChk Unapply)

$$\Omega_0' \xrightarrow{\mathsf{SMT}} \|\hat{\Theta}\|, \widehat{{}^{\mathsf{d}}\Xi = \overrightarrow{t}}, \Omega'' \qquad ''$$

with $\|\hat{\Theta}\|, \widehat{{}^{\mathsf{d}}\Xi}, \Delta''$ as $\hat{\Theta}$;

$\|\hat{\Theta}\|, \widehat{{}^{\mathsf{d}}\Xi = \overrightarrow{t}}, \Delta''$ as $\hat{\Theta}'$;

$[\Omega]\chi'$ as χ;

$\xi_{[\Omega]\chi'}$ as ξ;

$\|\hat{\Theta}\|, [\xi_{[\Omega]\chi'}]\widehat{{}^{\mathsf{d}}\Xi = \overrightarrow{t}}, \Omega''$ as Ω'

$\Omega_0' = \|\hat{\Theta}\|, \Omega'$ By inversion

$\Omega, \Omega' \xrightarrow{\mathsf{SMT}} \Omega, \widehat{{}^{\mathsf{d}}\Xi = \overrightarrow{t}}, \Omega''$ Straightforward

☞ $\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} \chi \leftrightarrow [\Omega, \Omega']^2 \chi'$ By Lemma G.27 (Equiv. Solutions)
and Lemma F.6 (Ext. Transitive)
and subst. props.

$\|\hat{\Theta}\|; \Gamma \mathrel{\widetilde{\lhd}} [\Omega, \Omega']^2 \chi'$ By Lemma G.26 (Typing Respects Equiv)
at same height

$\|\hat{\Theta}\|; \Gamma \lhd [\Omega, \Omega']^2 \chi'$ By i.h.

☞ $\|\hat{\Theta}\|, [\xi_{[\Omega]\chi'}](\widehat{{}^{\mathsf{d}}\Xi}, \Delta''); \Gamma \vdash [\Omega]\chi' \ \mathsf{fixInstChk} \dashv \|\hat{\Theta}\|, \Omega'$ By inversion
and then by rule
and subst. properties

- **Case**

$$\frac{\begin{array}{c}[\Omega](\mathscr{M}(F)) \rightsquigarrow [\Omega]\overrightarrow{\alpha}; \overrightarrow{\tau} \\ {}^{\mathsf{d}\div}\|\hat{\Theta}\| \vdash \wr [\Omega]\overrightarrow{\alpha}; [\Omega]F; [\Omega](\mathscr{M}(F))([\Omega]F) \wr \doteq {}^{\mathsf{d}}\Theta; R \\ \|\hat{\Theta}\|; \Gamma \mathrel{\widetilde{\vdash}} v_0 \Leftarrow \exists^{\mathsf{d}}\Theta.R \wedge {}^{\mathsf{d}}\Theta \ / \ \chi\end{array}}{\|\hat{\Theta}\|; \Gamma \mathrel{\widetilde{\vdash}} \mathtt{into}(v_0) \Leftarrow \{\nu : \mu[\Omega]F \mid [\Omega](\mathscr{M}(F))\} \ / \ \chi} \mathrel{\widetilde{\Leftarrow}}\mu$$

$\widehat{\Theta} \rhd \{\nu : \mu F \mid \mathscr{M}(F)\} \ \mathsf{type}[\xi]$ Given

$FV(\mathscr{M}(F)) = \emptyset$ By inversion

$\overset{\mathsf{d}}{\div}\|\hat{\Theta}\| \vdash \wr[\Omega]\overrightarrow{\alpha};[\Omega]F;[\Omega]\mathscr{M}([\Omega]F)\wr \doteq {}^{\mathsf{d}}\Theta;R$ Subderivation

$\|\overset{\mathsf{d}}{\div}\hat{\Theta}\| \vdash \wr[\Omega]\overrightarrow{\alpha};[\Omega]F;[\Omega]\mathscr{M}([\Omega]F)\wr \doteq {}^{\mathsf{d}}\Theta;R$ As $\overset{\mathsf{d}}{\div}-$ and $\|-\|$ commute

$\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$ Given

$\overset{\mathsf{d}}{\div}\hat{\Theta} \xrightarrow{\mathsf{SMT}} \overset{\mathsf{d}}{\div}\Omega$ By Lemma F.8

