Abstract

Syntactic Finitism in the Metatheory of Programming Languages

Jeffrey Sarnat

2010

One of the central goals of programming-language research is to develop mathematically sound formal methods for precisely specifying and reasoning about the behavior of programs. However, just as software developers sometimes make mistakes when programming, researchers sometimes make mistakes when proving that a formal method is mathematically sound. As the field of programming-language research has grown, these proofs have become larger and more complex, and thus harder to verify on paper. This phenomenon has motivated a great deal of research into the development of logical systems that provide an automated means to apply—and verify the application of—trusted reasoning principles to concrete proofs.

The boundary between trusted and untrusted reasoning principles is inherently blurry, and different researchers draw the line in different places. However, just as certain principles are widely recognized to allow the proofs of contradictory statements, others are so uncontroversially ubiquitous in practice that they can be considered beyond reproach. We posit the following questions: (1) what are these principles and (2) how much can we do with them?

Although neither has an uncontroversial answer, in this dissertation we propose an answer to the former by describing a philosophical viewpoint we refer to as *syntactic finitism*, in which the principles of case analysis and structural induction on abstract syntax are viewed as being *a priori* justified. We explore the latter question using some of the ideas and results from proof theory; along the way, we provide a syntactically finitary account of proofs by logical relations, and investigate the consequences of replacing structural induction with well-founded induction on the lexicographic path ordering from term rewriting theory. Finally, we argue that syntactically finitary proofs can be formalized in the *proofs as logic programs* paradigm popularized by the proof assistant Twelf; we prove the soundness of a modular termination-analysis that is central to the validity of this interpretation.

Syntactic Finitism in the Metatheory of Programming Languages

A Dissertation
Presented to the Faculty of the Graduate School
of
Yale University
in Candidacy for the Degree of
Doctor of Philosophy

by
Jeffrey Sarnat

Dissertation Director: Carsten Schürmann

May 2010

Contents

Acknowledgements

First and foremost, I owe an incredible debt of gratitude to my advisor, Carsten Schürmann, whose support for and faith in my ideas has been matched only by his tolerance for idiosyncrasies that would have broken lesser mentors. He has always treated me as a more of a colleague than a student, but has also played the unenviable role of concerned parent to my unruly teenager and, the (hopefully) less burdensome role of friend.

The starting point for the work in this dissertation was an investigation into the nature of proofs by logical relations. During my time as an undergraduate, I was fortunate enough to be introduced to this proof technique in two different classes, one taught by Karl Crary, the other by Richard Statman. Weak normalization for the simply typed λ-calculus was proved as an example in both classes, but one relied on a Kripke-style definition, complete with quantification over typing contexts and various substitution lemmas, whereas the other relied on much less technical machinery. One day, my friend and classmate Bor-Yuh Evan Chang suggested that the two of us spend some time trying to figure out the why one of these proofs seemed so much more complex than the other. We came up with the following explanation: in the simpler proof, every variable's type is recorded as part of its name, whereas in the more complex proof, the type of a variable must be looked up in a typing context. This simple observation was one of many that led to the development of the concept of *structural logical relation*, which plays an important role in my work.

My graduate career has been unusual in that it spanned across two continents and two universities, first Yale, then the IT University of Copenhagen. I would like to thank Linda Dobb, Drew McDermott and Vladimir Rokhlin for their essential help in dealing with the nontrivial administrative issues associated with completing the last years of my dissertation research in a foreign country.

Over the years, I have benefitted immensely from the knowledge and expertise of numerous colleagues on both sides of the Atlantic. I am especially grateful to Valery Trifonov for our many enlightening conversations early in my graduate career, to Adam Poswolsky, whose friendship and technical knowledge provided continuity in our trip across the Atlantic, and to all of the PLS group at ITU, who always treated me as though I were one of their own. In particular, I would like to thank Anders Schack-Nielsen, whose expertise was of great help when I was trying to find an elegant specification for the semantics of logic programming, Søren Debois, for his helpful comments on an earlier draft of this document, and Geoffrey Washburn for not only providing helpful feedback on my research, but creating a font that allows me to typeset the symbol ⏚. Beyond Yale and ITU, I benefitted immensely from conversations with Michael Rathjen and Anton Setzer, who were kind enough to answer some of my early questions about ordinal analysis, and from an email exchange with Georg Moser, in which some of my later questions about large ordinals were answered.

Finally, I am grateful to all of my friends, and especially my family, for their continued loving support in all my endeavors, even when they keep me far from home.

Chapter 1

Introduction

At its heart, computer science is not as much the study of computers as it is the study of problems that computers can be used to solve. But what does it mean for a problem to be solved with a computer? Simply writing a program is not enough, regardless of the computational paradigm; the program must also be *correct*, and establishing correctness requires the ability to reason, be it formally or informally, about the behavior of software.

We live in a world where it is increasingly common to rely on the correctness of computer programs, and where the consequences of incorrect programs can range from inconvenience to the loss of life. At the same time, the programs that we rely on are becoming increasingly complex, and thus harder to reason about informally. This phenomenon has motivated a great deal of programming-language research into the development of mathematically sound formal methods for precisely specifying and reasoning about the behavior of programs (e.g. operational semantics, denotational semantics, type systems and Hoare logic), along with practical tools that give programmers a (semi-) automated means to apply—and verify the application of—these techniques to concrete programs (e.g. type checkers, abstract interpretors and model checkers).

However, just as software developers sometimes make mistakes when programming, researchers sometimes make mistakes when proving that a formal method is mathematically sound. That is, simply writing a soundness proof is not enough; the proof must also be *correct*, and establishing the correctness of a proof requires the ability to reason, be it formally or informally, about the applicability of trusted reasoning principles. As the field of programming-languages research has grown, the proofs have become larger and more complex, and thus harder to verify informally. This phenomenon has motivated a great deal of research into the development of logical systems (e.g. Coq, Isabelle, Agda and Twelf) that allow researchers a (semi-) automated means to apply—and verify the application of—trusted reasoning principles to concrete proofs. But *exactly* which reasoning principles can be trusted?

This is a deeply philosophical question, whose answer varies from person to person; we do not attempt to answer it here. However, just as certain reasoning principles (e.g. the unrestricted comprehension axiom in naive set theory and *type*:*type* in type theory) are widely recognized to allow the proofs of contradictory statements (e.g. Russel's paradox, Girard's paradox), other reasoning principles are so uncontroversially ubiquitous in practice that they can be considered beyond reproach. We posit the following questions: (1) what are these principles and (2) how much can we do with them? Although neither has an uncontroversial answer, we propose an answer to the former in the remainder of this section, and explore the latter in the remainder of this dissertation.

1.1 Abstract Syntax and Judgments

The dear God made the whole numbers, everything else is the work of man.

Leopold Kronecker (attributed in [Web93])
quote translated from German by Dirk Schlimm

The tradition called "syntactic"... would without doubt have disappeared in one or two more decades, for want of any issue or methodology. The disaster was averted because of computer science – that great manipulator of syntax – which posed it some very important theoretical problems.

Jean-Yves Girard [GLT89]

Syntax is as fundamental to the study of programming languages as the whole numbers are to mathematics. Thus, we regard the definition of syntactic categories using BNF-style notation as being *a priori* justified. For example, the natural numbers (or, if we were to interpret z as a representation of the number 1, the whole numbers) can be defined in this manner as follows.

$$\textbf{Natural Numbers} \quad n, m \quad ::= \quad z \mid s\,n$$

The left-hand side of this definition establishes "n" and "m" as standing for arbitrary natural numbers; right-hand side allow us to "build up" natural numbers using the term constructors "z" and "s".

In order to reason about syntax, we need some means to express the properties that syntactic terms may or may not possess. Although there are several ways of doing this, perhaps the most basic and widespread is to define judgments using inference rules, a practice that we consider to be *a priori* justified. For example, we may define the assertion that a particular natural number is even, along with the assertion that a particular natural number is odd, as judgments defined by the following inference rules.

$$\frac{}{\mathit{even}(z)}\,\scriptstyle{evenz} \qquad \frac{\mathit{even}(n)}{\mathit{odd}(s\,n)}\,\scriptstyle{odds} \qquad \frac{\mathit{odd}(n)}{\mathit{even}(s\,n)}\,\scriptstyle{evens}$$

Judgments can also be used to represent computations. For example, we may define the assertion that the natural numbers n_0 and n_1 sum to the number m by defining the judgment $\mathit{add}(n_0;\ n_1;\ m)$ using the following inference rules.

$$\frac{}{\mathit{add}(z;\ n_1;\ n_1)}\,\scriptstyle{addz} \qquad \frac{\mathit{add}(n_0;\ n_1;\ m)}{\mathit{add}(s\,n_0;\ n_1;\ s\,m)}\,\scriptstyle{adds}$$

The relationship between the Ackermann function's inputs and outputs can be represented similarly.

$$\frac{}{\mathit{ack}(z;\ n_1;\ s\,n_1)}\,\scriptstyle{ackz} \qquad \frac{\mathit{ack}(n_0;\ s\,z;\ m)}{\mathit{ack}(s\,n;\ z;\ m)}\,\scriptstyle{acksz} \qquad \frac{\mathit{ack}(s\,n_0;\ n_1;\ m) \quad \mathit{ack}(n_0;\ m;\ m')}{\mathit{ack}(s\,n_0;\ s\,n_1;\ m')}\,\scriptstyle{ackss}$$

In general, inference rules can be used to "build up" derivations of judgments in much the same way that the clauses of a BNF-style definition can be used to build up syntactic terms. We refer to arbitrary derivations of a judgment such as $\mathit{ack}(n_0;\ n_1;\ m)$ using the notation $\mathcal{D} : \mathit{ack}(n_0;\ n_1;\ m)$ (or just $\mathcal{D}$ whenever the judgment is clear from the context), and say that a judgment is *inhabited* to mean that such a derivation exists. We write concrete derivations of judgments as trees; for example, a derivation of the judgment $\mathit{ack}(s\,z;\ s\,z;\ s\,(s\,(s\,z)))$ would be written as follows.

$$\frac{\dfrac{\dfrac{}{\mathit{ack}(z;\ s\,z;\ s\,(s\,z))}\,\scriptstyle{ackz}}{\mathit{ack}(s\,z;\ z;\ s\,(s\,z))}\,\scriptstyle{acksz} \qquad \dfrac{}{\mathit{ack}(z;\ s\,(s\,z);\ s\,(s\,(s\,z)))}\,\scriptstyle{ackz}}{\mathit{ack}(s\,z;\ s\,z;\ s\,(s\,(s\,z)))}\,\scriptstyle{ackss}$$

In general, judgments can be thought of as making logical assertions, where a derivation is a finite witness to an assertion's validity. That is, $\mathit{ack}(s\,z;\ s\,z;\ s\,(s\,(s\,z)))$

can be thought of as being “true” or “provable” (*evident* in the terminology of Martin-Löf [ML85]) by virtue of the fact that it is inhabited by the derivation written above. Judgments that make logical assertions of the form “the program e evaluates to a value v” are widely and uncontroversially used to specify the semantics of programming languages. Such judgmental specifications are referred to as *structural operational semantics* [Plo81].

Some judgments fail to define interesting logical assertions; this, however does not preclude their derivations from being interesting. One such judgment is defined below; its derivations are isomorphic to the natural numbers, as defined at the beginning of this section.

$$\frac{}{\mathit{nat}}\,{\scriptstyle z} \qquad \frac{\mathit{nat}}{\mathit{nat}}\,{\scriptstyle s}$$

In general, any syntactic category defined in BNF-style notation can be expressed as a judgment. Similarly, if we broaden our notion of BNF-style definition to allow syntactic categories that *depend* on one another, we can define syntactic categories whose terms are isomorphic to the derivations of arbitrary judgments. For example, the syntactic categories E^n and O^n, defined below, are isomorphic to derivations of the judgments $\mathit{even}(n)$ and $\mathit{odd}(n)$, respectively.

$$\begin{array}{lcl} E^n & ::= & \mathit{evenz}^z \mid (\mathit{evens}\ O^n)^{s\,n} \\ O^n & ::= & (\mathit{odds}\ E^n)^{s\,n} \end{array}$$

Although the use of dependencies in the definition of abstract syntax is not particularly widespread, such definitions are no less justifiable than judgments defined by inference rules. Thus, the distinction between syntactic categories and judgments is essentially a matter of notational preference. We refer to this observation as the *judgments-as-types principle*, for reasons that will be made more clear in Section 5.1.

1.2 Syntactic Finitism

Syntactic categories and judgments are both inherently *syntactically finitary* concepts in the sense that, although there may be unboundedly many natural numbers, each individual natural number is constructed from finitely many uses of the constructors z and s, and nothing else. Thus, we are *a priori* justified not only in building up concrete terms and derivations via the application of constructors and inference rules, but in taking apart abstract terms and derivations via *case analysis* and transforming them into potentially different types of objects via *structural induction* (i.e. well-founded induction on the subterm ordering). We show an example of such reasoning, which includes a non-trivial case analysis, in the theorem below.

Theorem 1.2.1 *If* $\mathcal{D} : \mathit{add}(n_0;\, n_1;\, m)$ *and* $\mathcal{E} : (\mathit{even}(n_0))$ *and* $\mathit{even}(n_1)$ *then* $\mathit{even}(m)$

Proof: By structural induction on $\mathcal{E}$ (n_0 and $\mathcal{D}$ are equally suitable candidates). Note that the statement of the theorem is shorthand for the following, more verbose statement: "For every n_0, n_1, m, for every $\mathcal{D} : (\mathit{add}(n_0;\, n_1;\, m))$ and for every $\mathcal{E} : (\mathit{even}(n_0))$ and for every $\mathcal{F} : \mathit{even}(n_1)$ there exists $\mathcal{G} : (\mathit{even}(m))$.

Case:

$$\mathcal{E} = \frac{}{\mathit{even}(z)}\,\mathit{evenz}$$

$$\mathcal{D} = \frac{}{\mathit{add}(z;\, n_1;\, n_1)}\,\mathit{addz}$$

$\mathit{even}(n_1)$ given

Case:

$$\mathcal{E} \quad = \quad \cfrac{\cfrac{\begin{array}{c}\mathcal{E}_0\\ even(n_0)\end{array}}{odd(s\,n_0)}\;odds}{even(s\,(s\,n_0))}\;evens$$

$$\mathcal{D} \quad = \quad \cfrac{\cfrac{\begin{array}{c}\mathcal{D}_0\\ add(n_0;\; n_1;\; m)\end{array}}{add(s\,n_0;\; n_1;\; s\,m)}\;adds}{add(s\,(s\,n_0);\; n_1;\; s\,(s\,m))}\;adds$$

$even(n_1)$	given
$even(m)$	by IH on $\mathcal{E}_0$
$odd(s\,m)$	by rule *odds*
$even(s\,(s\,m))$	by rule *evens*

□

Informally, Theorem 1.2.1 tells us that the given summand of two even numbers is even. However, thus far we have not shown that such a summand must always exist. We can do so by proving the statement "for every n_0, n_1, there exists an m such that $add(n_0;\; n_1;\; m)$," which, in this case follows by a straightforward structural induction on the structure of n_0. Thus, we can think of *add* as describing a (total) function[1] from its first two arguments to its third.

The analogous totality statement for *ack*—"for every n_0, n_1, there exists an m such that $ack(n_0;\; n_1;\; m)$"—can also be proven by a straightforward induction, not on

[1]Technically this only shows that *add*'s third argument is a potentially *nondeterministic* total function of its first two arguments, since we have not yet shown that it is unique (although in this case, doing so would be straightforward). In this document, we view all functions as being potentially nondeterministic until proven otherwise, and only explicitly prove determinacy when doing so is germane to the discussion at hand.

the structure of n_0 or on the structure of n_1, but on the *lexicographic ordering* of n_0 and n_1. In general, we consider *well-founded induction* to be *a priori* justified on any ordering that we consider intuitively well-founded, such as the lexicographic ordering of two well-founded orderings. For now, we restrict our attention to the subterm ordering on terms/derivations, and the lexicographic and simultaneous orderings on tuples of terms/derivations; we will consider both the consequences of this restriction in Chapter 3, and the impact of assuming much stronger ordering is well-founded in Chapter 4.

Recall that lexicographic orderings generalize the notion of the alphabetical ordering used to order words in dictionaries, and precisely captures the notion of "tie breaker" sometimes used to rank sports teams (e.g. teams are ranked first by their win-loss record, teams with the same record are ranked by their point differential, etc.). More precisely, one tuple is smaller than another iff the first n elements of each are equal in size, and the $(n + 1)$th element of the first is strictly smaller than the $(n + 1)$th element of the second. The simultaneous ordering can be seen as a special case of the lexicographic ordering: one tuple simultaneously smaller than another iff every element of the first is smaller-than or equal-in-size to the corresponding element in the second tuple, and moreover at least one of these inequalities is strict. The simultaneous ordering can be generalized to be commutative: one tuple is smaller than another iff the elements of the first can be permuted such that the resulting tuple is smaller than the second according to the ordinary simultaneous ordering. Strictly speaking this commutative generalization is unnecessary: any theorem whose inductive proof uses the natural sum ordering on an n-tuple of derivations can, in principle, be transformed into $n!$ mutually inductive theorems whose proofs use the conventional simultaneous ordering.

In terms of ordinal arithmetic, the lexicographic ordering defines ordinal multiplication; the significance of this observation will be expounded upon in Chapter 3.

1.3 Hypothetical Judgments and Bound Variables

Consider the syntactic category τ, defined below.

$$\sigma, \tau \quad ::= \quad o \mid \sigma \Rightarrow \tau$$

For now we wish to think of τ as defining propositions, where o is an empty proposition, and $\sigma \Rightarrow \tau$ is an implication. We have seen that judgments can be used to define logical assertions such as "the natural number n is even"; they can also be used to define assertions of a more transparently logical nature, such as "the formula τ is provable." We give an example of a *natural deduction* style proof system by the *hypothetical judgment* $\vdash \tau\ nd$ with the following inference rules.

$$\dfrac{\begin{array}{c}\dfrac{}{\vdash \sigma\ nd}\,x^\sigma \\ \vdots \\ \vdash \tau\ nd\end{array}}{\vdash \sigma \Rightarrow \tau\ nd}\,lam_{x^\sigma} \qquad \dfrac{\vdash \sigma \Rightarrow \tau\ nd \quad \vdash \sigma\ nd}{\vdash \tau\ nd}\,app$$

Informally, the rule lam_{x^σ} can be applied to a derivation containing any number of free uses of the inference rule x^σ, where we regard the exact choice of the name x^σ as being irrelevant to the identity of the resulting derivation. In other words, we can view the rule lam_{x^σ} as *binding* the *inference-rule variable* x^σ. In general, we view variables as holes that can be filled with arbitrary derivations, thus justifying the following principle of substitution: given a derivation $\mathcal{D} : (\vdash \tau\ nd)$ that contains some number of free occurrences of the inference-rule variable x^σ, and a derivation $\mathcal{E} : (\vdash \sigma\ nd)$, there exists a derivation of $\vdash \tau\ nd$, written $\mathcal{D}[\mathcal{E}/x^\sigma]$, in which all occurrences of x^σ in $\mathcal{D}$ have been replaced by $\mathcal{E}$. Given the machinery described thus far, we either view the (capture-avoiding) substitution principle (and thus the notation $\mathcal{D}[\mathcal{E}/x^\sigma]$) as being *a priori* justified, or as a theorem that must be proven by induction (where $\mathcal{D}[\mathcal{E}/x^\sigma] = \mathcal{F}$ would be formulated judgmentally). In this document, we adopt the former view for the sake of convenience, although our results do not depend on this in any essential way.