$\hat{\Theta}$ present Given

Ω present As $\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$

$\overset{\mathsf{d}}{\div}\hat{\Theta} \vdash \wr\overrightarrow{\alpha};F;\mathscr{M}(F)\wr \doteq {}^{\mathsf{d}}\Theta';R'$ By Lemma F.14 (Uncomplete Unrolling)

$[\Omega](R', {}^{\mathsf{d}}\Theta') = (R, {}^{\mathsf{d}}\Theta)$ ″

$\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\vdash}} v_0 \Leftarrow \exists {}^{\mathsf{d}}\Theta.R \wedge {}^{\mathsf{d}}\Theta \mathbin{/} \chi$ Subderivation

$\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\vdash}} v_0 \Leftarrow \exists [\Omega]{}^{\mathsf{d}}\Theta'.[\Omega]R' \wedge [\Omega]{}^{\mathsf{d}}\Theta' \mathbin{/} \chi$ By equalities

$\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\vdash}} v_0 \Leftarrow [\Omega](\exists {}^{\mathsf{d}}\Theta'.R' \wedge {}^{\mathsf{d}}\Theta') \mathbin{/} \chi$ Property of subst.

$\overline{\hat{\Theta}} \rhd \exists {}^{\mathsf{d}}\Theta'.R \wedge {}^{\mathsf{d}}\Theta'\ \mathsf{type}[_]$ By Lemma F.3

$[\hat{\Theta}](R', \hat{\Theta}') = (R', \hat{\Theta}')$ By Lemma F.15

$\|\hat{\Theta}\|;\Gamma \mathrel{\widetilde{\lhd}} \chi$ Given

$\hat{\Theta} \xrightarrow{\mathsf{SMT}} \Omega$ Given

$\hat{\Theta};\Gamma \vdash v_0 \Leftarrow \exists {}^{\mathsf{d}}\Theta.R \wedge {}^{\mathsf{d}}\Theta \mathbin{/} \chi' \dashv \Delta'$ By i.h.

☞ $\|\hat{\Theta}\|, \Delta';\Gamma \vdash \lfloor [\Omega]\chi' \rfloor\ \mathsf{fixInstChk} \dashv \|\hat{\Theta}\|, \Omega'$ ″

☞ $\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} \chi \leftrightarrow [\Omega, \Omega']^2\chi'$ ″

☞ $\hat{\Theta};\Gamma \vdash \mathrm{into}(v_0) \Leftarrow \{\nu : \mu F \mid \mathscr{M}(F)\} \mathbin{/} \chi' \dashv \Delta'$ By Alg⇐μ

- **Case**

$$\frac{}{\|\hat{\Theta}\|; \Gamma \mathrel{\widetilde{\vdash}} \{e\} \Leftarrow \downarrow[\Omega]N \;/\; (e \Leftarrow [\Omega]N)}\;\widetilde{\Leftarrow}\downarrow$$

☞	$\hat{\Theta}; \Gamma \vdash \{e\} \Leftarrow \downarrow N \;/\; (e \Leftarrow N) \dashv \cdot$	By Alg$\Leftarrow\downarrow$
	$\overline{\hat{\Theta}} \rhd \downarrow N$ type$[_]$	Given
	$\overline{\hat{\Theta}} \rhd N$ type$[_]$	By inversion
	$\|\overline{\hat{\Theta}}\| \vdash [\Omega]N$ type$[_]$	By Lemma F.12
		and Ω present
	$\|\overline{\hat{\Theta}}\| \vdash [\Omega]N \equiv^{-} [\Omega]N$	Lemma G.11 (Tp./Meas. Equiv. Reflexive)
	$\|\hat{\Theta}\| \vdash [\Omega]N \equiv^{-} [\Omega]N$	Lemma C.41 (Ix.-Level Weakening)
	$\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} \cdot \leftrightarrow \cdot$	By $\widetilde{\lhd}\leftrightarrow$Empty
	$\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} (e \Leftarrow [\Omega]N) \leftrightarrow (e \Leftarrow [\Omega]N)$	By $\widetilde{\lhd}\leftrightarrow\Leftarrow-$
☞	$\|\hat{\Theta}\| \mathrel{\widetilde{\lhd}} (e \Leftarrow [\Omega]N) \leftrightarrow [\Omega]^2(e \Leftarrow N)$	By def. of $[-]-$ and e ground
	Let $\Delta' = \cdot$.	
	Let $\Omega' = \cdot$.	
☞	$\|\hat{\Theta}\|, \Delta' \vdash \underbrace{\mathsf{tt}}_{\lfloor e \Leftarrow [\Omega]N \rfloor}$ fixInst $\dashv \|\hat{\Theta}\|, \Omega'$	By a rule, equalities

(5)

$\Theta \xrightarrow{\mathsf{SMT}} \Theta$		By Lemma F.5 (Ext. Reflexive)
$\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow P \mathbin{/} \chi$		Given
$\|\Theta\|;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow [\Theta]P \mathbin{/} \chi$		By definitions
$\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$		Given
$\|\Theta\|;\Gamma \mathrel{\widetilde{\lhd}} [\Theta]\chi$		By definitions
$[\Theta]P = P$		By def.
$\overline{\Theta} \vdash P\ \mathsf{type}[_]$		Presupposed derivation
$\overline{\Theta} \rhd P\ \mathsf{type}[_]$		Decl. WF implies Algo. WF
$\Theta\ \mathsf{present}$		
	$\Theta;\Gamma \vdash v \Leftarrow P \mathbin{/} \chi' \dashv \Delta'$	By i.h.
	$\|\Theta\|, [\xi_{\chi'}]\Delta';\Gamma \vdash [\Theta]\chi'\ \mathsf{fixInstChk} \dashv \|\Theta\|, \Omega'$	$''$
	$\Theta, [\xi_{\chi'}]\Delta';\Gamma \vdash \chi'\ \mathsf{fixInstChk} \dashv \Theta, \Omega'$	By definitions
☞	$\Theta;\Gamma \rhd v \Leftarrow P$	By Alg⇐Val

(6) • **Case**

$$\frac{\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow P \mathbin{/} \chi \qquad \Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} \mathtt{return}\, v \Leftarrow \uparrow P}\ \widetilde{\Leftarrow}\uparrow$$

$\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow P \mid \chi$	Subderivation
$[\Theta]P = P$	By def. of $[-]-$
$\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow [\Theta]P \mid \chi$	By equality
$\Theta \xrightarrow{\mathsf{SMT}} \Theta$	By $\xrightarrow{\sim}$Empty
$\Theta \vdash \uparrow P\ \mathsf{type}[_]$	Presupposed derivation
$\Theta \vdash P\ \mathsf{type}[_]$	By inversion
$\Theta \rhd P\ \mathsf{type}[_]$	Straightforward
$\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$	Subderivation
$\Theta;\Gamma \rhd v \Leftarrow P$	By i.h.
$\Theta;\Gamma \rhd \mathtt{return}\, v \Leftarrow \uparrow P$	By Alg$\Leftarrow\uparrow$

- **Case**

$$\frac{\Theta;\Gamma \mathrel{\widetilde{\vdash}} g \Rightarrow \uparrow(\exists^{\mathsf{d}}\Xi.R \wedge \overrightarrow{\psi}) \qquad \Theta,{}^{\mathsf{d}}\Xi,\overrightarrow{\psi};\Gamma,x:R \mathrel{\widetilde{\vdash}} e_0 \Leftarrow L}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} \mathtt{let}\, x = g;\, e_0 \Leftarrow L}\ \widetilde{\Leftarrow}\mathsf{let}$$