We can also define natural deduction style proofs using BNF-style notation.

$$e^\tau \quad ::= \quad x^\tau \mid (lam\ x^\sigma .\, e^\tau)^{\sigma \Rightarrow \tau} \mid (app\ e_1^{\sigma\Rightarrow\tau}\ e_2^\sigma)^\tau$$

If we think of τ as defining types rather than propositions, e^τ clearly defines the well-typed terms of the simply typed λ-calculus. For example, we can define the notion of single- and multi-step reduction by β-contraction and η-expansion using the following judgments (where we omit type superscripts where they are easily inferred or irrelevant).

$$\frac{}{app\ (lam\ x^\tau .\, e_1^\sigma)\ e_2^\tau \longrightarrow e_1^\sigma[e_2^\tau/x^\tau]}\ \textit{red-beta} \qquad \frac{}{e^{\sigma\Rightarrow\tau} \longrightarrow lam\ x^\tau . app\ e^{\sigma\Rightarrow\tau}\ x^\sigma}\ \textit{exp-eta}$$

$$\frac{e_1^{\sigma\Rightarrow\tau} \longrightarrow e_1'^{\sigma\Rightarrow\tau}}{app\ e_1^{\sigma\Rightarrow\tau}\ e_2^\sigma \longrightarrow app\ e_1'^{\sigma\Rightarrow\tau}\ e_2^\sigma}\ \textit{cong-app1} \qquad \frac{e_2^\sigma \longrightarrow e_2'^\sigma}{app\ e_1^{\sigma\Rightarrow\tau}\ e_2^\sigma \longrightarrow app\ e_1^{\sigma\Rightarrow\tau}\ e_2'^\sigma}\ \textit{cong-app1}$$

$$\frac{e^\sigma \longrightarrow e'^\sigma}{lam\ x^\tau .\, e^\sigma \longrightarrow lam\ x^\tau .\, e'^\sigma}\ \textit{cong-lam}_{x^\sigma}$$

$$\frac{}{e^\tau \longrightarrow^* e^\tau}\ \textit{reds-refl} \qquad \frac{e_1^\tau \longrightarrow e_2^\tau \quad e_2^\tau \longrightarrow^* e_3^\tau}{e_1^\tau \longrightarrow^* e_3^\tau}\ \textit{reds-trans}$$

Alternatively, $\longrightarrow$ and $\longrightarrow^*$ can be thought of as defining reductions on derivations of $\vdash \tau\ nd$. In other words, the judgments-as-types principle can be viewed as a generalization of the *proofs-as-programs* principle of the Curry-Howard isomorphism. We view the following theorem as being valid under either interpretation.

Theorem 1.3.1 (Congruence of Multistep Reduction)

1. *If* $e^{\tau} \longrightarrow^{*} e'^{\tau}$ *then* $lam\ x^{\sigma}.e \longrightarrow^{*} lam\ x^{\sigma}.e'^{\tau}$

2. *If* $e_1^{\sigma\Rightarrow\tau} \longrightarrow^{*} e_1'^{\sigma\Rightarrow\tau}$ *and* $e_2^{\sigma} \longrightarrow^{*} e_2'^{\sigma}$ *then* $(app\ e_1^{\sigma\Rightarrow\tau}\ e_2^{\sigma}) \longrightarrow^{*} (app\ e_1^{\sigma\Rightarrow\tau}\ e_2'^{\sigma})$

Proof: 1 is by straightforward induction on the structure of the given derivation; 2 is by straightforward simultaneous induction on the structure of the given derivations. □

At times, we will find it convenient to reason about whether a variable occurs freely within a term or derivation. In general, we write the judgmental specification of such a judgment along the lines of $x^{\tau} \in FV(e^{\tau})$, although in practice we do not specify the inference for such judgments for the sake of brevity.

1.4 The Role of Consistency

In some ways, our goals are similar in spirit to the goals of *Hilbert's program.* Originally proposed by the mathematician David Hilbert in the early 20th century, Hilbert's program aimed to provide an axiomatic formalization of all mathematical reasoning and to justify the consistency of this axiomatization using only *finitary* methods similar in spirit to the notion of syntactically finitary methods outlined above. Informally, a logic is consistent whenever its proof theory is somehow nontrivial in a meaningful way. Because we are interested in formalizing informal concepts, it seems prudent to consider how the notion of consistency might be made precise.

Perhaps the most common way to demonstrate that a logic is consistent is to show that it is "free from contradiction"; i.e. , that there can be no proof of a formula such as $F \wedge \neg F$ or $z = s\,z$. This definition is somewhat unsatisfying, because it relies on specific logical connectives that may or may not exist within the formalism under consideration. However, this notion can be generalized somewhat: a logic is consistent if, and only if, there is at least one formula that cannot be proved. This notion of *absolute consistency* usually follows as a corollary from a normalization theorem of some kind, such as cut elimination: first it is shown that all proofs can be normalized, then it is shown by case analysis that there are no normal proofs of a particular formula. Most of the time, absolute consistency captures what we mean by consistency. Unfortunately, this is not always the case.

Consider the (propositional) logic whose only atomic formula is true, and whose only logical connective is implication (under the Curry-Howard interpretation, this would be the simply-typed λ-calculus with a constant at base type). Every formula in this logic is provable, and yet the proofs themselves are not without meaning; several conservative extensions of this logic are consistent in the absolute sense, and moreover, the proof terms of this logic can be normalized. Unfortunately, neither of these observations brings us any closer to a uniform definition of consistency.

Consider an inconsistent logic, such as naive set theory, whose syntactic category of formulas we denote (in this paragraph only) using F. We can define the formulas of a new logic by a BNF-style equation of the form $G ::= F \mid \perp$ (where $\perp$ is not a formula in F), whose proof rules are those of naive set theory. Clearly, the newly resulting logic is an "absolutely consistent," conservative extension of naive set theory. And yet we consider neither naive set theory, nor this trivial conservative extension, to be consistent in any useful sense.

Defining a logic to be consistent whenever its proof theory admits some sort of normalization theorem is similarly problematic. Consider, for example, a consistent first-order theory, such as Heyting Arithmetic or the algebraic theory of groups, whose syntactic category of formulas we denote (in this paragraph only) using F. If we were to add a "normal" proof rule of the form

$$\frac{}{\vdash F}\,\textit{foo}$$

then the resulting logic would clearly be inconsistent in any reasonable sense of the word. And yet, every proof in the resulting logic can be “reduced” to a “normal” proof; namely, the proof that consists entirely of an application of the rule *foo*.

In general, we argue that any attempt to precisely define logical consistency will be fraught with exceptions. Thus, we eschew a precise definition in favor of an informal test, originally employed by Supreme Court Justice Potter Stewart to help clarify a somewhat different concept: we may not be able to precisely define consistency, but we know it when we see it.

Although Hilbert’s program successfully spawned a great deal of pioneering research into the foundations of mathematics, its goals were eventually shown to be unobtainable: in 1931, Kurt Gödel published his now famous *incompleteness theorems*, the second of which shows that no consistent logical formalism that is capable of expressing certain very basic arithmetical truths can be used to prove its own consistency, let alone the consistency of more powerful logic. Although Gödel’s original result was phrased in terms of absolute consistency, we claim that it applies to any reasonable interpretation of the word.

It follows that any consistency proof must rely, either implicitly or explicitly, on assumptions that are at least as strong as the statement being proven. Although this means that it is not possible to prove consistency of any interesting logic outright, consistency proofs can be usefully thought of as reducing one set of possibly questionable assumptions to another set of more intuitively trustworthy assumptions.

1.5 Outline and Contributions

We have argued that the syntactically finitary reasoning principles outlined thus far—that is, the ability to form terms and derivations via the application of inference rules and to transform them into different types of objects via case analysis and induction—characterize a reasonable lower bound on the forms of reasoning used by programming-languages researchers. Moreover, we argue that syntactic finitism essentially captures the notion of proof formalized by the proof assistant Twelf [PS99], which has been successfully used to formalize an impressive array of programming languages metatheorems. Our goal is to better understand the foundational expressivity of these principles; that is, in which situations does syntactic finitism suffice, and in which situations is it absolutely necessary to employ more powerful forms of reasoning?

Through the lens of the Curry-Howard isomorphism, any typed λ-calculus can be viewed as a logic, and thus any metatheorem about one can be viewed as an equivalent metatheorem about the other. Consistency proofs can provide a sort of benchmark for the expressivity of proof-theoretic assumptions; therefore it makes sense for us to investigate syntactic finitism with regards to the kinds of metatheorems about typed lambda calculi that, via Curry-Howard, imply consistency. Although examples of such metatheorems abound, the most obvious are normalization theorems. One of the most widely used class of mathematical structures for proving normalization theorems, and consistency meta-theorems in general, is the class of *logical relations.*

Logical relations are usually defined as an infinite family of infinite sets, and thus would appear to be incompatible with syntactic finitism; however, in Chapter 2, we will give a finitary characterization of logical relations that we refer to as *structural logical relations.* Because structural logical relations are syntactically finitary, they can be implemented in Twelf (see `http://www.twelf.org/slr` for the source code), thus solving a long-standing open problem [Abe08].

It should be noted that the existence of structural logical relations does not mean that syntactic finitism can always be used to prove consistency outright; rather, structural logical relations are syntactically finitary reductions from one consistency theorem to another in much the same way that Karp reductions are polynomial-time reductions from the computational complexity of one class of decision problems to another. Thus, the proof theoretic assumptions that are usually implicitly taken for granted in a typical proof by logical relations are made explicit in proof by structural logical relations.

However, the question still remains as to when syntactic finitism can be used to prove consistency outright. Just as finitism is sometimes considered to be formalized by the first-order theory PA_1 (i.e. Peano arithmetic where the principle of induction is restricted to Σ_1-formulas), in Chapter 3, we will argue that syntactically finitary proofs can, in principle, be formalized in PA_2 (i.e. Peano arithmetic where the principle of induction is restricted to Σ_2-formulas, or, equivalently, PA_1 extended with a notion of transfinite induction up to ω^ω). Thus, using some of the ideas and results from proof theory, which will also be discussed in Chapter 3, syntactic finitism can be associated with the proof-theoretic ordinal ω^{ω^ω}. Because we associate the proof assistant Twelf with the concept of syntactic finitism, we are able to use this ordinal to give a direct bound on the expressivity of this proof-assistant, which is a novel contribution. It follows as a corollary that proofs such as the consistency of Peano Arithmetic and the weak normalization of Gödel's T cannot be formalized directly in Twelf, or any other formal system that accurately captures the notion of syntactic finitism using the subterm ordering.

In Chapter 4, we will investigate the impact of extending our notion of syntactic finitism from induction on orderings built up from subterm orderings to induction on those same orderings built up from the **much** stronger *lexicographic path ordering* of term rewriting fame. We will see that using the lexicographic path ordering in this context allows us to prove much more powerful theorems than would be possible in the context of term rewriting. We provide an example of such a theorem—a novel cut-elimination-like theorem that implies the weak normalization of Gödel's T—in Chapter 4. This proof has been formalized in a prototypical extension to Twelf based on the lexicographic path ordering; see `http://www.twelf.org/lpo`.

Finally, we are also interested in formalizing the notion of syntactic finitism using methods that are more in the spirit of syntactic finitism than PA_2. We regard LF [HHP87] as providing a uniform means to express judgments, and the *proofs-as-logic-programs* paradigm popularized by the proof assistant Twelf as a more syntactic formalization of syntactic finitism than PA_2. In Chapter 5, we will attempt to justify this paradigm as a valid interpretation of syntactic finitism, and present a novel syntactically finitary specification for the semantics of logic programming in which unification is treated as a black-box; we then prove, using syntactically finitary methods extended by a principle of well-founded induction, the soundness of a judgmentally-specified mode/termination-analysis that is central to the interpretation of logic programs as syntactically finitary proofs. The mode/termination checker is modular in the ordering on LF terms, and thus can serve as a suitable foundation for novel implementations of of the Twelf termination checker, such as the lexicographic path ordering. Although such analyses, along with their proofs of soundness, have been sketched in the past [RP96, Pie05], our full proof soundness of soundness for a fully-specified mode/termination checker for a fully-specified semantics of logic programming is a novel contribution. Moreover, if the mode/termination is instantiated with the subterm ordering, then the soundness proof can, in principle, be formalized in PA_2 extended with the principle of transfinite induction up-to $\omega^{\omega+1}$—or, equivalently, PA_3—which is optimal with regards to the proof-theoretic bounds described in Chapter 3.

In Chapter 6 we conclude by discussing related and future work.

Chapter 2

Structural Logical Relations

A proof by logical relations (also sometimes referred to as Tait's method) relies on an interpretation of types as relations between objects. On paper, these interpretations are usually formulated in set theory. Consider the simply typed λ-calculus, as defined in Section 1.3. Given some property of expressions that we wish to reason about (e.g. reducibility to a normal form), a unary logical relation $[\![\tau]\!]$ would be defined as the smallest set satisfying the following conditions.

$$\begin{array}{lll} e^o \in [\![o]\!] & \text{iff} & e^o \text{ has the desired property} \\ e^{\sigma \Rightarrow \tau} \in [\![\sigma \Rightarrow \tau]\!] & \text{iff} & \text{for all } e_2^\sigma, \text{ if } e_2 \in [\![\sigma]\!] \text{ then } \mathit{app}\ e^{\sigma \Rightarrow \tau}\ e_2^\sigma \in [\![\tau]\!] \end{array}$$

The hallmark characteristic of a logical relation is the definition at function types: functions are in the relation if they map related arguments to related results. A proof by logical relations typically proceeds in two stages: first, it is shown, usually by induction on τ, that $e^\tau \in [\![\tau]\!]$ implies e^τ has the desired property (we refer to this as the *Escape Lemma*); then it is shown, usually by induction on e^τ that, for every e of type τ, $e \in [\![\tau]\!]$ (this is often referred to as the *Fundamental Theorem* or *Basic Lemma*). The desired theorem is then a direct consequence of modus ponens.

Taking a step back, we notice that the set-theoretic characterization of logical relation relies on the following concepts: structural induction on τ and e^τ, the ability to express a desired property about λ-calculus terms, and connectives "for all" and "if ... then." Logical relations are typically formulated with the understanding that these concepts all live on the same level. The idea of *structural logical relations* [SS08] is to separate these concepts in such a way that we only need to employ syntactically finitary reasoning: induction and case analysis on types and terms takes place on the meta-level (as usual), whereas logical connectives and properties about λ-calculus terms will be represented by an *assertion logic*, whose formulas and proof theory can be described syntactically.

A paper proof of weak normalization for the simply typed λ-calculus using a conventional, set-theoretic notion of logical relation can be found in [Pfe92]. A proof using structural logical relations for the same property is described in [SS08]. We recount this proof and extended it to Gödel's T here.

The overall structure of a proof by structural logical relations is much like its more conventional set-theoretic counterpart: the Escape Lemma is proved by induction on types, and the Fundamental Theorem is proved by induction on terms (although, unlike conventionally defined logical relations, proving the Fundamental Theorem for a structural logical relation does not require defining or reasoning about simultaneous substitutions). However, structural logical relations require proving one theorem, which we refer to as *Extraction*, that demonstrates the consistency of the assertion logic; although conventional proofs using logical relations do not require an explicit proof of the Extraction Theorem, we argue that this is because the act of defining a conventional logical relation requires such a consistency theorem to be implicitly assumed.

2.1 Normalization of the Simply Typed λ-calculus

We are interested in proving weak normalization for the notion of $\beta\eta$ conversion defined in 1.3. We define two judgments for canonical and atomic forms of well-typed terms e^τ, written judgmentally as $e^\tau \Uparrow$ and $e^\tau \Downarrow$.

$$\dfrac{}{x^\sigma \Downarrow}\,u$$

$$\vdots$$

$$\dfrac{e^o \Downarrow}{e^o \Uparrow}\,{\scriptstyle can\text{-}o} \qquad \dfrac{e^\tau \Uparrow}{lam\ x^\sigma.\, e^\tau \Uparrow}\,{\scriptstyle can\text{-}arr^{x,u}}$$

$$\dfrac{e_1^{\sigma\Rightarrow\tau} \Downarrow \quad e_2^\sigma \Uparrow}{app\ e_1^{\sigma\Rightarrow\tau}\ e_2^\sigma \Downarrow}\,atm\_app$$

Informally, canonical forms are normal in the sense that they are β-short and η-long, whereas atomic terms are variables applied to some number of canonical terms. It should be noted that, although any term of the form $e^{\sigma\Rightarrow\tau}$, can always η-expanded to $(lam\ x^\sigma.app\ e^{\sigma\Rightarrow\tau}\ x^\sigma)$, if $e^{\sigma\Rightarrow\tau}$ is canonical then $(lam\ x^\sigma.app\ e^{\sigma\Rightarrow\tau}\ x^\sigma)$ will β-reduce back to $e^{\sigma\Rightarrow\tau}$. Thus, although canonical terms can always be reduced, they can never be reduced in an interesting way.

We aim to prove a theorem of the form "for all e^τ, e^τ reduces to a canonical term." Although it is straightforward express the property "e^τ reduces to a canonical term" syntactically (i.e. by defining a judgment whose sole inference rule has premisses of the form $e^\tau \longrightarrow^* e'^\tau$ and $e'^\tau \Uparrow$), this alone is not enough to capture the potentially-nested logical connectives used to define a logical relation. Thus, we define an assertion logic that is expressive enough to describe both the logical connectives and, using atomic formulas, the property "can be converted to canonical form."