$\Theta;\Gamma \mathrel{\widetilde{\vdash}} g \Rightarrow \uparrow(\exists^{\mathsf{d}}\Xi.\, R \wedge \overrightarrow{\psi})$	Subderivation
$\Theta;\Gamma \rhd g \Rightarrow \uparrow P'$	By i.h.
$\Theta \vdash \exists^{\mathsf{d}}\Xi.\, R \wedge \overrightarrow{\psi} \equiv^{+} P'$	$''$
$P' = \exists^{\mathsf{d}}\Xi.\, R' \wedge \overrightarrow{\psi'}$	By inversion
$\Theta, {}^{\mathsf{d}}\Xi \vdash R \equiv^{+} R'$	$''$
$\Theta, {}^{\mathsf{d}}\Xi \vdash \overrightarrow{\psi} \equiv \overrightarrow{\psi'} : \mathbb{B}$	$''$
$\Theta, {}^{\mathsf{d}}\Xi, \overrightarrow{\psi} \vdash R \equiv^{+} R'$	By Lemma C.41 (Ix.-Level Weakening)
$\Theta, {}^{\mathsf{d}}\Xi, \overrightarrow{\psi}; \Gamma, x : R \mathrel{\widetilde{\vdash}} e_0 \Leftarrow L$	Subderivation
$\Theta \vdash \Gamma \equiv^{+} \Gamma$	Lemma G.11 (repeated)
$\Theta, {}^{\mathsf{d}}\Xi, \overrightarrow{\psi} \vdash \Gamma \equiv^{+} \Gamma$	By Lemma C.41 (Ix.-Level Weakening) (repeated)
$\Theta, {}^{\mathsf{d}}\Xi, \overrightarrow{\psi} \vdash \Gamma, x : R \equiv^{+} \Gamma, x : R'$	Add entry
$\Theta \vdash L\ \mathsf{type}[_]$	Presupposed derivation
$\Theta, {}^{\mathsf{d}}\Xi, \overrightarrow{\psi} \vdash L \equiv^{-} L$	Lemma G.11 (Tp./Meas. Equiv. Reflexive) and Lemma C.41 (Ix.-Level Weakening)

$\Theta, {}^{\mathsf{d}}\Xi, \overrightarrow{\psi}; \Gamma, x : R' \mathrel{\widetilde{\vdash}} e_0 \Leftarrow L$	Lemma G.26 (Typing Respects Equiv)
$\Theta, {}^{\mathsf{d}}\Xi, \overrightarrow{\psi}; \Gamma, x : R' \vdash e_0 \Leftarrow L$	Lemma G.6 (Semidecl. Typing Sound)
$\Theta, {}^{\mathsf{d}}\Xi, \overrightarrow{\psi'}; \Gamma, x : R' \vdash e_0 \Leftarrow L$	By Lemma C.71 (Ctx. Equiv. Compat)
$\Theta, {}^{\mathsf{d}}\Xi, \overrightarrow{\psi'}; \Gamma, x : R' \mathrel{\widetilde{\vdash}} e_0 \Leftarrow L$	Lemma G.7 (Semidecl. Typing Complete)
$\Theta, {}^{\mathsf{d}}\Xi, \overrightarrow{\psi'}; \Gamma, x : R' \rhd e_0 \Leftarrow L$	By i.h.

☞ $\Theta;\Gamma \rhd \mathsf{let}\ x = g;\ e_0 \Leftarrow L$ By Alg⇐let

- **Case**

$$\frac{\Theta \mathrel{\widetilde{\vdash}} \forall a \overset{\mathsf{d}}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.M <:^{-} L \,/\, W \qquad \Theta \mathrel{\widetilde{\models}} W \qquad \Theta, a \div \mathbb{N}; \Gamma, x : \downarrow \forall a' \overset{\mathsf{d}}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.a' < a \supset [a'/a]M \mathrel{\widetilde{\vdash}} e_0 \Leftarrow \forall^{\mathsf{d}}\Xi.M}{\Theta;\Gamma \mathrel{\widetilde{\vdash}} \mathsf{rec}\, x : (\forall a \overset{\mathsf{d}}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.M).\ e_0 \Leftarrow L}\ \widetilde{}\Leftarrow\mathsf{rec}$$

$\Theta \mathrel{\widetilde{\vdash}} \forall a \overset{\mathsf{d}}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.M <:^{-} L \,/\, W$ Subderivation

$\Theta \mathrel{\widetilde{\models}} W$ Subderivation

$\Theta \vdash \forall a \overset{\mathsf{d}}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.M <:^{-} L$ Lemma G.38

$\Theta, a \div \mathbb{N}; \Gamma, x : \downarrow \forall a' \overset{\mathsf{d}}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.a' < a \supset [a'/a]M \mathrel{\widetilde{\vdash}} e_0 \Leftarrow \forall^{\mathsf{d}}\Xi.M$ Subderivation

$\Theta, a \div \mathbb{N}; \Gamma, x : \downarrow \forall a' \overset{\mathsf{d}}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.a' < a \supset [a'/a]M \rhd e_0 \Leftarrow \forall^{\mathsf{d}}\Xi.M$ By i.h.