The formulas of the assertion logic for our example are defined by the following.

$$\begin{array}{lll} \textbf{Formulas} & F, G & ::= \; hc^{\tau}(e^{\tau}) \mid ha^{\tau}(e^{\tau}) \mid wh^{\tau}(e_1^{\tau}, e_2^{\tau}) \mid F \supset G \mid \forall x^{\tau}.F \\ \textbf{Predicates} & P, Q & ::= \; F \mid (x^{\tau}.P) \end{array}$$

Informally, we think of $hc^{\tau}(e^{\tau})$ as meaning "e^{τ}" has a normal form, $ha^{\tau}(e^{\tau})$ as meaning "e^{τ} has an atomic form" and $wh^{\tau}(e_1^{\tau}, e_2^{\tau})$ as meaning "e_1^{τ} weak head reduces to e_2^{τ}"; the logical connectives are intuitionistic. In general, we will omit type superscripts from expressions and formulas when they can easily be inferred from the context, or when they are irrelevant (e.g. $hc^{\tau}(app\ e_1\ e_2)$ or $hc(app\ e_1^{\sigma\Rightarrow\tau}\ e_2^{\sigma})$ instead of $hc^{\tau}((app\ e_1^{\sigma\Rightarrow\tau}\ e_2^{\sigma})^{\tau})$). A predicate is a formula with some number of distinguished bound variables that can be thought of as place-holders for arbitrary λ-calculus terms; if P is of the form $x^{\tau}.P'$, then we write $P(e^{\tau})$ to mean $P'[e^{\tau}/x^{\tau}]$.

We formalize the notion of provability using sequents of the form $\Psi \vdash F$, where Ψ is the notion of context defined below. We depart slightly from convention by using Ψ to keep track of not only which logical hypotheses may be used freely in the deduction of a sequent, but also which free λ-calculus variables may be used in the deduction as well. Some of these variables are meant to represent arbitrary λ-calculus terms (i.e. they are eigenvariables), and some are meant to represent arbitrary λ-calculus variables (e.g. they can be assumed to be atomic). We distinguish the latter from the former by adding a flag, $v^{x^{\tau}}$, to the context. We also depart slightly from convention by attaching names to the logical hypotheses in Ψ. Although doing so is not useful in presenting the proof rules, it simplifies the presentation of proof terms (i.e. the proof rules in BNF-style notation), which will be useful in Chapter 4. To this end, we assume the existence of a (dependent) syntactic category of *hypotheses*, whose only elements are *hypothetical variables* that we denote generically by h^F.

$$\textbf{Contexts} \quad \Psi \; ::= \; \cdot \mid \Psi, h^F \mid \Psi, x^{\tau} \mid v^{x^{\tau}}$$

In order for Ψ to be well formed, we require that each variable x^τ or h^F occurs in Ψ at most once; this condition can typically be satisfied by tacitly renaming bound variables. We sometimes abuse notation by writing Ψ, Ψ' for the concatenation of two contexts. We write $\Psi \supseteq \Psi'$ if Ψ can be obtained from Ψ' by deleting some number of declarations (we omit the inference rules for $\supseteq$ for the sake of brevity).

Our rules are formulated in the style of Pfenning [Pfe95], itself similar to **G3i** of [TS00]; the exact formulation of rules is immaterial to the applicability of our technique, although we feel that this presentation corresponds to an especially natural notion of proof term. We depart slightly from convention by denoting the occurrence of F in Ψ using the premiss $h^F \in \Psi$ (which can be expressed using inference rules in the obvious way), rather than writing Ψ as Ψ', h^F, Ψ''; we feel that this presentation makes the proof rules easier to read. The proof rules for the assertion logic are written below.

$$\frac{\Psi \vdash ha^{\tau_2 \Rightarrow \tau_1}(e_1) \quad \Psi \vdash hc^{\tau_2}(e_2)}{\Psi \vdash ha^{\tau_1}(app\ e_1\ e_2)}\ {\scriptstyle ha\text{-}app} \qquad \frac{\Psi, x^\sigma, v^{x^\sigma} \vdash hc^\tau(app\ e\ x)}{\Psi \vdash hc^{\sigma \Rightarrow \tau}(e)}\ {\scriptstyle hc\text{-}arr}$$

$$\frac{\Psi \vdash wh^o(e, e') \quad \Psi \vdash hc^o(e')}{\Psi \vdash hc^o(e)}\ {\scriptstyle hc\text{-}wh} \qquad \frac{\Psi \vdash ha^o(e)}{\Psi \vdash hc^o(e)}\ {\scriptstyle hc\text{-}atm} \qquad \frac{x^\tau \in \Psi \quad v^{x^\tau} \in \Psi}{\Psi \vdash ha^\tau(x^\tau)}\ {\scriptstyle ha\text{-}var}$$

$$\frac{}{\Psi \vdash wh^\tau(app\ (lam\ x.\, e_1)\ e_2, e_1[e_2/x])}\ {\scriptstyle wh\text{-}beta} \qquad \frac{\Psi \vdash wh^{\tau_2 \Rightarrow \tau_1}(e_1, e_1')}{\Psi \vdash wh^{\tau_1}(app\ e_1\ e_2, app\ e_1'\ e_2)}\ {\scriptstyle wh\text{-}app}$$

$$\frac{\Psi, h^F \vdash G}{\Psi \vdash F \supset G}\ {\scriptstyle impr} \qquad \frac{\Psi \vdash F_1 \quad \Psi, h_2^F \vdash G \quad h'^{F_1 \supset F_2} \in \Psi}{\Psi \vdash G}\ {\scriptstyle impl}$$

$$\frac{\Psi, x^\tau \vdash F}{\Psi \vdash \forall x^\tau . F}\ {\scriptstyle allr} \qquad \frac{\Psi, h^{F[e^\tau/x^\tau]} \vdash G \quad h'^{\forall x^\tau . F} \in \Psi}{\Psi \vdash G}\ {\scriptstyle alll} \qquad \frac{h^F \in \Psi}{\Psi \vdash F}\ {\scriptstyle axiom}$$

We consider the rules *hc-arr*, *impr*, *impl*, *allr* and *alll* to be binders for the variables introduced in the contexts of their subderivations. The informal description of the meanings of the atomic formulas is justified by Lemma 2.1.2.

Several of the proofs in this section are by induction over terms and derivations that can potentially contain free variables. Sometimes, especially when we are inducting over multiple different syntactic categories, we will need to be able to express the relationship between the free variables of different syntactic categories. For example, the below definition of *variable contexts* and *compatible derivations of* $e^\tau \Uparrow$ will be useful in the proof of Lemma 2.1.2.

Definition 2.1.1 (Variable Contexts, Compatible Canonicity Derivations) *We say that* Ψ *is a* variable context *iff the following judgment is inhabited.*

$$\frac{}{\cdot \ \texttt{varctx}} \qquad \frac{\Psi \ \texttt{varctx}}{\Psi, x^\tau, v^{x^\tau} \ \texttt{varctx}}$$

We say that a derivation of the form $\mathcal{D} : (e^\tau \Uparrow)$ *or* $\mathcal{D} : (e^\tau \Downarrow)$ *is* compatible *with a context* Ψ *iff for every free inference-rule variable* $u^{x^\tau \Downarrow}$ *in* $\mathcal{D}$*,* $x^\tau \in \Psi$ *and* $v^{x^\tau} \in \Psi$*. The notion of compatible canonicity derivation can easily be formulated judgmentally; we omit this definition for the sake of brevity.*

Lemma 2.1.2 (Extraction on Cut-Free Proofs) *For all* Ψ*, if* Ψ *is a variable context then:*

1. *for all* $\mathcal{D} : (\Psi \vdash hc^\tau(e))$ *there exists* e' *such that* $e \longrightarrow^* e'$ *and* $\mathcal{E} : (e' \Uparrow)$ *and* $\mathcal{E}$ *is compatible with* Ψ

2. *for all* $\mathcal{D} : (\Psi \vdash ha^\tau(e))$ *there exists* e' *such that* $e \longrightarrow^* e'$ *and* $\mathcal{E} : (e' \Downarrow)$ *and* $\mathcal{E}$ *is compatible with* Ψ

3. *for all* $\mathcal{D} : (\Psi \vdash wh^\tau(e_1, e_2))$ *there exists* $\mathcal{E} : (e_1 \longrightarrow e_2)$ *and* $\mathcal{E}$ *is compatible with* Ψ

Proof: By mutual induction on $\mathcal{D}$ in each case, using Theorem 1.3.1. $\square$

For practical reasons we will work with the assertion logic with cut. The judgment $\Psi \overset{\text{cut}}{\vdash} F$ is defined by inference rules similar to the ones above except with $\overset{\text{cut}}{\vdash}$ instead of $\vdash$, plus one additional rule.

$$\frac{\Psi \overset{\text{cut}}{\vdash} F \quad \Psi, h^F \overset{\text{cut}}{\vdash} G}{\Psi \overset{\text{cut}}{\vdash} G}\,{\scriptstyle cut}$$

The rules for defining the atomic formulas P vary for each structural logical relation. We show later in the paper that the choice of inference rules cannot be arbitrary: they must preserve some form of cut-elimination property of the sequent-calculus in order to apply Lemma 2.1.2 in the proof of Extraction.

Returning to the definition of *structural logical relations*, we can now define the logical relation judgmentally, in terms of a (unary) logical predicate.

$$\frac{}{[\![o]\!] = (x^o.hc(x^o))} \qquad \frac{[\![\sigma]\!] = P_\sigma \quad [\![\tau]\!] = P_\tau}{[\![(\sigma \Rightarrow \tau)]\!] = (x^{\sigma\Rightarrow\tau}.\forall y^\sigma.P_\sigma(y^\sigma) \supset P_\tau(app\ x^{\sigma\Rightarrow\tau}\ y^\sigma))}$$

We are justified in treating $[\![\tau]\!] = P$ as a function by the following lemma.

Lemma 2.1.3 (Existence, Uniqueness of Logical Relation Predicate)

1. *For every τ, there exists a predicate $x^\tau.F$ such that $[\![\tau]\!] = x^\tau.F$*

2. *If $[\![\tau]\!] = P$ and $[\![\tau]\!] = P'$ then $P = P'$*

Proof: Each case is by a straightforward induction on the structure of τ. □

Thus, $e^\tau \in [\![\tau]\!]$ and $[\![\tau]\!]e^\tau$ can both be viewed as shorthand for the formula $F[e^\tau/x^\tau]$ where $[\![\tau]\!] = x^\tau.F$.

Our structural logical relations argument is structured as follows.

1. For all τ, $\cdot \overset{\text{cut}}{\vdash} [\![\tau]\!](e^\tau) \supset hc(e^\tau)$. (Escape Lemma)

2. For all e^τ there exists $\mathcal{D} : (\cdot \overset{\text{cut}}{\vdash} [\![\tau]\!](e^\tau))$. (Fundamental Theorem)

3. If $\overset{\text{cut}}{\vdash} hc(e^\tau)$ then there exists e'^τ

   such that $e^\tau \longrightarrow^* e'^\tau$ and $e'^\tau \Uparrow$. (Extraction)

Note the distinction between the statement of 3 and Lemma 2.1.2; in order to bridge the gap between the two, we will need to prove some sort of cut-elimination result.

Lemma 2.1.4 (Escape)

1. *For all contexts* Ψ *and types* τ*, there exists* $\mathcal{D} : \Psi \overset{\text{cut}}{\vdash} \forall x^\tau.[\![\tau]\!](x^\tau) \supset hc(x^\tau)$

2. *For all contexts* Ψ *and types* τ*, there exists* $\mathcal{D} : \Psi \overset{\text{cut}}{\vdash} \forall x^\tau.ha(x^\tau) \supset [\![\tau]\!](x^\tau)$

Proof: By mutual induction on τ.

Case: $\tau = o$. Direct, by reasoning using the definition of $[\![\tau]\!]$ and rules *allr, impr* and *axiom* in 1, and the same rules plus *ha-var* and *hc-atm.*

Case: $\tau = \tau_2 \Rightarrow \tau_1$. By induction hypothesis, we can obtain derivations of the following sequents.

$$\Psi \overset{\text{cut}}{\vdash} \forall x^{\tau_1}.\, [\![\tau_1]\!](x^{\tau_1}) \supset hc(x^{\tau_1}) \tag{2.1}$$

$$\Psi \overset{\text{cut}}{\vdash} \forall x^{\tau_1}.\, ha(x^{\tau_1}) \supset [\![\tau_1]\!](x^{\tau_1}) \tag{2.2}$$

$$\Psi \overset{\text{cut}}{\vdash} \forall x^{\tau_2}.\, [\![\tau_2]\!](x^{\tau_2}) \supset hc(x^{\tau_2}) \tag{2.3}$$

$$\Psi \overset{\text{cut}}{\vdash} \forall x^{\tau_2}.\, ha(x^{\tau_2}) \supset [\![\tau_2]\!](x^{\tau_2}) \tag{2.4}$$

1. Follows by induction hypotheses (2.1) and (2.4), using the definition of $[\![\tau]\!]$ and rules *allr*, *impr*, *alll*, *impl*, *cut*, and *hc-arr*.

2. Follows by induction hypotheses (2.2) and (2.3), using rules *allR*, *impR*, *allL*, *cut*, and *ha-app*.

□

The following lemma will be needed to prove the Fundamental Theorem.

Lemma 2.1.5 (Closure Under Weak Head Expansion) *For all contexts Ψ and for all types τ, there exists $\mathcal{D} : \Psi \overset{\text{cut}}{\vdash} \forall x^\tau. \forall y^\tau. wh(x^\tau, y^\tau) \supset [\![\tau]\!](y^\tau) \supset [\![\tau]\!](x^\tau)$*

Proof: By induction on the structure of τ.

Case: $\tau = o$. Direct, by the definition of $[\![\tau]\!]$ and rules *allr*, *impr*, *axiom*, and *hc-wh*.

Case: $\tau = \tau_2 \Rightarrow \tau_1$. By induction hypothesis, we can obtain a derivation of

$$\Psi \overset{\text{cut}}{\vdash} \forall x^{\tau_1}. \forall e'^{\tau_1}. wh^{\tau_1}(e, e') \supset [\![\tau_1]\!](e') \supset [\![\tau_1]\!](e)$$

The claim follows from the definition of $[\![\tau]\!]$ and rules *allr*, *impr*, *alll*, *impl*, *cut*, and *wh-app*.

□

The statement of the Fundamental Theorem does not rely on lifting the notion of logical relations to simultaneous substitutions, but it does rely on lifting the notion of logical relation to contexts, which plays a similar role in the proof of the fundamental theorem as the notion of variable context played in the proof of Lemma 2.1.2.

Definition 2.1.6 (Contexts in the LR, Compatible λ-Calculus Terms) *We say that Ψ is* in the logical relation *iff the following judgment is inhabited.*

$$\frac{}{\cdot\ \texttt{inLR}} \qquad \frac{\Psi\ \texttt{inLR} \quad [\![\tau]\!] = P}{(\Psi, x^\tau, h^{P(x^\tau)})\ \texttt{inLR}}$$

We say that an expression e^τ is compatible *with Ψ iff all of the free variables in e^τ are declared in Ψ. This notion of compatible expression can be defined judgmentally, although we omit its definition for the sake of brevity.*

Theorem 2.1.7 (Fundamental) *For every e^τ and for every Ψ, if Ψ is in the logical relation and e^τ is compatible with Ψ then $[\![\tau]\!] = P$ and $\Psi \overset{\text{cut}}{\vdash} P(e^\tau)$*

Proof: By induction on e^τ.

Case: x^τ

Ψ `inLR` and x^τ is compatible with Ψ — given

$x^\tau \in \Psi$ — by def of compatibility

$h^{P(x^\tau)} \in \Psi$ and $[\![\tau]\!] = P$

by a straightforward induction on Ψ `inLR` using $x^\tau \in \Psi$

Case: $lam\ x^\sigma.e^\tau$

Ψ `inLR` and e^τ is compatible with Ψ — given

$[\![\sigma]\!] = P_\sigma$ — by Lemma 2.1.3

$(\Psi, x^\sigma, h^{P_\sigma(x^\sigma)})$ `inLR` — by rule

e^τ is compatible with $\Psi, x^\sigma, h^{P_\sigma(x^\sigma)}$ — by def of compatibility

$\Psi, x^\sigma, P_\sigma(x^\sigma) \overset{\text{cut}}{\vdash} P_\tau(e^\tau)$ and $[\![\tau]\!] = P_\tau$ — by IH

$\Psi, x^\sigma, P_\sigma(x^\sigma) \overset{\text{cut}}{\vdash} wh(app\ (lamx^\sigma.e^\tau)\ x^\sigma, e^\tau)$ — by rule *wh-beta*

$\Psi, x^\sigma, P_\sigma(x^\sigma) \overset{\text{cut}}{\vdash} \forall y^\tau. \forall y'^\tau. wh(y^\tau, y'^\tau) \supset P_\tau(y'^\tau) \supset P_\tau(y^\tau)$ — by Lemma 2.1.5

$\Psi \overset{\text{cut}}{\vdash} \forall y^\sigma.P_\sigma(y^\sigma) \supset P_\tau(app\ (lam\ x^\sigma.e^\tau)\ y^\sigma)$

by rules *allr*, *impr*, *cut*, *alll* *impl* and *axiom*

and the renamability of bound variables

$[\![\tau]\!](lamx^\sigma.e^\tau) = \forall y^\sigma.P_\sigma(y^\sigma) \supset P_\tau(app\ (lam\ x^\sigma.e^\tau)\ y^\sigma)$

Case: $app\ e_1^{\sigma\Rightarrow\tau}\ e_2^\sigma$

Ψ `inLR` and $app\ e_1^{\sigma\Rightarrow\tau}\ e_2^\sigma$ is compatible with Ψ — given

$\Psi \vdash P_1(e_1^{\sigma \Rightarrow \tau})$ and $[\![\sigma \Rightarrow \tau]\!] = P_1$ by IH on e_1

$P_1 = (x^{\sigma \Rightarrow \tau}.\forall y^{\sigma}.P_{\sigma}(y^{\sigma}) \supset P_{\tau}(app\ x^{\sigma \Rightarrow \tau}\ y^{\sigma}))$

and $[\![\sigma]\!] = P_{\sigma}$ and $[\![\tau]\!] = P_{\tau}$ by inversion on $[\![\sigma \Rightarrow \tau]\!]$

$\Psi \vdash P_2(e_2^{\sigma})$ and $[\![\sigma]\!] = P_{\sigma}$ by IH on e_2

$P_2 = P_{\sigma}$ by Lemma 2.1.3

$\Psi \vdash P_{\tau}(app\ e_1^{\sigma \Rightarrow \tau}\ e_2^{\sigma})$ by rules *cut*, *foralll*, *impl* and *axiom*

□

Thus far, we have shown that for any well-typed e^{τ}, we can produce a derivation of $\cdot \overset{\text{cut}}{\vdash} hc(e^{\tau})$ by applying the Fundamental Theorem, followed by the Escape Lemma, followed by applications of the rules *cut*, *foralll*, and *impl*. However, we are not yet done: Lemma 2.1.2 requires the proof of $hc^{\tau}(e)$ to be *cut*-free; that is, the assertion logic must be consistent. If we were willing to accept the consistency of the assertion logic *a priori*, then we would be done. However, in this example, no additional assumptions are necessary: we are able to prove cut elimination directly, using the syntactically finitary procedure outlined in [Pfe95]. First, we prove that *cut* is an admissible rule in the cut-free sequent calculus, then use this admissibility lemma to prove full cut elimination.

It should be noted that, in the proof the following Lemma 2.1.8, we consider a formula of the form $F[e^{\tau}/x^{\tau}]$ to be a subterm of $\forall x^{\tau}.F$, despite the fact that e^{τ} may in fact be considerably larger than x^{τ}. We justify this heretofore undiscussed property of the subterm ordering by noting that, although λ-calculus terms may occur inside of formulas, the reverse does not hold, and thus λ-calculus terms cannot meaningfully impact the structure of a formula. In general, we only consider the terms from one syntactic category to be taken into consideration when considering the size of terms from another syntactic category when the two syntactic categories are mutually-recursive. This assumption is made more mathematically precise in Example 5.2.42.

Lemma 2.1.8 (Cut Admissibility) *If $\mathcal{D} : (\Psi \vdash F)$ and $\mathcal{E} : (\Psi, h^F \vdash G)$ then $\Psi \vdash G$*

Proof: By induction on the lexicographic ordering of F, followed by the simultaneous ordering on the structure of $\mathcal{D}$ and $\mathcal{E}$. See [Pfe95] for details. Because the only proof rules for atomic formulas are all right-rules, the extra cases introduced by our atomic propositions all fall into the category of (easy) "right commutative conversions.' $\square$

Theorem 2.1.9 (Cut-elimination) *For any Ψ, and for any $\mathcal{D} : (\Psi \overset{\text{cut}}{\vdash} F)$ there exists $\mathcal{E} : (\Psi \vdash F)$*

Proof: By straightforward induction on the structure of $\mathcal{D}$, using Lemma 2.1.8. $\square$

Corollary 2.1.10 (Extraction) *If $\cdot \overset{\text{cut}}{\vdash} hc^\tau(e)$ then there exists e' such that $e \longrightarrow^* e'$ and $e' \Uparrow$*

Proof: By Theorem 2.1.9 and Lemma 2.1.2 $\square$

The following theorem summarizes all the work we have done so far. It shows that the simply typed λ-calculus is weakly normalizing.

Theorem 2.1.11 (Weak Normalization) *For any τ and for any (closed) e^τ, there exists e'^τ, such that $e'^\tau \Uparrow$ and $e^\tau \longrightarrow^* e'^\tau$.*

Proof: Direct, by Theorem 2.1.7, Lemma 2.1.4, the rules *cut*, *alll*, *impl*, *axiom* and Corollary 2.1.10. $\square$

2.2 Normalization of Gödel's T

Proofs by logical relations are popular in large part because they tend to scale well. We believe that structural logical relations preserve this property. Here, we extend our proof of weak normalization to Gödel's T, an extension of the simply-typed λ-calculus with terms for expressing natural numbers (i.e. zero and successor), and primitive recursion operator for each type τ.[1]

$$e^\tau \quad ::= \quad \ldots \mid z^o \mid (s\ (e^o))^o \mid (r_\tau\ ({e_1}^\tau)\ ({e_2}^{\tau\Rightarrow o\Rightarrow\tau})\ ({e_3}^o))^\tau$$

We will see that the definition of the logical relation is unchanged, as are most of the theorems; however, in order to prove the Fundamental Theorem, we will need a proof-theoretically non-trivial extension to the assertion logic.

The new reduction rules for Gödel's T are defined below. As usual, we omit type superscripts when they are easily inferred or irrelevant.

$$\frac{e^o \longrightarrow e'^o}{s\ e^o \longrightarrow s\ e'^o}\ \textit{red-s} \qquad \frac{}{r_\tau\ ({e_2}^\tau)\ ({e_3}^{\tau\Rightarrow o\Rightarrow\tau})\ (z^o) \longrightarrow e_2^\tau}\ \textit{red-rz}$$

$$\frac{}{r_\tau\ e_1\ e_2\ (s\ e_3) \longrightarrow \mathit{app}\ (\mathit{app}\ e_2\ (r_\tau\ e_1\ e_2\ e_3))\ e_3}\ \textit{red-rs}$$

$$\frac{e_1 \longrightarrow e_1'}{r_\tau\ e_1\ e_2\ e_3 \longrightarrow r_\tau\ e_1'\ e_2\ e_3}\ \textit{red-rc1} \qquad \frac{e_2 \longrightarrow e_2'}{r_\tau\ e_1\ e_2\ e_3 \longrightarrow r_\tau\ e_1\ e_2'\ e_3}\ \textit{red-rc2}$$

$$\frac{e_3 \longrightarrow e_3'}{r_\tau\ e_1\ e_2\ e_3 \longrightarrow r_\tau\ e_1\ e_2\ e_3'}\ \textit{red-rc3}$$

The definition of canonical and atomic forms must be updated for the new expressions.

$$\frac{}{z^o \Uparrow}\ \textit{can-z} \qquad \frac{e^o \Uparrow}{s\ e^o \Uparrow}\ \textit{can-s} \qquad \frac{e_1^\tau \Uparrow \quad e_2^{\tau\Rightarrow o\Rightarrow\tau} \Uparrow \quad e_3^o \Downarrow}{r_\tau\ ({e_1}^\tau)\ ({e_2}^{\tau\Rightarrow o\Rightarrow\tau})\ ({e_3}^o) \Downarrow}\ \textit{atm-r}$$

[1]This is a slight abuse of terminology: Gödel actually referred to something computationally equivalent to these terms as *computable functions of finite type*, whereas his theory "T" was a quantifier-free logic whose atomic formulas were equations between computable functions of finite type [Göd58]. We justify ourselves by noting that the normalization of the former implies the consistency of the latter, and vice versa.

The definition of multi-step reduction is unchanged, although we need to add the following additional cases to the congruence theorem (Theorem 1.3.1).

Theorem 2.2.1 (Congruence of Multistep Reduction)

...

3. If $e^o \longrightarrow^* e'^o$ *then* $s\ e^o \longrightarrow^* s\ e'^o$

4. If $e_1 \longrightarrow^* e_1$ *and* $e_2 \longrightarrow^* e_2$ *and* $e_3 \longrightarrow^* e_3$ *then* $r_\tau\ e_1\ e_2\ e_3 \longrightarrow^* r_\tau\ e'_1\ e'_2\ e'_3$

Proof: The statement and proofs of 1 and 2 are the same as Theorem 1.3.1. 3 is by straightforward induction on the structure of the given derivation, and 4 is by straightforward simultaneous induction on the structures of the given derivations. □

The definition of $[\![\tau]\!]$ for Gödel's T is identical to the judgment defined in Section 2.1 for the simply typed λ-calculus, as are the proofs of Lemma 2.1.3, the Escape Lemma and Closure Under Weak Head Expansion, even though the notion of weak head reduction must be expanded with the following rules.

$$\frac{\Psi \vdash wh(e_3, e'_3)}{\Psi \vdash wh(r_\tau\ e_1\ e_2\ e_3, r_\tau\ e_1\ e_2\ e'_3)}\ \textit{wh-rc} \qquad \frac{}{\Psi \vdash wh((r_\tau\ e_1\ e_2\ z^o), e_1)}\ \textit{wh-rz}$$

$$\frac{}{\Psi \vdash wh((r_\tau\ e_1\ e_2\ (s\ e_3)), (app\ (app\ e_2\ (r_\tau\ e_1\ e_2\ e_3))\ e_3))}\ \textit{wh-rs}$$

In addition, *hc* and *ha* need the following additional right rules.

$$\frac{}{\Psi \vdash hc(z^o)}\ \textit{hc-z} \qquad \frac{\Psi \vdash hc(e^o)}{\Psi \vdash hc(s\,e^o)}\ \textit{hc-s} \qquad \frac{\Psi \vdash hc(e_1) \quad \Psi \vdash hc(e_2) \vdash \Psi \vdash ha(e_3)}{\Psi \vdash ha(r_\tau\ e_1\ e_2\ e_3)}$$

None of the above extensions to the assertion logic is at all problematic. However, in order to prove the Fundamental Theorem, we will need to make an extension to the assertion logic of non-trivial proof-theoretic strength in the form of the following left-rule for hc^o.

$$\frac{\begin{array}{c}\Psi \overset{\text{cut}}{\vdash} P(z^o) \quad \Psi, x^o, h^{P(x^o)} \overset{\text{cut}}{\vdash} P(s\ x^o) \quad \Psi, x^o, h^{ha(x^o)} \overset{\text{cut}}{\vdash} P(x^o) \\ \Psi, x^o, y^o, h_1^{wh(x^o,y^o)}, h_2^{P(y^o)} \overset{\text{cut}}{\vdash} P(x^o) \quad \Psi, h^{P(e^o)} \overset{\text{cut}}{\vdash} F \quad h'^{hc^o(e^o)} \in \Psi\end{array}}{\Psi \overset{\text{cut}}{\vdash} F}\ \mathit{hcl}$$

The rule *hcl* can be thought of as defining a catamorphism-like induction principle, not on types or terms, but on proofs of $hc^o(e)$: the predicate P plays the role of induction hypothesis, and for each of the right rules that can be used to prove $hc^o(e^o)$—*hc-z*, *hc-s*, *hc-atm* and *hc-wh*—*hcl* has a *minor premiss* that plays the role of induction case (we refer to $\Psi, h^{P(e^o)} \vdash F$ as the *major premiss*). In other words, the atomic predicate-symbol hc^o can be thought of as an inductive definition in the style of Martin-Löf [ML71]. This should not come as a great surprise: in order to reason about a λ-calculus with primitive recursion, we use an assertion logic with a notion of iteration. In the absence of pairs, iteration and primitive recursion aren't obviously the same thing, so it should also come as no surprise that we find it convenient to augment the assertion logic with conjunction.

$$\textbf{Formulas} \quad F, G \quad ::= \quad \ldots \mid F \wedge G$$

$$\frac{\Psi \overset{\text{cut}}{\vdash} F \quad \Psi \overset{\text{cut}}{\vdash} G}{\Psi \overset{\text{cut}}{\vdash} F \wedge G}\ \mathit{andr}$$

$$\frac{\Psi, h^F \overset{\text{cut}}{\vdash} G \quad h'^{F\wedge G} \in \Psi}{\Psi \overset{\text{cut}}{\vdash} F \wedge G}\ \mathit{andr1} \qquad \frac{\Psi, h^G \overset{\text{cut}}{\vdash} G \quad h'^{F\wedge G} \in \Psi}{\Psi \overset{\text{cut}}{\vdash} F \wedge G}\ \mathit{andr2}$$

The fundamental theorem is mostly the same as before, but with some new cases, one of which relies on the Escape Lemma. For the sake of convenience, we prove that weakening is an admissible rule for the assertion logic; the other structural rules will be useful in Chapter 4.

Lemma 2.2.2 (Structural Rules for the Assertion Logic)

(Weakening) If $\mathcal{D} : (\Psi \overset{\text{cut}}{\vdash} F)$ *and* $\mathcal{E} : (\Psi' \supseteq \Psi)$ *then* $\Psi' \overset{\text{cut}}{\vdash} F$

(Exchange) If $\Psi, h^G, h'^{G'}, \Psi' \overset{\text{cut}}{\vdash} F$ *then* $\Psi, h'^{G'}, h^G, \Psi' \overset{\text{cut}}{\vdash} F$

(Contraction) If $\Psi_0, h^G, \Psi_1, h'^G, \Psi_2 \overset{\text{cut}}{\vdash} F$ *then* $\Psi_0, h^G, \Psi_1, \Psi_2 \overset{\text{cut}}{\vdash} G$ *and* $\Psi_0, \Psi_1, h'^G, \Psi_2 \overset{\text{cut}}{\vdash} F$

(Substitution) If $\Psi, x^\tau, \Psi' \overset{\text{cut}}{\vdash} F$ *and* $fv(e^\tau) \subseteq \Psi$ *then* $\Psi, \Psi'[e^\tau/x] \overset{\text{cut}}{\vdash} G[e^\tau/x^\tau]$

Proof: Weakening is by straightforward induction on the structure of $\mathcal{D}$, where the *axiom* case is proven by a straightforward induction on the structure of $\mathcal{E}$; Exchange, contraction and substitution are straightforward inductions on the given derivations. □

Recall that definitions of Ψ `inLR` and the compatibility of e^τ with Ψ have not changed from Definition 2.1.6.

Theorem 2.2.3 (Fundamental) *For every* e^τ *and for every* Ψ*, if* Ψ *is in the logical relation and* e^τ *is compatible with* Ψ *then* $[\![\tau]\!] = P$ *and* $\Psi \overset{\text{cut}}{\vdash} P(e^\tau)$

Proof: Exactly the same as Theorem 2.1.7 (i.e. by induction on e^τ), but with new cases. We show them below.

Case: z^o

Ψ `inLR` and z^o is compatible with Ψ	given
$[\![o]\!] = x^o.hc(x^o)$	by def of $[\![o]\!]$
$\Psi \overset{\text{cut}}{\vdash} hc(z^o)$	by rule *hc-z*

Case: $s\ e^o$

Ψ `inLR` and $s\ e^o$ is compatible with Ψ given

e^o is compatible with Ψ by def of compatible

$\Psi \overset{\text{cut}}{\vdash} P(e^o)$ and $[\![o]\!] = P$ by IH on e

$P = x^o.hc(x^o)$ by inversion on $[\![o]\!]$

$\Psi \overset{\text{cut}}{\vdash} hc(s\ e^o)$ by rule *hc-s*

Case: $r_\tau\ (e_1{}^\tau)\ (e_2{}^{\tau\Rightarrow o\Rightarrow\tau})\ (e_3{}^o)$

Ψ `inLR` and $r_\tau\ (e_1{}^\tau)\ (e_2{}^{\tau\Rightarrow o\Rightarrow\tau})\ (e_3{}^o)$ is compatible with Ψ given

$\mathcal{D}_1 : (\Psi \overset{\text{cut}}{\vdash} P_1(e_1^\tau))$ and $[\![\tau]\!] = P_1$ by IH on e_1^τ

$\mathcal{D}_2 : (\Psi \overset{\text{cut}}{\vdash} P_2(e_2^{\tau\Rightarrow o\Rightarrow\tau}))$ and $[\![\tau \Rightarrow o \Rightarrow \tau]\!] = P_2$ by IH on $e_2^{\tau\Rightarrow o\Rightarrow\tau}$

$P_2 = x_2^{\tau\Rightarrow o\Rightarrow\tau}.\forall x_1^\tau.P_\tau(x_1^\tau) \supset \forall x_3^o.hc(x_3^o) \supset P_\tau(app\ (app\ x_2^{\tau\Rightarrow o\Rightarrow\tau}\ x_1^\tau)\ x_3^o)$

and $[\![\tau]\!] = P_\tau$ by inversion on $[\![\tau \Rightarrow o \Rightarrow \tau]\!] = P_2$

$P_1 = P_\tau$ by Lemma 2.1.3

$\mathcal{E}_0 : (\Psi, h'^{hc(e_3)} \overset{\text{cut}}{\vdash} \forall x^\tau.\forall y^\tau.wh(x^\tau, y^\tau) \supset P_\tau(y^\tau) \supset P_\tau(x^\tau))$

by Lemma 2.1.5

$\mathcal{E}_1 : (\Psi \overset{\text{cut}}{\vdash} \forall x^\tau.P_\tau(x^\tau) \supset hc(x^\tau))$ by Lemma 2.1.4 (1)

$\mathcal{E}_2 : (\Psi \overset{\text{cut}}{\vdash} \forall x^\tau.ha(x^\tau) \supset P_\tau(x^\tau))$ by Lemma 2.1.4 (2)

$\mathcal{E}_3 : (\Psi \overset{\text{cut}}{\vdash} \forall x^{\tau\Rightarrow o\Rightarrow\tau}.P_2(x^{\tau\Rightarrow o\Rightarrow\tau}) \supset hc(x^{\tau\Rightarrow o\Rightarrow\tau}))$ by Lemma 2.1.4 (1)

let $P = x^o.(P_\tau(r_\tau\ (e_1{}^\tau)\ (e_2{}^{\tau\Rightarrow o\Rightarrow\tau})\ (x^o)) \wedge hc(x^o))$ (convenient definition)

$\Psi, h'^{hc(e_3)} \overset{\text{cut}}{\vdash} P(z^o)$

by rules *andr, cut, alll, impl, axiom, wh-rz, hc-z*

and Lemma 2.2.2 on $\mathcal{E}_0$ and $\mathcal{D}_1$

$\Psi, h'^{hc(e_3^o)}, x^o, h^{P(x^o)} \overset{\text{cut}}{\vdash} P(s\ x^o)$

by rules *andr, cut, alll, impl, andl1, andl2, axiom, hc-s*

and Lemma 2.2.2 on $\mathcal{E}_0$ and $\mathcal{D}_2$

$\Psi, h'^{hc(e_3^o)}, x^o, h^{ha(x^o)} \overset{\text{cut}}{\vdash} P(x^o)$

by rules *andr, cut, alll, impl, ha-r, hc-atm*

and Lemma 2.2.2 on $\mathcal{E}_1, \mathcal{D}_1, \mathcal{E}_3, \mathcal{D}_2$ and $\mathcal{E}_2$

$\Psi, h'^{hc(e_3^o)}, x^o, y^o, h_1^{wh(x^o,y_o)}, h_2^{P(y^o)} \vdash^{\text{cut}} P(x^o)$

by rules *andr, cut, alll, impl, wh-rc, axiom, andl1, hc-wh, andl2*

and Lemma 2.2.2 on $\mathcal{E}_0$

$\Psi, h'^{hc(e_3^o)}, h^{P(e_3^o)} \vdash^{\text{cut}} P_\tau(r_\tau\ (e_1{}^\tau)\ (e_2{}^{\tau\Rightarrow o\Rightarrow\tau})\ (e_3{}^o))$

by rules *andl1* and *axiom*

$\Psi, h'^{hc(e_3^o)} \vdash^{\text{cut}} P_\tau(r_\tau\ (e_1{}^\tau)\ (e_2{}^{\tau\Rightarrow o\Rightarrow\tau})\ (e_3{}^o))$ by rule *hcl*

$\Psi \vdash^{\text{cut}} hc(e_3^o)$ by IH on e_3^o by IH on e_3^o and def of $[\![o]\!]$

$\Psi \vdash^{\text{cut}} P_\tau(r_\tau\ (e_1{}^\tau)\ (e_2{}^{\tau\Rightarrow o\Rightarrow\tau})\ (e_3{}^o))$ by rule *cut*

□

As before, all that remains to be proven is the Extraction theorem for the assertion logic. However, as we shall see in Chapter 3, the syntactically finitary methods that we have considered thus far are simply too weak to prove such a theorem. In Chapter 4, we will see how syntactic finitism can be extended such that this difficulty can be overcome.