☞ $\Theta;\Gamma \rhd \mathsf{rec}\, x : (\forall a \overset{\mathsf{d}}{\div} \mathbb{N}, {}^{\mathsf{d}}\Xi.M).\ e_0 \Leftarrow L$ By Alg⇐rec

- The remaining cases for expression typing are straightforward.

(7) Similar to part (6).

(8) Similar to part (4).

- **Case** $\widetilde{}$Spine$\supset$: Similar to $\widetilde{}\Leftarrow\wedge$ case of part (4).
- **Case** $\widetilde{}$SpineApp: Similar to $\widetilde{}\Leftarrow\times$ case of part (4).
- **Case** $\widetilde{}$SpineNil: Straightforward. Use Lemma G.11 (Tp./Meas. Equiv. Reflexive) and $\widetilde{\lhd}\leftrightarrow$Empty and Lemma G.14 (Prob. Equiv. Reflexive).
- **Case** $\widetilde{}$Spine$\forall$: Impossible because $[\Omega]M \neq \forall$.

(9) Similar to a combination of part (5) and the $\widetilde{\Leftarrow}\exists$ case of part (4), case analyzing whether $N = \forall$. □

Theorem G.1 (Alg. Typing Complete)**.**

(1) If $\Theta;\Gamma \vdash h \Rightarrow P$ *then* $\Theta;\Gamma \rhd h \Rightarrow P$.

(2) If $\Theta;\Gamma \vdash g \Rightarrow \uparrow P$ *then* $\Theta;\Gamma \rhd g \Rightarrow \uparrow P'$ *and* $\Theta \vdash P \equiv^+ P'$ *for some* P'.

(3) If $\Theta;\Gamma \vdash v \Leftarrow P$ *then* $\Theta;\Gamma \rhd v \Leftarrow P$.

(4) If $\Theta;\Gamma \vdash e \Leftarrow N$ *then* $\Theta;\Gamma \rhd e \Leftarrow N$.

(5) If $\Theta;\Gamma;[P] \vdash \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$ *then* $\Theta;\Gamma;[P] \rhd \{r_i \Rightarrow e_i\}_{i\in I} \Leftarrow N$.

(6) If $\hat{\Theta};\Gamma;[N] \vdash s \Rightarrow \uparrow P$ *then* $\Theta;\Gamma;[N] \rhd s \Rightarrow \uparrow P'$ *and* $\Theta \vdash P \equiv^+ P'$ *for some* P'.

Proof.

(1)

	$\Theta;\Gamma \vdash h \Rightarrow P$	Given
	$\Theta;\Gamma \widetilde{\vdash} h \Rightarrow P$	By Lemma G.7 (Semidecl. Typing Complete)
	$\Theta;\Gamma \rhd h \Rightarrow P$	By Lemma G.39 (Aux. Alg. Typing Complete)

(2)

	$\Theta;\Gamma \vdash g \Rightarrow \uparrow P$	Given
	$\Theta;\Gamma \widetilde{\vdash} g \Rightarrow \uparrow P$	By Lemma G.7 (Semidecl. Typing Complete)
☞	$\Theta;\Gamma \rhd g \Rightarrow \uparrow P'$	By Lemma G.39 (Aux. Alg. Typing Complete)
☞	$\Theta \vdash P \equiv^+ P'$	〃

(3)

	$\Theta;\Gamma \vdash v \Leftarrow P$	Given
	$\Theta;\Gamma \mathrel{\widetilde{\vdash}} v \Leftarrow P \mathbin{/} \chi$	By Lemma G.7 (Semidecl. Typing Complete)
	$\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$	$''$
☞	$\Theta;\Gamma \rhd v \Leftarrow P$	By Lemma G.39 (Aux. Alg. Typing Complete)

(4) Similar to part (1).

(5) Similar to part (1).

(6)

	$\Theta;\Gamma;[N] \vdash s \Rightarrow \uparrow P$	Given
	$\Theta;\Gamma;[N] \mathrel{\widetilde{\vdash}} s \Rightarrow \uparrow P \mathbin{/} \chi$	By Lemma G.7 (Semidecl. Typing Complete)
	$\Theta;\Gamma \mathrel{\widetilde{\lhd}} \chi$	$''$
☞	$\Theta;\Gamma;[N] \rhd s \Rightarrow \uparrow P'$	By Lemma G.39 (Aux. Alg. Typing Complete)
☞	$\Theta \vdash P \equiv^{+} P'$	$''$

□