Chapter 3

Ordinal Analysis

In the previous chapter, we demonstrated that structural logical relations can be viewed as syntactically finitary reductions from one sort of consistency theorem (e.g. normalization for a typed λ-calculus) to another (the extraction theorem for an assertion logic with cut). In the case of the simply typed λ-calculus, syntactically finitary methods are enough to prove the relevant extraction theorem directly, and thus normalization can be proved outright. In general this is not always be the case. For example, we have seen that structural logical relations can be used to perform a similar reduction for Gödel's T, but, as we shall see, the syntactically finitary methods outlined thus far are simply too proof-theoretically weak to prove the normalization of Gödel's T outright. We will demonstrate this point semi-formally, using some of the ideas and results from the branch of proof theory known as *ordinal analysis*, in which consistency theorems are finitistically reduced to well-ordering theorems for *natural* systems of ordinal notations (see [Rat06] for a good introduction to the field). Many of the theorems in this chapter employ techniques that go well beyond what we consider to be syntactically finitary, although we will attempt to remain as true to the spirit of syntactic finitism as possible.

3.1 An Introduction to Ordinals

In mathematics, as in linguistics, the distinction between cardinal and ordinal numbers is the distinction between quantity and order; in English, the words *one*, *two* and *three* are cardinal numbers, whereas *first*, *second* and *third* are ordinal numbers. In mathematics, the distinction is somewhat analogous; cardinal numbers denote size, whereas ordinal numbers denote well-orderings.

Below we informally describe the intuition behind some of the basic ordinals, and ordinal operations, that we consider important for this dissertation. A more formal development of much of this material can be found in [Gal91], or, alternatively, nearly any textbook on set theory or proof theory (e.g. [Kun80, TS00]).

3.1.1 Ordinal Arithmetic

Consider the syntactic category of even and odd natural numbers, as defined below.

Even Numbers	E	$::=$	$\mathit{zero} \mid \mathit{plustwo}_{\mathit{even}}\ E$
Odd Numbers	O	$::=$	$\mathit{one} \mid \mathit{plustwo}_{\mathit{odd}}\ O$
Natural Numbers with Parity	N	$::=$	$E \mid O$

Clearly, the syntactic categories E, O, N, and n (defined in Section 1.1) all have the same cardinality, since there are obvious bijections between each of them. In particular, N can be viewed as a natural alternative to n: the two syntactic categories are clearly intended to represent the same concept. Under the subterm orderings for n, E and O—written $<$, $<_E$ and $<_O$, respectively—these syntactic categories are clearly *order-isomorphic* as well; their *order-type*—that is, the name for this equivalence class of total well-orderings on syntactic categories—is usually denoted using the symbol ω. An *ordinal* is a canonical element of an order type; for the purposes of this exposition, we conflate these concepts somewhat by referring to order-types and ordinals with the same names.

Consider the following ordering relation $<_N$ on N, as defined below.

$$\frac{E <_E E'}{E <_N E'} \qquad \frac{O <_O O'}{O <_N O'} \qquad \frac{}{E <_N O}$$

The relation $<_N$ is clearly a well-ordering: any strictly monotonically decreasing sequence of Es and Os is finite, and any strictly monotonically decreasing sequence of N's can be split into a sequence of Os followed by a sequence of Es. If we were to make an analogy to type theory, N can be thought of as the sum-type $E + O$, where $<_N$ is the lifting of $<_E$ and $<_O$ to $<_{E+O}$ by arbitrarily declaring terms tagged by *inl* to be smaller than terms tagged by *inr*. In the realm of ordinals, this operation defines *ordinal addition*, and where the order type of N under $<_N$ is usually denoted $\omega + \omega$ or as $\omega \cdot 2$. Because there is an order-preserving embedding of ω into $\omega + \omega$, but not vice-versa, we think of ω as being smaller than $\omega + \omega$ in much the same way that we would consider the syntactic category E to be smaller than the syntactic category N. We overload the symbols $<$, $\leq$ and $=$—normally reserved for comparing natural numbers—to refer to this comparison of ordinals as well; the relation $<$ is both total and well-founded.

In some ways, ordinal addition behaves similarly to addition on natural numbers; it is associative, and the uninhabited syntactic category $false$, whose order type is usually written using the symbol 0, is the identity element for this operation. Moreover, the syntactic category *true*, whose sole element is $\langle\rangle$, has an order type usually denoted using the symbol 1, and behaves much like the natural number of the same name. In general, any natural number n corresponds to a *finite* ordinal that can be written $\overbrace{1 + \ldots + 1}^{n \text{ times}}$. The finite ordinals behave exactly the same under ordinal addition (defined above) and ordinal multiplication (defined below) as the natural numbers do under arithmetic addition and multiplication (thus we occasionally treat one as though it were the other for the sake of convenience). However, this analogy only takes us so far: in general, ordinal addition is not commutative. For example, $1+\omega$ is equal to ω (consider the bijective, order-preserving function that maps *inl* $\langle\rangle$ to z, and *inr* n to $s\,n$), but $\omega + 1$ is not.

Every ordinal which is neither 0 nor a *successor ordinal* (i.e. can be written $\alpha+1$) is defined to be a *limit ordinal.* In general, every limit ordinal can be described as smallest ordinal that is strictly larger than every element of a strictly increasing, infinite sequence of ordinals. For example, the ordinal ω is the limit of the sequence $0, 1, 2, \ldots$; the ordinal $\omega+\omega$ is the limit of the sequence $\omega, \omega+1, \omega+2, \ldots$; the ordinal $\omega + \omega + \omega$ is the limit of the sequence $\omega + \omega, \omega + \omega + 1, \omega + \omega + 2, \ldots$; etcetera. In general each limit ordinal can be associated with a canonical *fundamental sequence*[1] , although the exact definition of fundamental sequence varies depending on the presentation; we write the nth element of the fundamental sequence for α as $\alpha[n]$. We have already seen what the fundamental sequences look like for ω, $\omega+\omega$, $\omega+\omega+\omega$, etc. The notion of *ordinal multiplication* will allow us to define the ordinal $\omega \cdot \omega$, whose fundamental sequence is defined as $(\omega \cdot \omega)[n] = \overbrace{\omega + \ldots + \omega}^{n \text{ times}} = \omega \cdot n$.

The ordinary, arithmetic notion of multiplication can be defined in terms of repeated addition. The same is true for ordinal multiplication, but we prefer to describe its behavior as a primitive notion. Given any two syntactic categories A and B, we can define a syntactic category whose sole term constructor is $\langle A; B\rangle$; such a construction is analogous to the product-type $A \times B$. We can use the notion of lexicographic ordering, discussed in Section 1.2, to lift well-orderings $<_A$ (on terms in A) and $<_B$ (on terms in B) to $<_{lex}$ (on terms in $A \times B$). This is formalized by

[1]In general, it will be possible to describe the fundamental sequences for all of the ordinals discussed in this document syntactically, where the notion of fundamental sequence can be formalized as syntactically finitary, deterministic, strictly increasing functions from the natural numbers to the ordinals.

the judgments below.

$$\frac{A <_A A'}{\langle A; B\rangle <_{lex} \langle A'; B'\rangle} \qquad \frac{A = A' \quad B <_B B'}{\langle A; B\rangle <_{lex} \langle A'; B'\rangle}$$

As we have already discussed in Section 1.2, we view the well-foundedness of lexicographic combinations of orderings as being *a priori* justified. As it turns out, this use of the lexicographic ordering corresponds to ordinal multiplication: $<_A$ has order type α, and $<_B$ has order-type β, then $<_{lex}$, as defined above, has order type $\beta \cdot \alpha$ (note the swap in the placement of α and β). As is the case with ordinal addition, ordinal multiplication is associative but not commutative; $2 \cdot \omega$ is equal to ω, but $\omega \cdot 2$ is not. Ordinal multiplication distributes over addition on the right, but not on the left: for any α and β, $\alpha \cdot (\beta_1 + \beta_2) = \alpha \cdot \beta_1 + \alpha \cdot \beta_2$, but, as we have already seen, $(1 + 1) \cdot \omega = \omega \neq \omega \cdot (1 + 1)$.

The definition of ordinal exponentiation, β^α, is considerably more complex than ordinal addition or multiplication. However, for our purposes, it suffices to consider only exponentiation at base ω. Given a syntactic category A and the total, well-founded ordering $<_A$, let L be a syntactic category of A-lists, whose elements are sorted in descending order according to $<_A$. We can lift the ordering $<_A$ to $<_L$ via a slight generalization of the lexicographic order: the empty list is smaller than all non-empty lists, and two non-empty lists are ordered lexicographically by their heads under $<_A$, followed by their tails under $<_L$. If $<_A$ has order-type α, then $<_L$ has order-type ω^α. If we eliminate the assumption that $<$ is a total ordering, and eliminate the restriction that L's lists must be sorted, then, by treating these unsorted lists as multisets, lifting $<_A$ to $<_L$ via the *multiset ordering* [DM79] has order-type ω^α as well [HK91]. Ordinal exponentiation has some of the basic properties of ordinary exponentiation: $\omega^0 = 1$, $\omega^1 = \omega$ and for every α and β, $\omega^\alpha \cdot \omega^\beta = \omega^{\alpha+\beta}$ and $(\omega^\alpha)^\beta = \omega^{\alpha\cdot\beta}$.

We find it useful to define the iterated exponentiation function ω_n as follows.

$$\omega_0 = 1$$
$$\omega_{n+1} = \omega^{\omega_n}$$

This definition is more or less standard, except that some authors prefer to define ω_0 to be 0 or ω. Regardless of the convention adopted, the function that maps n to ω_n can be used as the fundamental sequence for an ordinal that is usually denoted by the symbol ε_0, which will be discussed in greater detail in subsequent sections.

3.1.2 Ordinal Notation Systems

An ordinal notation system is essentially just a syntactic category coupled with a transitive, total, well-founded ordering, whose terms are intended to give names to ordinals. We do not require that the well-foundedness of such an ordering to be proven outright, although we do require that the totality, transitivity and decidability of the ordering can be proven using syntactically-finitary methods.

For example, we have already seen how the terms of the syntactic categories n, E, and O, coupled with the subterm orderings $<, <_E$ and $<_O$, can be used to give names to the finite ordinals, whereas these notation systems themselves have order type ω. Similarly, the terms in the syntactic category N, coupled with the ordering $<_N$, can give names to both the finite ordinals and ordinals of the form $\omega+n$, whereas the order-type of this notation system is $\omega + \omega$. In general, the order type of any ordinal notation system will be the supremum of all of the ordinals named within the notation system. Note that this means that no ordinal notation system can provide a name for every ordinal: otherwise, the order-type for such a notation system would be strictly larger than itself, leading to a contradiction known as the Burali-Forti paradox. Thus, every ordinal notation system is necessarily incomplete.

The following theorem, due to Georg Cantor, is useful in helping us to define ordinal notation systems.

Theorem 3.1.1 (Cantor Normal Form) *For every non-zero ordinal α, there exist ordinals $\alpha_1 \ldots \alpha_n$ such that $\alpha \geq \alpha_1 \geq \ldots \geq \alpha_n$ and $\alpha = \omega^{\alpha_1} + \ldots + \omega^{\alpha_n}$*

Note that the Cantor Normal Form theorem does not contradict the Burali-Forti paradox; using symbols for 0, ordinal addition, and ordinal exponentiation at base ω, we can define a notation system for any ordinal α whose representation in Cantor normal form (CNF), satisfies $\alpha > \alpha_1 \geq \ldots \geq \alpha_n$ (although the notion of equality over the ordinals in this notation system is more nuanced than just syntactic equality). However, such an ordinal notation system cannot represent any of the ε-*numbers*, that is ordinals that are fixed-points of the equation $\alpha = \omega^\alpha$. In fact, the order-type for this notation system is the smallest ε-number, ε_0. The ordinal ε_0 plays an important role in the proof-theoretic analysis of number theory, described in Section 3.2.1.

However, there is nothing to stop us from adding a symbol for ε_0 to the ordinal notation system described above: the resulting notation system would have order type ε_1, i.e. the next-smallest fixed-point of $\alpha = \omega^\alpha$ after ε_0. Similarly, if we then add a symbol for ε_1, the resulting ordinal notation system would have order-type ε_2, and so forth and so on. Adding all of the symbols ε_n, where n is a finite ordinal, would result in an ordinal notation system whose order-type is ε_ω. Adding a symbol for $\varepsilon_\omega$would have order type $\varepsilon_{\omega+1}$, and so forth and so on. But what about the ordinals that are fixed-points of the equation $\alpha = \varepsilon_\alpha$?

Clearly we can continue this process indefinitely. To this end, we can define the (binary) *Veblen function* $\varphi_\alpha(\beta)$ (sometimes written $\varphi(\alpha, \beta)$) informally, as follows. We define $\varphi_0(\beta)$ to be ω^β and $\varphi_{\alpha+1}$ is the function that iterates over the fixed-points of φ_α. Thus, we can write ε_α as $\varphi_1(\alpha)$. If α is a limit ordinal, then φ_α is function that iterates over the ordinals that are fixed points of all of the functions of the form $\varphi_{\alpha'}$, where $\alpha' < \alpha$. We can, using a symbol for the Veblen function, along with symbols for 0 and ordinal addition, define an ordinal notation system which is much more expressive than anything we have considered thus far. The order-type of this ordinal notation system is referred to as the *Feferman-Schütte ordinal*, and is typically denoted using the symbol Γ_0 (see [Gal91] for a more formal development of the ordinals up to Γ_0). The Veblen function can be generalized to operate on n arguments, for some fixed number $n \geq 2$; the supremum of the order-types of the resulting ordinal-notation systems is referred to as the *small Veblen ordinal* [Mos04]. The small Veblen ordinal is the largest ordinal that we are interested in for the purposes of this dissertation.

3.2 Ordinals in Proof Theory

Given an ordinal notation system $\mathcal{O}$, whose order type is α, the principle of *transfinite induction up to* α is the principle of well-founded induction applied to the well-ordering of $\mathcal{O}$, but not the order-type of $\mathcal{O}$ itself (e.g. the principle of transfinite induction up to ω is just the ordinary principle of mathematical induction). Note that we do not necessarily consider the principle of transfinite induction to be syntactically finitary, but we can consider the effect of extending syntactically finitary reasoning with transfinite induction.

3.2.1 Gentzen's Theorems

This sort of consideration is well precedented. As we have already discussed, it is a consequence of Gödel's second incompleteness theorem that, in general, proofs of logical consistency reduce one set of proof-theoretic assumptions to another. In 1936 Gerhard Gentzen published a consistency proof for a natural-deduction style formulation of classical first-order arithmetic, using a combination of finitary methods and transfinite induction up to ε_0 (using an ordinal-notation system based on lists of numbers) [Gen36]. In 1938, Gentzen presented a refined version of this result, this time for a sequent calculus formulation of first-order arithmetic and using the ordinal notation system for ε_0 based on CNF described in Section 3.1.2 [Gen38].

Both proofs proceed roughly as follows: using finitary methods, any proof of a contradiction ($\vdash 1 = 2$ in [Gen36], $\cdot \vdash \cdot$ in [Gen38]) can be assigned an ordinal measure, and, again using finitary methods, any such proof can be reduced in such a way that this ordinal measure decreases; thus, by transfinite induction, any proof of a contradictory statement can be reduced to a proof of size 0, and it can easily be seen via case-analysis that no such proof exists. Gentzen argued that finitary methods must be considered valid beyond any reasonable doubt, meaning that the consistency of arithmetic can be reduced to believing in the principle of transfinite induction up to ε_0.

Gentzen did not provide a formal system to capture what he meant by "finitary methods," but from his description it is clear that he was referring to a small, constructive subset of the principles formalized by first-order arithmetic. In the intervening years, it has been argued, most prominently by Tait, that the concept of finitism is captured by the quantifier-free theory of primitive recursive arithmetic, PRA [Tai81] (although [Rat06] observes that Gentzen's results can be formalized in terms of elementary recursive arithmetic, which is even weaker than PRA). In general, Gentzen's formulations of classical first-order arithmetic are essentially equivalent to the formal system now referred to as Peano arithmetic (PA), and, every theorem of PRA can be proven in a fragment of Peano arithmetic, PA_1, where induction is restricted to Σ_1-predicates (and, as it turns out, vice-versa [Par66, Min73a]).

Definition 3.2.1 (Fragments of Peano Arithmetic) *The system PA_n is the usual first-order formulation of Peano arithmetic in which the induction schema is restricted to Σ_n (or, equivalently, Π_n) formulas when written in prenex normal form.*

Thus, Gentzen's 1936 and 1938 results can be summarized as follows, where $\mathrm{TI}_1(\varepsilon_0)$ is the principle of transfinite induction up to ε_0 restricted to Σ_1-predicates, and CON(PA) is the encoding of the statement "Peano arithmetic is consistent" within PA.

$$\mathrm{PA}_1 + \mathrm{TI}_1(\varepsilon_0) \vdash \mathrm{CON(PA)}$$

Thus, by Gödel's second incompleteness theorem, if we believe that PA is consistent, then we must also believe the following.

$$\mathrm{PA} \nvdash \mathrm{TI}_1(\varepsilon_0)$$

In 1943, Gentzen proved, again using only finitary methods, that for any ordinal α that is strictly smaller than ε_0, the principle of transfinite induction up to ω^α on formulas of *degree n* can be replaced with the principle of transfinite induction up to α on formulas of degree $n+1$ [Genb] (Gentzen defined the degree of a formula to be the total number of logical connectives, although it has been shown that this result holds even when "degree" is defined to be the number of quantifier-alternations the formula has in prenex normal form; see Theorem 3.2.6). Thus, by induction on the structure of α (as represented in CNF), the principle of transfinite induction up to any fixed ordinal $\alpha < \varepsilon_0$ is an admissible rule in first-order arithmetic. This result can be summarized as follows.

$$\mathrm{PA} \vdash \mathrm{TI}_1(< \varepsilon_0)$$

In other words, by the 1936 and 1938 results, ε_0 is an upper bound on the provable well-orderings of first-order arithmetic, and by the 1943 result, this bound is precise. Thus, the ordinal ε_0 is sometimes referred to as *the proof theoretic ordinal* of arithmetic. The branch of proof theory known as *ordinal analysis* concerns itself primarily with finding the proof theoretic ordinals of various logical theories, where the expressivity of different theories can usually be accurately compared in terms of the size of their proof theoretic ordinals. However, we do have to be careful: the definition of a proof-theoretic ordinal for a given theory depends on the ordinal notation system used, although this is rarely a complication for any *natural*[2] ordinal notation system. [Rat07]

3.2.2 Ordinal Recursive Hierarchies

In Section 1.2, we saw how structural induction can be used to prove the totality of functions such as *add*, and how induction on a lexicographic ordering can be used to prove the totality of functions such as *ack*; the induction orderings used in these proofs have order type ω and ω^2, respectively. In general, the principle of transfinite induction up to α can also be used to prove the totality of functions; we refer to such an application of transfinite induction up to α as *transfinite recursion up to α.*

We define two ordinal-recursive hierarchies of functions below. Recall that we write $\alpha[n]$ for the nth element of the limit ordinal α's fundamental sequence.

Definition 3.2.2 (Fast Growing Hierarchy) *For a given ordinal notation system of order-type α, we define the* fast-growing hierarchy *at F_α to be the class of number-theoretic functions f_β defined by transfinite recursion up to α.*

[2]As with the notion of "consistent logic," we cannot define what it means for an ordinal notation system to be "natural," but we know it when we see it. For example, it is possible to use an ordinal notation system of order type ω^2 to prove the consistency of PA; this ordering would make direct reference to the first order predicate "the Gödel numbering of F is provable in PA" in such a way that the the consistency of PA is directly implied by the well-foundedness of this ordering [Rob65, Rat07]. We do not consider such an ordinal notation system to be natural.

$$\begin{aligned} f_0(n) &= s\,n \\ f_{\beta+1}(n) &= \overbrace{(f_\beta \circ \ldots \circ f_\beta)}^{n \textit{ times}}(n) \\ f_\beta(n) &= f_{\beta[n]}(n) && \textit{if } \beta \textit{ is a limit ordinal} \end{aligned}$$

where $\circ$ denotes function composition, and the ellipses can be eliminated by defining an auxiliary function that is primitive recursive on n.

Definition 3.2.3 (Hardy Hierarchy) *For a given ordinal notation system of order-type α, we define the* Hardy hierarchy *at H_α to be the class of number-theoretic functions h_β defined by transfinite recursion up to α.*

$$\begin{aligned} h_0(n) &= n \\ h_{\beta+1}(n) &= h_\beta(s\,n) \\ h_\beta(n) &= h_{\beta[n]}(n) && \textit{if } \beta \textit{ is a limit ordinal} \end{aligned}$$

Although the definitions of the fast growing and Hardy hierarchies depend on the exact definition of fundamental sequence, for the ordinals up to ε_0 (and, to a lesser extent, Γ_0) the definition of fundamental sequence is standard.

It is worth mentioning that the function that maps n to $ack(n, n)$ (as defined in Section 1.2) grows roughly at the same rate as f_ω and h_{ω^ω} (folklore), and that the Ackermann function "grows faster" than any primitive recursive function (also folklore; see [Sza93] for a syntactically finitary proof) in the following sense.

Definition 3.2.4 *We say that a number-theoretic function f* majorizes *the number-theoretic function g iff there exists some n such that, for every $m > n$, $f(m) > g(m)$.*

The following theorem demonstrates some interesting properties of the Hardy and fast growing hierarchies.

Theorem 3.2.5 *For any $\alpha < \varepsilon_0$*

1. *for any n, $f_\alpha(n) = h_{\omega^\alpha}(n)$*

2. *for any n, m, if $n < m$ then $h_\alpha(n) < h_\alpha(m)$ and $f_\alpha(n) < f_\alpha(m)$*

3. *for any β if $\beta < \alpha$ then h_α majorizes h_β, and f_α majorizes f_β*

Proof: Each is by transfinite induction up to α. See [BW87] for details. $\square$

In Section 1.2, we saw that functions can be defined syntactically by first giving judgmental definitions of their defining equations, then by using case analysis and induction to prove totality. In first-order theories, such as Peano arithmetic, the provably functions are characterized similarly. A function f can be represented by an atomic predicate-symbol P_f, where f's defining equations $f(\mathit{inputs}) = \mathit{outputs}$ are axioms (usually expressible as Horn clauses) for formulas of the form $P_f\langle \mathit{inputs}; \mathit{outputs}\rangle$; in general P_f need not be atomic so long as its prenex normal form is at most Σ_1, and the defining equations for f need not be axioms, so long as $P_f\langle \mathit{inputs}; \mathit{outputs}\rangle$ is provable iff $f(\mathit{inputs}) = \mathit{outputs}$. For example, addition (defined judgmentally in Section 1.1) could be represented in Peano arithmetic by a tertiary atomic predicate symbol *add*, whose axioms would be $\forall x.\forall y.\mathit{add}(z;\ x;\ x)$ and $\forall x_1.\forall x_2.\forall y.\mathit{add}(x_1;\ x_2;\ y) \supset \mathit{add}(s\,x_1;\ x_2;\ s\,y)$; similarly, the function $3 \times x$ could then be represented by the binary predicate $x.y.\exists y'.\mathit{add}(x;\ x;\ y') \wedge \mathit{add}(x;\ y';\ y)$. Given such a representation, we say that f is *provably total* in a first order theory iff the formula $\forall \mathit{inputs}.\exists \mathit{outputs}.P_f\langle \mathit{inputs}; \mathit{outputs}\rangle$ can be proved.

Ordinal recursive hierarchies can sometimes be used to characterize the provably total functions of a given logical theory, where ordinal notation systems can either be defined to be part of the term-algebra for the logic, or else are defined in terms of Gödel-numbering. The following statements provide examples. Recall the definition of ω_n from the end of Section 3.1.1. Note that the principle $\mathrm{TI}_1(\omega)$ is equivalent to the ordinary principle of mathematical induction restricted to Σ_1-formulas (e.g. $\mathrm{PA}_1 + \mathrm{TI}_1(\omega) = \mathrm{PA}_1$).

Theorem 3.2.6 (Ordinal Hierarchies and Fragments of PA)

1. *[BW87]*

   (a) *For every $\alpha < \varepsilon_0$, f_α and h_α are provably total in PA.*

   (b) *If f is a provably total function in PA, then f is majorized by both h_{ε_0} and f_{ε_0}.*

2. *[Min73a, CR91]*

   (a) *For every n, $PA_1 + TI_1(\omega_{n+2}) \vdash CON(PA_{n+2})$*

   (b) *For every n, $PA_{n+1} \vdash TI_1(\omega_{n+1})$*

3. *[CR91]*

   (a) *[Par66] For every $\alpha < \omega_{n+1}$, f_α and h_{ω^α} are provably total in PA_{n+1}.*

   (b) *If f is a provably total function in PA_{n+1}, then f is majorized by both $f_{\omega_{n+1}}$ and $h_{\omega_{n+2}}$.*

4. *[Min73a, CR91] For any $\alpha < \varepsilon_0$, $(PA_{n+2} + TI_1(\alpha))$ can prove the same theorems as $(PA_{n+1} + TI_1(\omega^\alpha))$*

Proof: The proofs are nontrivial. Some of them (e.g. [BW87, CR91]) make essential use of the infinitary inference rule known as the ω-rule, popularized by Kurt Schütte.

$$\frac{\vdash P(0) \quad \vdash P(1) \quad \vdash P(2) \quad \ldots}{\vdash \forall x.P(x)}\ {\scriptstyle \omega\text{-rule}}$$

Although this practice appears on the surface to be outside of the realm of the finitary, infinitary proof-theoretic analyses can, in general, be given (syntactically) finitary presentations [Buc91]. □

The above theorem, combined with the one below, will have interesting consequences with regards to the provability of the Extraction theorem for the assertion logic defined in Section 2.2.

Theorem 3.2.7 (Gödel's T and ε_0)

1. *For every e^τ in Gödel's T, the function that normalizes e^τ is ε_0-recursive.*
2. *For every number-theoretic function f that is provably total in PA, f can be represented as a term in Gödel's T.*

Proof: 1 can be proved by a non-trivial assignment of ordinals smaller than ε_0 to terms [How70, Sch77] (the former is for a formulation of Gödel's T similar to ours, the latter is in terms of combinators). 2 follows from a double-negation translation from classical first-order arithmetic into intuitionistic first-order arithmetic followed by Gödel's Dialectica [Göd58] transformation; see [AF98] for more detail. □

3.2.3 Syntactic Finitism and PA$_2$

In Chapter 1, we argued that the notions of abstract syntax and hypothetical judgment should be granted the same epistemic status by programming languages researchers as the natural numbers are granted by mathematicians. However, strictly speaking, the difference between the former and the latter is a matter of convenience, rather than expressivity. In general, abstract syntax trees and derivations can be represented by natural numbers via primitive recursive Gödel numberings.

Thus, manipulating well-formed terms and derivation trees via case-analysis and the application of inference rules can be formalized in PA_1.

We have taken the perspective that substitutions on higher-order derivations are *a priori* well-defined, or, alternatively, can be proven to be well-defined using syntactically finitary methods. When manipulating the Gödel numberings of hypothetical judgments, we have no choice: substitution must be defined and reasoned about explicitly. Fortunately, substitution is, in general, a primitive recursive operation, and thus can also be formalized in terms of Gödel numbers within PA_1.

In general, we view the concept of induction to be syntactically finitary only when it is used to transform arbitrary terms/derivations of given syntactic categories/judgments into terms and derivations of potentially different forms. In other words, syntactically finitary inductions define functions from tuples of derivations to tuples of derivations. In the language of first-order arithmetic, this would correspond to proving a sentence of the following form.

$$\forall x.x \text{ is a Gödel numbering for } (\mathcal{D}_1, \ldots, \mathcal{D}_n) \supset \exists y.y \text{ is a Gödel numbering for } (\mathcal{E}_1, \ldots, \mathcal{E}_m)$$

In general, such proofs follow by induction on x with an induction hypothesis of the following form, which is classically equivalent to a Σ_1-predicate.

$$x.(x \text{ is a Gödel numbering for } (\mathcal{D}_1, \ldots, \mathcal{D}_n) \supset \exists y.y \text{ is a Gödel numbering for } (\mathcal{E}_1, \ldots, \mathcal{E}_m))$$

Every proof of every theorem in Chapters 1, 2, 4 and 5 follow this template, albeit for potentially different instances of the induction principle. Thus far, we have only allowed syntactically finitary proofs to use induction metrics to be built up from lexicographic orderings[3] applied subterm orderings. Subterm orderings have order-type at most ω, and lexicographic orderings correspond to ordinal multiplication,

[3]Recall that any use of the simultaneous ordering can be replaced with a use of the lexicographic ordering.

meaning that any one syntactically finitary proof may use an induction principle of the form $\mathrm{TI}_1(\omega^n)$, and thus the class of syntactically finitary proofs use an induction principle no stronger than $\mathrm{TI}_1(\omega^\omega)$. This characterizes all of the proofs of all of the theorems in Chapter 2 and Chapter 5, save Theorem 5.2.62, which is modular in the order type of its induction metric. Thus, all such proofs can be formalized in the system $\mathrm{PA}_1 + \mathrm{TI}_1(\omega^\omega)$.

By Theorem 3.2.6, $\mathrm{PA}_1 + \mathrm{TI}_1(\omega^\omega)$ is equivalent to PA_2, which has a proof-theoretic ordinal ω^{ω^ω}. With this in mind, we can use Gödel's second incompleteness theorem along with some of the theorems of Section 3.2.2 to prove the following theorem.

Theorem 3.2.8 (The Outer Limits of Syntactic Finitism) *The following statements are true for the notion of syntactic finitism in which induction is restricted to lexicographic orderings built from subterm orderings.*

1. *Any proof of the consistency of first-order arithmetic is not syntactically finitary.*

2. *The function that realizes any syntactically finitary proof is majorized by $h_{\omega^{\omega^\omega}}$ and f_{ω^ω}.*

3. *Syntactically finitary methods cannot be used to provide a proof of weak normalization for Gödel's T.*

4. *Syntactically finitary methods cannot be used to provide a proof of the Extraction theorem for the assertion logic of Section 2.2.*

Proof: 1 follows directly from Gödel's second incompleteness theorem. 2 is a direct application of Theorem 3.2.6. 3 follows from 2, Theorem 3.2.6 and Theorem 3.2.7. 4 follows from 3 and the Fundamental Theorem and Escape Lemma of Section 2.2. $\Box$

In Chapter 4 we will see how extending the notion of syntactic finitism to a stronger ordering can be used to overcome the limitations enumerated by Theorem 3.2.8.

Chapter 4

Lexicographic Path Induction

Programming languages theory is full of problems that reduce to proving the consistency of a logic. Although the principle of transfinite induction is routinely employed by logicians in proving such theorems, it is rarely used by programming languages researchers, who often prefer alternatives such as proofs by logical relations and model theoretic constructions.

This phenomenon can be explained at least in part by the fact that ordinals can be notoriously tricky to work with, and are usually quite inconvenient to define and manipulate syntactically (this is in large part due to the fact that most ordinal notation systems rely on a notion of equality that it much more sophisticated than syntactic equality). The Burali-Forti paradox illustrates that any ordinal notation system is necessarily incomplete, and in practice, as ordinals get bigger, the notation systems needed to describe them become more complex.

In contrast, the lexicographic path ordering (LPO) is powerful (its order type approaches the small Veblen ordinal[DO88, Mos04]), is well understood by computer scientists and is easy to define and reason about syntactically. Furthermore, it has been used to prove the termination of term rewriting systems (TRSs) for decades, where, ironically, its considerable proof-theoretic strength cannot be fully harnessed. In this chapter, we consider the effect of extending the notion of syntactic finitism to allow induction on the lexicographic path ordering, in much the same way that Gentzen considered the effect of extending the notion of finitism to allow transfinite induction up to ε_0.

The LPO is more than strong enough to prove theorems such as cut elimination for Peano Arithmetic, Heyting Arithmetic, and weak normalization for Gödel's T. Various cut-elimination and normalization procedures can often be expressed as term rewriting systems. Thus, one might hope to prove the weak normalization of Gödel's T, or cut elimination for Heyting Arithmetic using the lexicographic path ordering to show the termination of such a TRS.

However, this is impossible. The length of the reduction sequences in any TRS whose termination can be proven using the LPO are majorized by the Hardy hierarchy at ω^{ω^ω} (or, equivalently, the fast growing hierarchy at ω^ω) [Wei95]; because all of the functions from the Hardy hierarchy (and fast growing hierarchy) at ordinals less than ε_0 can be implemented in Gödel's T, the normalization of Gödel's T cannot be proved by formulating it as a TRS whose termination is shown using the LPO. Moreover, if the rules of a TRS can be shown to be reducing using the LPO, then the resulting termination proof can be modified modified such that it is valid in a fragment of Peano Arithmetic where induction is restricted to Π_2-predicates (i.e. PA_2) [Buc95]. By Gödel's second incompleteness theorem, one cannot prove the consistency of arithmetic from within a fragment of arithmetic, therefore a cut-elimination procedure for arithmetic cannot be shown to be terminating by formulating as a TRS whose rules are reducing according to the LPO.

However, it is possible to harness the strength of the LPO as an induction principle that we call *lexicographic path induction*, which combines the comfort of structural induction with the expressive strength of transfinite induction. In [SS09], a novel consistency proof by lexicographic path induction was given for an intuitionistic theory of inductive definitions, based on the system by Martin-Löf [ML71] that inspired the definition of inductive types in type theory [Dyb91, PM93], and several sequent calculi in the programming languages literature [MM00, MT03, Bro06, GMN08], and can be instantiated to a version of Heyting Arithmetic. Here, we apply the same technique to prove the extraction theorem for the assertion logic described in Section 2.2, thus completing the proof of the weak normalization of Gödel's T. This proof has been formalized in a prototypical extension of Twelf (`http://www.twelf.org/lpo/`) providing empirical evidence for the usefulness of lexicographic path induction.

4.1 The Lexicographic Path Ordering

The lexicographic path ordering (LPO) provides a modular way of specifying orderings on finite labeled trees whose constructors have fixed arity.

Given a finite signature Σ of fixed arity constructors (not to be confused with the notion of signature to be introduced in Chapter 5), whose elements we denote generically by the letters f and g, labeled trees are defined formally as follows

$$\textbf{Labeled Trees} \quad \mathsf{s}, \mathsf{t} \quad ::= \quad \mathsf{f}(\mathsf{s}_1, \ldots, \mathsf{s}_n)$$

where the arity of f, denoted $\#\mathsf{f}$, is n for $n \geq 0$, and $(\mathsf{s}_1, \ldots, \mathsf{s}_n)$ is informal shorthand for a list of length n whose (dependent) BNF-style definition we omit for the sake of readability. We use Σ^n to denote the constructors of Σ of arity n. Although signatures can in principle be infinite, all of the signatures considered in this dissertation are finite. In principle, the labeled trees described by a signature Σ could be defined using a non-dependent, non-hypothetical BNF-style definition.

Definition 4.1.1 (Lexicographic Path Ordering) *Given a precedence relation $<$ on Σ we define $<_{\mathsf{lpo}}$ as the smallest relation on trees that satisfies the following:*

$\mathsf{s} = \mathsf{f}(\mathsf{s}_1, \ldots, \mathsf{s}_n) <_{\mathsf{lpo}} \mathsf{g}(\mathsf{t}_1, \ldots, \mathsf{t}_m) = \mathsf{t}$ *iff at least one of the following holds:*

1. $\mathsf{f} < \mathsf{g}$ *and for all* $i \in 1 \ldots, n$ $\mathsf{s}_i <_{\mathsf{lpo}} \mathsf{t}$

2. $\mathsf{f} = \mathsf{g}$ *and there exists* $k \in 1, \ldots, n$ *s.t. for all* $i < k$ $\mathsf{s}_i = \mathsf{t}_i$, $\mathsf{s}_\mathsf{k} <_{\mathsf{lpo}} \mathsf{t}_\mathsf{k}$ *and for all* $j \in k+1, \ldots, n$ $\mathsf{s}_j <_{\mathsf{lpo}} \mathsf{t}$

3. $\mathsf{s} \leq_{\mathsf{lpo}} \mathsf{t}_i$, *for some* $i \in 1, \ldots, n$

where $\mathsf{s} \leq_{\mathsf{lpo}} \mathsf{t}$ *is shorthand for "*$\mathsf{s} = \mathsf{t}$ *or* $\mathsf{s} <_{\mathsf{lpo}} \mathsf{t}$*." Note that* $<_{\mathsf{lpo}}$ *and* $\leq_{\mathsf{lpo}}$ *can be defined judgmentally (i.e. without the use of ellipses); we omit the inference rules for the sake of brevity and readability.*

We are concerned exclusively with instances of $<_{\mathsf{lpo}}$ where $<$ is transitive and well-founded (and therefore irreflexive). The LPO has several nice properties, including the preservation of transitivity and well-foundedness of $<$. It can be shown in a subsystem of second-order arithmetic that f $<$ is well-founded, then $<_{\mathsf{lpo}}$ is as well [Buc95]. However, for our purposes, we will treat the well-foundedness of $<_{\mathsf{lpo}}$, and thus the validity of Definition 4.1.1, as an *a priori* justified extension to our notion of syntactic finitism.

Lemma 4.1.2 (Properties of LPO)

(Subterm) $\mathsf{t} <_{\mathsf{lpo}} \mathsf{f}(\ldots \mathsf{t} \ldots)$

(Monotonicity) If $\mathsf{s} <_{\mathsf{lpo}} \mathsf{t}$, *then* $\mathsf{f}(\ldots \mathsf{s} \ldots) <_{\mathsf{lpo}} \mathsf{f}(\ldots \mathsf{t} \ldots)$.

(Transitivity) If $<$ *is transitive, then so is* $<_{\mathsf{lpo}}$.

(Big head) If $\mathsf{s} = \mathsf{f}(\mathsf{s}_1, \ldots, \mathsf{s}_n)$*,* $\mathsf{g}_1 < \mathsf{f}, \ldots, \mathsf{g}_m < \mathsf{f}$ *and* t *is built up from* $\mathsf{s}_1, \ldots, \mathsf{s}_n$ *and* $\mathsf{g}_1, \ldots, \mathsf{g}_m$ *then* $\mathsf{t} <_{\mathsf{lpo}} \mathsf{s}$*.*

Proof: The subterm, monotonicity and transitivity properties are shown (via straightforward inductions) in [KL80]. The big head property (or, more precisely, any given instantiation of the Big Head property) can be shown by a straightforward induction on the structure of t. □

Example 4.1.3 *Let* $\Sigma = \{\mathsf{z}, \mathsf{succ}, \mathsf{op}\}$*, where* $\#\mathsf{z} = 0$*,* $\#\mathsf{succ} = 1$ *and* $\#\mathsf{op} = 2$*, and let* $<$ *be defined as* $\mathsf{z} < \{\mathsf{succ}, \mathsf{op}\}$*,* $\mathsf{succ} < \mathsf{op}$*. The following inequalities hold for every* s *and* t

1. $\mathsf{succ}^n(\mathsf{s}) <_{\mathsf{lpo}} \mathsf{op}(\mathsf{s}, \mathsf{t})$ *and* $\mathsf{succ}^n(\mathsf{t}) <_{\mathsf{lpo}} \mathsf{op}(\mathsf{s}, \mathsf{t})$*, for every n*
2. $\mathsf{succ}(\mathsf{op}(\mathsf{s}, \mathsf{t})) <_{\mathsf{lpo}} \mathsf{op}(\mathsf{s}, \succ \mathsf{t})$
3. $\mathsf{op}(\mathsf{s}, \mathsf{op}(\mathsf{s}, \mathsf{op}(\mathsf{s}, \mathsf{t}))) <_{\mathsf{lpo}} \mathsf{op}(\mathsf{succ}\ \mathsf{s}, \mathsf{t})$
4. $\mathsf{op}(\mathsf{s}, \mathsf{op}(\mathsf{succ}\ \mathsf{s}, \mathsf{t})) <_{\mathsf{lpo}} \mathsf{op}(\mathsf{succ}\ \mathsf{s}, \mathsf{succ}\ \mathsf{t})$

The first inequality can be seen as an instance of the big head property. The second inequality highlights another interesting property of the LPO: if a large constructor (in this case op*) occurs beneath a small constructor (in this case* succ*), then "bubbling up" the larger constructor results in a larger term, or viewed the other way, "bubbling up" the smaller constructor results in a smaller term; this observation will play an important role in Lemma 4.2.9. The third inequality highlights the application of the second clause of Definition 4.1.1: in a sense, one can think a partially applied constructor as being a constructor in its own right, where the "precedence" of* $\mathsf{op}(\mathsf{s}, -)$ *is smaller than* $\mathsf{op}(\mathsf{succ}\ \mathsf{s}, -)$*. The last inequality can be used to help show that the Ackermann function, when formulated as a term rewriting system, terminates.*

4.2 The Extraction Proof

We give now the proof of the Extraction Theorem, where we reason about derivations of the form $\mathcal{D} : (\cdot \overset{\text{cut}}{\vdash} hc(e^\tau))$ using lexicographic path induction. Although proof terms are in a sense finite trees, we do not apply lexicographic path induction to them directly because proof terms contain information that we do not consider relevant to the size of a proof. Instead, we apply the principle to *skeletons* of proof trees, which will be defined in 4.2.2; lexicographic path induction on skeletons subsumes structural induction on proof trees.

In many ways, our proof follows the same general structure as Gentzen's later proof of the consistency of arithmetic [Gen38], and the Howard [How70] and Schütte [Sch77] proofs of normalization for Gödel's T. All involve assigning well-founded orderings to proof trees/λ-calculus terms (ϵ_0 for Gentzen, Howard and Schütte, the LPO here) and all unfold inductions all-at-once, rather than one-at-a-time. Like Gentzen's proof, we demonstrate normalization for a restricted class of sequents ($\cdot \overset{\text{cut}}{\vdash} \cdot$ for Gentzen, $\Psi \overset{\text{cut}}{\vdash} hc(e^\tau)$ whenever Ψ `varctx` here). Our proof differs from the others in that lexicographic path induction is applied directly to skeletons of proof terms, whereas in Gentzen's, Howard's and Schütte's proofs, the assignment of ordinals to proofs/λ-calculus terms is very complex. This discrepancy shouldn't be too surprising: the order type of the lexicographic path ordering approaches the small Veblen ordinal [DO88, Mos04], which is ***MUCH*** larger than ϵ_0; it is often the case that using stronger-than-necessary assumptions leads to simpler proofs.

4.2.1 Proof Terms

In order to define our proof, we find it convenient to manipulate proof terms written in BNF-style, as defined below. As we have already seen, the distinction between the two styles of definition is largely a matter of notational preference. As usual, we will omit dependency superscripts when they are easily inferred or irrelevant.

$$
\begin{array}{lcl}
C^F, D^F, E^F & ::= & (\mathit{axiom}\ h^F)^F \mid (\mathit{cut}\ C^F\ (h^F.D^G))^G \mid (\mathit{andr}\ C^F\ D^G)^{F\wedge G} \mid \\
 & & (\mathit{andl1}\ (h^{F_1}.C^G)\ h'^{F_1\wedge F_2})^G \mid (\mathit{andl2}\ (h^{F_2}.C^G)\ h'^{F_1\wedge F_2})^G \mid \\
 & & (\mathit{impr}\ (h^F.C^G))^{F\supset G} \mid (\mathit{impl}\ C^{F_1}\ (h^{F_2}.D^G)\ h'^{F_1\supset F_2})^G \mid \\
 & & (\mathit{allr}\ (x^\tau.C^F))^{\forall x^\tau F} \mid (\mathit{alll}\ (h^{F[t^\tau/x^\tau]}.C^G)\ h'^{\forall x^\tau.F})^G \mid \mathit{hc\text{-}z}^{hc(z^o)} \mid \\
 & & (\mathit{hc\text{-}s}\ C^{hc(e^o)})^{hc(s\ e^o)} \mid (\mathit{hc\text{-}arr}\ (x^\tau.v^{x^\tau}.C^{hc(app\ e^{\tau\Rightarrow\sigma}\ x^\tau)}))^{hc(e^{\tau\Rightarrow\sigma})} \mid \\
 & & (\mathit{hc\text{-}wh}\ C^{wh(e^o,e'^o)}\ D^{hc(e'^o)})^{hc(e^o)} \mid (\mathit{hc\text{-}atm}\ C^{ha(e^o)})^{hc(e^o)} \mid \\
 & & \quad (\mathit{hcl}\quad D_0^{P(z^o)}\ (x^o.h^{P(x^o)}.D_1^{P(s\ x^o)})\ (x^o.h^{ha(x^o)}.D_2^{P(x^o)}) \\
 & & \qquad\qquad (x^o.y^o.h_1^{wh(x^o,y^o)}.h_2^{P(y^o)}.D_3^{P(x^o)})\ (h^{P(e^o)}.E^F)\ h'^{hc(e^o)})^F \mid \\
 & & (\mathit{ha\text{-}var}\ x^\tau\ v^{x^\tau})^{ha(x^\tau)} \mid (\mathit{ha\text{-}app}\ C^{ha^{\sigma\Rightarrow\tau}(e)}\ D^{hc^\sigma(e')})^{ha^\tau(app\ e\ e')} \mid \\
 & & (\mathit{ha\text{-}r}\ C^{hc^{\tau\Rightarrow o\Rightarrow\tau}(e_1)}\ D^{hc^\tau(e_2)}\ E^{ha^o(e_3)})^{ha^\tau(r_\tau\ e_1\ e_2\ e_3)} \mid \\
 & & \mathit{wh\text{-}beta}^{wh(app\ (lam\ x.\,e_1)\ e_2,e_1[e_2/x])} \mid \\
 & & (\mathit{wh\text{-}app}\ C^{wh(e_1,e_1')})^{wh(app\ e_1\ e_2,app\ e_1'\ e_2)} \mid \\
 & & \mathit{wh\text{-}rz}^{wh((r_\tau\ e_1\ e_2\ z^o),e_1)} \mid \\
 & & \mathit{wh\text{-}rs}^{wh((r_\tau\ e_1\ e_2\ (s\ e_3)),(app\ (app\ e_2\ (r_\tau\ e_1\ e_2\ e_3))\ e_3))} \mid \\
 & & (\mathit{wh\text{-}rc}\ C^{wh(e_3,e_3')})^{wh(r_\tau\ e_1\ e_2\ e_3,r_\tau\ e_1\ e_2\ e_3')}
\end{array}
$$

The following definition helps us to summarize the equivalence between the proof terms as written above, and derivation trees.

Definition 4.2.1 (Compatible Proof Terms) *A proof term C^F is* compatible with Ψ *iff every free x^τ, v^{x^τ} and h^F is declared in Ψ. This notion of compatibility can be formalized using a judgment of the form $\Psi \vdash C^F$; we omit the inference rules for the sake of brevity.*

Lemma 4.2.2 *Every derivation $\mathcal{D} : (\Psi \overset{\text{cut}}{\vdash} F)$ is isomorphic to some D^F which is compatible with Ψ, and vice versa.*

Proof: By straightforward induction. $\square$

Lemma 4.2.2 justifies treating proof terms $\mathcal{D} : (\Psi \vdash F)$ and proof derivations that satisfy $\Psi \vdash D^F$ interchangeably, without loss of generality.

The following definition plays an important role in our proof of extraction.

Definition 4.2.3 (Right Normal) *We say that a derivation* $\mathcal{D} : (\Psi \overset{\text{cut}}{\vdash} F)$ *(or the equivalent proof term* D^F*) is* right normal *iff it contains only right-rules. We can formulate the concept of* right normal *judgmentally, but omit its inference rules for the sake of brevity.*

We are interested mostly in right-normal proofs of atomic formulas for the sake of our proof of the Extraction theorem. The following lemma helps explain why we are justified in thinking of the rule *hcl* as being an iterator over (right-normal) proof of $hc(e^\tau)$.

Lemma 4.2.4 (Folding Right-Normal Proofs) *For every right-normal* $\mathcal{C} : (\Psi \overset{\text{cut}}{\vdash} hc(e^\tau))$ *and every* $\mathcal{D}_0 : (\Psi \overset{\text{cut}}{\vdash} P(z^o))$ *and* $\mathcal{D}_1 : (\Psi, x^o, h^{P(x^o)} \overset{\text{cut}}{\vdash} P(s\ x^o))$ *and* $\mathcal{D}_2 : (\Psi, x^o, h^{ha(x^o)} \overset{\text{cut}}{\vdash} P(x^o))$ *and* $\mathcal{D}_3 : (\Psi, x^o, y^o, h_1^{wh(x^o,y^o)}, h_2^{P(y^o)} \overset{\text{cut}}{\vdash} P(x^o))$ *there exists* $\mathcal{E} : (\Psi \vdash P(e^\tau))$ *such that* $\mathcal{E}$ *is built up using only instances of* $\mathcal{D}_0$*,* $\mathcal{D}_1$*,* $\mathcal{D}_2$ *and* $\mathcal{D}_3$ *under substitutions, and the cut rule.*

Proof: By straightforward induction on $\mathcal{C}$. The proof can be realized by a function on proof terms, which can be expressed using the following judgment.

$$\texttt{fold-hc}\ C\ D_0\ (x.h.D_1)\ (x.h.D_2)\ (x.y.h_1.h_2.D_3) = E$$

We define the inference rules for `fold-hc` below.

$$\frac{}{\texttt{fold-hc}\ \textit{hc-z}\ D_0\ (x.h.D_1)\ (x.h.D_2)\ (x.y.h_1.h_2.D_3) = D_0}$$

$$\frac{\texttt{fold-hc}\ C\ D_0\ (x.h.D_1)\ (x.h.D_2)\ (x.y.h_1.h_2.D_3) = E}{\texttt{fold-hc}\ (\textit{hc-s}\ C^{hc(e)})\ D_0\ (x.h.D_1)\ (x.h.D_2)\ (x.y.h_1.h_2.D_3) = \textit{cut}\ E\ (h.D_1[e/x])}$$

$$\frac{}{\texttt{fold-hc}\ (\textit{hc-atm}\ C^{ha(e)})\ D_0\ (x.h.D_1)\ (x.h.D_2)\ (x.y.h_1.h_2.D_3) = \textit{cut}\ C^{ha(e)}\ (h.D_2[e/x])}$$

$$\frac{\texttt{fold-hc}\ C_1^{hc(e')}\ D_0\ (x.h.D_1)\ (x.h.D_2)\ (x.y.h_1.h_2.D_3) = E}{\begin{array}{c}\texttt{fold-hc}\ (\textit{hc-wh}\ C_0^{wh(e,e')}\ C_1^{hce'})\ D_0\ (x.h.D_1)\ (x.h.D_2)\ (x.y.h_1.h_2.D_3)\\ = \textit{cut}\ C_0^{wh(e,e')}\ (h_0.\textit{cut}\ E\ (h_1.D_3[e/x][e'/y]))\end{array}}$$

The proof that `fold-hc` is a function (i.e. its *effectiveness lemma*) has essentially the same structure as the proof that `fold-hc` is meant to represent. □

4.2.2 Ordering Proof Terms

Although proof terms contain the same amount of information as proof trees, we do not consider all of this information to be relevant to the size of proofs. In particular, we do not consider the "size" of hypotheses or λ-calculus terms to be relevant, nor, with the notable exception of *cut*, do we consider formulas or predicates relevant. We also do not consider dependency information to be relevant to the size of proof terms. Therefore, we map proof terms into labeled trees, called *skeletons*, which are obtained from proof terms by *stripping* spurious information. Because we consider the size of cut-formulas to be relevant to the size of proofs (as is often the case in cut-elimination-like theorems), skeletons are defined for formulas as well. The signature Σ for skeletons is defined as follows. Note the use of the sans-serif font to distinguish skeletons from proof terms.

$$\begin{array}{lcl}
\Sigma^0 & = & \mathsf{hc}, \mathsf{ha}, \mathsf{wh}, \mathsf{axiom}, \mathsf{hc\text{-}z}, \mathsf{ha\text{-}var}, \mathsf{wh\text{-}beta}, \mathsf{wh\text{-}rz}, \mathsf{wh\text{-}rs} \\
\Sigma^1 & = & \mathsf{all}, \mathsf{andl1}, \mathsf{andl2}, \mathsf{impr}, \mathsf{allr}, \mathsf{alll}, \mathsf{hc\text{-}s}, \mathsf{hc\text{-}arr}, \mathsf{hc\text{-}atm}, \mathsf{wh\text{-}app}, \mathsf{wh\text{-}rc} \\
\Sigma^2 & = & \mathsf{and}, \mathsf{imp}, \mathsf{andr}, \mathsf{impl}, \mathsf{hc\text{-}wh}, \mathsf{ha\text{-}app} \\
\Sigma^3 & = & \mathsf{cut}, \mathsf{ha\text{-}r} \\
\Sigma^5 & = & \mathsf{hcl}
\end{array}$$

We define the stripping functions $[\![F]\!] = \mathsf{s}$ and $[\![C^F]\!] = \mathsf{s}$ below. As usual, we view the system of equations as shorthand for a judgmental definition, whose interpretation as a deterministic function follows from a straightforward induction.

$$
\begin{aligned}
\llbracket hc(e^\tau) \rrbracket &= \mathsf{hc} & \llbracket ha(e^\tau) \rrbracket &= \mathsf{ha} \\
\llbracket wh(e^\tau, e'^\tau) \rrbracket &= \mathsf{wh} & \llbracket F \wedge G \rrbracket &= \mathsf{and}(\llbracket \mathsf{F} \rrbracket, \llbracket \mathsf{G} \rrbracket) \\
\llbracket F \supset G \rrbracket &= \mathsf{imp}(\llbracket \mathsf{F} \rrbracket, \llbracket \mathsf{G} \rrbracket) & \llbracket \forall x^\tau . F \rrbracket &= \mathsf{all}(\llbracket F \rrbracket) \\
\llbracket \mathit{axiom}\ h \rrbracket &= \mathsf{axiom} & \llbracket \mathit{cut}\ C^F\ (h^F.D) \rrbracket &= \mathsf{cut}(\llbracket F \rrbracket, \llbracket C \rrbracket, \llbracket D \rrbracket) \\
\llbracket \mathit{andr}\ C\ D \rrbracket &= \mathsf{andr}(\llbracket C \rrbracket, \llbracket D \rrbracket) & \llbracket \mathit{andl1}\ (h.C)\ h' \rrbracket &= \mathsf{andl1}(\llbracket C \rrbracket) \\
\llbracket \mathit{andl2}\ (h.C)\ h' \rrbracket &= \mathsf{andl2}(\llbracket C \rrbracket) & \llbracket \mathit{impr}\ (h.C) \rrbracket &= \mathsf{impr}(\llbracket C \rrbracket) \\
\llbracket \mathit{impl}\ C\ (h.D)\ h' \rrbracket &= \mathsf{impl}(\llbracket C \rrbracket, \llbracket D \rrbracket) & \llbracket \mathit{allr}\ x^\tau . C \rrbracket &= \mathsf{allr}(\llbracket C \rrbracket) \\
\llbracket \mathit{alll}\ (h.C)\ h' \rrbracket &= \mathsf{alll}(\llbracket C \rrbracket) & \llbracket \mathit{hc\text{-}z} \rrbracket &= \mathsf{hc\text{-}z} \\
\llbracket \mathit{hc\text{-}s}\ C \rrbracket &= \mathsf{hc\text{-}s}(\llbracket C \rrbracket) & \llbracket \mathit{hc\text{-}arr} x.v.C \rrbracket &= \mathsf{hc\text{-}arr}(\llbracket C \rrbracket) \\
\llbracket \mathit{hc\text{-}wh}\ C\ D \rrbracket &= \mathsf{hc\text{-}wh}(\llbracket C \rrbracket, \llbracket D \rrbracket) & \llbracket \mathit{hc\text{-}atm}\ C \rrbracket &= \mathsf{hc\text{-}atm}(\llbracket C \rrbracket) \\
\llbracket \mathit{ha\text{-}var}\ x\ v \rrbracket &= \mathsf{ha\text{-}var} & \llbracket \mathit{ha\text{-}app}\ C\ D \rrbracket &= \mathsf{ha\text{-}app}(\llbracket C \rrbracket, \llbracket D \rrbracket) \\
\llbracket \mathit{ha\text{-}r}\ C\ D\ E \rrbracket &= \mathsf{ha\text{-}r}(\llbracket C \rrbracket, \llbracket D \rrbracket, \llbracket E \rrbracket) & \llbracket \mathit{wh\text{-}beta} \rrbracket &= \mathsf{wh\text{-}beta} \\
\llbracket \mathit{wh\text{-}app}\ C \rrbracket &= \mathsf{wh\text{-}app}(\llbracket C \rrbracket) & \llbracket \mathit{wh\text{-}rz} \rrbracket &= \mathsf{wh\text{-}rz} \\
\llbracket \mathit{wh\text{-}rs} \rrbracket &= \mathsf{wh\text{-}rs} & \llbracket \mathit{wh\text{-}rc}\ C \rrbracket &= \mathsf{wh\text{-}rc}(\llbracket C \rrbracket)
\end{aligned}
$$

$$\llbracket \mathit{hcl}\ D_0\ (x.h.D_1)\ (x.h.D_2)\ (x.y.h_1.h_2.D_3)\ (h.E)\ h' \rrbracket = \mathsf{hcl}(\llbracket D_0 \rrbracket, \llbracket D_1 \rrbracket, \llbracket D_2 \rrbracket, \llbracket D_3 \rrbracket, \llbracket E \rrbracket)$$

We define the precedence ordering on skeletons as follows, which will then be lifted to an ordering on finite trees via the lexicographic path ordering. Note that we adopt the standard distinction between *atomic* and *compound* formulas, which will play an important role in our proof; to eliminate the possibility of any ambiguity, we formalize this notion using the judgments below.

$$\frac{}{hc^\tau(e)\ \texttt{atomic}} \qquad \frac{}{ha^\tau(e)\ \texttt{atomic}} \qquad \frac{}{wh^\tau(e,e')\ \texttt{atomic}}$$

$$\frac{}{F \wedge G\ \texttt{compound}} \qquad \frac{}{F \supset G\ \texttt{compound}} \qquad \frac{}{\forall x^\tau.F\ \texttt{compound}}$$

Definition 4.2.5 (Skeleton Ordering) *We define $<$ as the least* transitive *ordering on the elements of Σ satisfying all of the following:*

1. *If* f *corresponds to an atomic formula (i.e. it is one of* hc*,* ha *or* wh*) and* g *corresponds to a logical connective (i.e. it is of the form* and*,* imp *or* all*) then* $\mathsf{f} < \mathsf{g}$
2. *If* f *corresponds to a formula, and* g *corresponds to a proof rule, then* $\mathsf{f} < \mathsf{g}$
3. *If* f *corresponds to a right-rule or a compound left-rule then* $\mathsf{f} < \mathsf{cut}$
4. $\mathsf{cut} < \mathsf{hcl}$
5. *If* f *corresponds to an atomic right-rule, then* $\mathsf{f} < \mathsf{axiom}$

$<_{\mathsf{lpo}}$ *is the lifting of $<$ to skeletons via the LPO. Note that $<$ and $<_{\mathsf{lpo}}$ can both be defined judgmentally. We omit the explicit definition of their inference rules for the sake of brevity.*

The first two clauses of Definition 4.2.5 are motivated by the first two clauses of Lemma 4.2.6, which will be used by Lemma 4.2.9; the third clause is motivated by Lemma 4.2.9 as well. The last two clauses are motivated by Lemma 4.2.7.

Lemma 4.2.6 (Properties of Skeletons)

1. *For every* atomic formula F *and every* compound formula G*,* $[\![\mathsf{F}]\!] <_{\mathsf{lpo}} [\![\mathsf{G}]\!]$
2. *For every F and every C,* $[\![\mathsf{F}]\!] <_{\mathsf{lpo}} [\![\mathsf{C}]\!]$

3. For every C, every t and every x, $[\![C]\!] = [\![C[t/x]]\!]$

Proof: Each case is straightforward by structural induction; both 1 and 2 are instances of the big head principle. □

Because the LPO has the subterm property, and because skeletons contain most of the structural information of proof terms, all of the instances of structural induction on C in this chapter can be replaced by lexicographic path induction on $[\![C]\!]$.

4.2.3 The Normalization Procedure

Our proof is structured as follows. Consider $\Psi \vdash C^F$, where Ψ is a variable context (see Definition 2.1.1) and F is an atomic formula: our goal is to find a right normal form for C^F. In this situation, C^F must either be an atomic right rule applied to subderivations $C_0^{F_0}, \ldots, C_n^{F_n}$, where each F_i is also atomic, or C^F is of the form *cut* $D^G\ h^G.E^F$. In the former case, we right-normalize $C_1^{F_1}, \ldots, C_n^{F_n}$ and apply the same atomic right rule to the result. In the latter case, we must find a proof term C'^F which is smaller than C^F, and right-normalize C'^F by induction. In this case, the calculation of C'^F depends on whether the cut-formula G is atomic or compound. Observe that, if G is atomic, then, by induction, we can right-normalize D^G into D'^G; C' is obtained by eliminating all uses of h^G from $h^G.E^F$, making use of D'^G and Lemma 4.2.4. If G is compound, we perform what is essentially a small-step version of the cut-admissibility proof in [Pfe95], where, for reasons that will be explained later, we must be careful to avoid any "commutative conversions" for *hcl*.

In the following lemmas, note that applying the structural properties such weakening and exchange (see Lemma 2.2.2) to a derivation of $\Psi \vdash C^F$ will result in a derivation of $\Psi' \vdash C^F$, where the underlying structure of C^F is unchanged.

Lemma 4.2.7 (Atomic Cut Reduction) *For every C^F and $h^F.D^G$, if F is atomic and C^F is right-normal and $\Psi \vdash C^F$ and $\Psi, h^F \vdash D^G$, then there exists E^G such that $\Psi \vdash E^G$ and $[\![E^G]\!] \leq_{\mathsf{lpo}} [\![D^G]\!]$.*

Proof: By structural induction on D^G. Most cases are straightforward, often using weakening and exchange before applying the induction hypothesis, and using monotonicity of $<_{\mathsf{lpo}}$ on the result. The non-trivial uses of h^F come from *axiom* and *hcl*. If $D^G =$ *axiom* h^F (in which case $F = G$), we return C^F, which is smaller than *axiom* h^F by the big head principle and the fifth clause of Definition 4.2.5. If D^F is a non-trivial instance of *hcl*, we induct on D^F's subterms and apply Lemma 4.2.4, whose output is smaller than D^F by the third clause of Lemma 4.2.6, the big head principle and the fourth clause of Definition 4.2.5. The proof can be realized by a function on proof terms, which can be expressed using the following judgment.

$$\texttt{redA}\ C^F\ (h^F.D^G) = E^G$$

We list some representative cases of `redA`'s inference rules below.

$$\frac{}{\texttt{redA}\ C^F\ (h^F.\mathit{axiom}\ h^F)\ = C^F} \qquad \frac{}{\texttt{redA}\ C^F\ (h^F.\mathit{axiom}\ h'^G) = \mathit{axiom}\ h'^G}$$

$$\frac{\begin{array}{c}\texttt{redA}\ C^{hc(e^\tau)}\ (h^{hc(e^\tau)}.D_0) = D_0' \quad \texttt{redA}\ C^{hc(e^\tau)}\ (h^{hc(e^\tau)}.D_1) = D_1' \\ \texttt{redA}\ C^{hc(e^\tau)}\ (h^{hc(e^\tau)}.D_2) = D_2' \quad \texttt{redA}\ C^{hc(e^\tau)}\ (h^{hc(e^\tau)}.D_3) = D_3' \\ \texttt{redA}\ C^{hc(e^\tau)}\ (h^{hc(e^\tau)}.D_4^G) = D_4'^G \\ \texttt{fold-hc}\ C^{hc(e^\tau)}\ D_0\ (x.h'.D_1')\ (x.h'.D_2')\ (x.y.h_1.h_2.D_3') = E^{P(e^\tau)}\end{array}}{\begin{array}{c}\texttt{redA}\ C^{hc(e^\tau)}\ (h^{hc(e^\tau)}.\mathit{hcl}\ D_0\ (x.h'.D_1)\ (x.h'.D_2)\ (x.y.h_1.h_2.D_3)\ (h'^{P(e^\tau)}.D_4^G)\ h^{hc(e^\tau)}) \\ = \mathit{cut}\ E^{P(e^\tau)}\ (h'^{P(e^\tau)}.D_4'^G)\end{array}}$$

$$\frac{\begin{array}{c}\texttt{redA}\ C^F\ (h^F.D_0) = D_0' \quad \texttt{redA}\ C^F\ (h^F.D_1) = D_1' \\ \texttt{redA}\ C^F\ (h^F.D_2) = D_2' \quad \texttt{redA}\ C^F\ (h^F.D_3) = D_3' \\ \texttt{redA}\ C^F\ (h^F.D_4^G) = D_4'^G\end{array}}{\begin{array}{c}\texttt{redA}\ C^F\ (h^F.\mathit{hcl}\ D_0\ (x.h'.D_1)\ (x.h'.D_2)\ (x.y.h_1.h_2.D_3)\ (h'^{P(e^\tau)}.D_4^G)\ h''^{hc(e^\tau)}) \\ = \mathit{hcl}\ D_0'\ (x.h'.D_1')\ (x.h'.D_2')\ (x.y.h_1.h_2.D_3')\ (h'^{P(e^\tau)}.D_4'^G)\ h''^{hc(e^\tau)}\end{array}}$$

$\square$

The following definition will play an important role in showing that compound cuts can be reduced: because we will only need to reduce such proof terms in contexts that only contain compound formulas, we will not be forced to consider so-called *left-commutative conversions* on *hcl*. The significance of this fact will be expounded upon in the proof of Lemma 4.2.9.

Definition 4.2.8 (Compound Context) *A context* Ψ *is* compound *iff, for every* $h^F \in \Psi$, F *is a compound formula. This can be formalized judgmentally using the inference rules below.*

$$\frac{}{\cdot\ \texttt{compound}} \qquad \frac{}{\Psi, x^\tau\ \texttt{compound}} \qquad \frac{\Psi\ \texttt{compound}}{\Psi, x^\tau\ \texttt{compound}}$$

$$\frac{\Psi\ \texttt{compound}}{\Psi, v^{x^\tau}\ \texttt{compound}} \qquad \frac{\Psi\ \texttt{compound} \quad F\ \texttt{compound}}{\Psi, h^F\ \texttt{compound}}$$

Lemma 4.2.9 (Compound Cut Reduction) *If* Ψ `compound` *and* $\Psi \vdash \mathit{cut}\, C^F\ (h^F.D^G)$ *then there exists an* E^G *such that* $\Psi \vdash E^G$ *and* $[\![E^F]\!] <_{\mathsf{lpo}} [\![\mathit{cut}\ C^F\ (h^F.D^G)]\!]$.

Proof: By induction on the structure of $\mathit{cut}\, C^F\ (h^F.D^G)$. The proof can be realized by a function on proof terms, which can be expressed using the following judgment.

$$\texttt{redC}\ (\mathit{cut}\, C^F\ (h^F.D^G)) = E$$

We list some representative cases below; most use Lemma 4.2.6, clause 2.

If C^F is a left rule, or if D^G is either right rule or left rule that acts on a hypothesis other than h^F, then the *cut* is "commutative," and the offending rule will be bubbled up (see Definition 4.2.5, clause 3). Note that the restriction on Ψ means that we never encounter commutative cuts of atomic left rules, and thus will never have to bubble one past a *cut*. This is critical, because as we have seen in Lemma 4.2.7, cut must be smaller than hcl.

$$\frac{}{\texttt{redC}\ (\mathit{cut}\ C\ (h.\mathit{andr} D_1 D_2)) = \mathit{andr}\ (\mathit{cut}\ C\ h.D_1)\ (\mathit{cut}\ C\ h.D_2)}$$

$$\frac{}{\texttt{redC}\ (\mathit{cut}\ (\mathit{andl2}\ (h'.C)\ h'')\ (h.D)) = \mathit{andl2}\ (h'.\mathit{cut}\ C\ h.D)\ h''}$$

$$\frac{}{\texttt{redC}\ (\mathit{cut}\ C\ (h.\mathit{andl2}\ (h'.D)\ h'')) = \mathit{andl2}\ (h'.\mathit{cut}\ C\ h.D)\ h''}$$

If C^F is a right rule, and D^G is a left rule that acts on h^F, then the cut is "essential." The sizes of cut-formulas play a crucial role in these cases. The $\forall$ essential case uses Lemma 4.2.6, clause 3.

$$\frac{}{\begin{array}{c}\texttt{redC}\ (\mathit{cut}\ (\mathit{impr}\ h_0^F.C_0^G)\ (h^{F\supset G}.\mathit{impl}\ D_0\ (h_1^G.D_1)\ h^{F\supset G}))\ = \\ \mathit{cut}\ (\mathit{cut}\ (\mathit{cut}\ (\mathit{impr}\ h_0^F.C_0)\ h^{F\supset G}.D_0)\ (h_0^F.C_0))\ (h_1^G.\mathit{cut}\ (\mathit{impr}\ h_0^F.C_0)\ h^{F\supset G}.D_1)\end{array}}$$

$$\frac{}{\begin{array}{c}\texttt{redC}\ (\mathit{cut}\ (\mathit{allr}\ x.C)\ (h^{\forall x.F}.\mathit{alll}\ (h'^{F[e/x]}.D)\ h^{\forall x.F}))\ = \\ \mathit{cut}\ (C[e/x])\ (h'^{F[e/x]}.\mathit{cut}\ (\mathit{allr}\ x.C)\ h^{\forall x.F}.D)\end{array}}$$

If C^F or D^G is a compound cut, we apply the induction hypothesis and monotonicity of $<_{\mathsf{lpo}}$; if either is an atomic cut, we bubble it up (see Lemma 4.2.6, clause 1).

$$\frac{\texttt{redC}\ (cut\ C_0^{F\wedge G}\ h'^{F\wedge G}.C_1) = C'}{\texttt{redC}\ (cut\ (cut\ C_0^{F\wedge G}\ h'^{F\wedge G}.C_1)\ D) = cut\ C'\ h.D}$$

$$\frac{\texttt{redC}\ (cut\ D_0^{F\supset G}\ h'^{F\supset G}.D_1) = D'}{\texttt{redC}\ (cut\ C\ (h.cut\ D_0^{F\supset G}\ h'^{F\supset G}.D_1)) = cut\ C\ h.D'}$$

$$\frac{}{\texttt{redC}\ (cut\ (cut\ C_0^{hc(e^\tau)}\ h'^{hc(e^\tau)}.C_1)\ h.D) = cut\ C_0^{hc(e^\tau)}\ (h'^{hc(e^\tau)}.cut\ C_1\ h.D)}$$

$$\frac{}{\texttt{redC}\ (cutC(h.cutD_0^{hc(e^\tau)}(h'^{hc(e^\tau)}.D_1))) = cut\ (cut\ C\ h.D_0^{hc(e^\tau)})\ (h'^{hc(e^\tau)}.cut\ C\ h.D_1)}$$

If C^F or D^F is an application of *axiom*, then we return the obvious derivation, which will be smaller by the monotonicity of $<_{\mathsf{lpo}}$.

$$\frac{}{\texttt{redC}\ (cut\ (axiom\ h')\ h.D) = axiom\ h'} \qquad \frac{h \neq h'}{\texttt{redC}\ (cut\ C\ (h.axiom\ h')) = axiom\ h'}$$

$$\frac{}{\texttt{redC}\ (cut\ C\ (h.axiom\ h)) = C}$$

□

The following proposition will be needed to use the Compound Cut Reduction lemma in the proof of the Extraction theorem.

Proposition 4.2.10 (World Subsumption) *For all* Ψ*, if* Ψ `varctx` *then* Ψ `compound`

Proof: By straightforward induction on the given derivation. □

The following Right Normalization lemma plays an analogous role to the Cut Elimination theorem of Chapter 2. Although the proof of the theorem here is quite different, the result can be seen as a special case of Cut Elimination for atomic formulas.

Theorem 4.2.11 (Right Normalization) *For all C^F, if F is atomic $\Psi \vdash C^F$ and Ψ* `varctx` *then there exists a right normal D^F such that $\Psi \vdash D^F$.*

Proof: By lexicographic path induction on $[\![C^F]\!]$. The extraction theorem can be realized by a function on proof terms, which can be expressed as the following judgment.

$$\texttt{extract}\ C^F = D^F$$

We give some representative cases of the inference rules for `extract` below.

$$\frac{F\ \texttt{compound}\quad \texttt{redC}\ (\mathit{cut}\ C^F\ (h^F.D)) = E\quad \texttt{extract}\ E = E'}{\texttt{extract}\ (\mathit{cut}\ C^F(h^F.D)) = E'}$$

$$\frac{F\ \texttt{atomic}\quad \texttt{redA}\ C^F\ (h^F.D) = E\quad \texttt{extract}\ E = E'}{\texttt{extract}\ (\mathit{cut}\ C^F(h^F.D)) = E'}$$

$$\frac{\texttt{extract}\ C = C'\quad \texttt{extract}\ D = D'}{\texttt{extract}\ (\mathit{hc\text{-}wh}\ C\ D) = (\mathit{hc\text{-}wh}\ C'\ D')}\qquad \frac{\texttt{extract}\ C = D}{\texttt{extract}\ (\mathit{hc\text{-}arr}\ x.v.C) = (\mathit{hc\text{-}arr}\ x.v.D)}$$

$$\frac{\texttt{extract}\ C_1 = C_1'\quad \texttt{extract}\ C_2 = C_2'\quad \texttt{extract}\ C_3 = C_3'}{\texttt{extract}\ (\mathit{ha\text{-}r}\ C_1\ C_2\ C_3) = (\mathit{ha\text{-}r}\ C_1'\ C_2'\ C_3')}$$

□

The following theorem is analogous to Lemma 2.1.2.

Lemma 4.2.12 (Right-Normal Extraction) *For all Ψ, if Ψ is a variable context then:*

1. *If $D^{hc(e))}$ is right-normal and $\Psi \vdash D^{hc(e))}$ then there exists e' such that $e \longrightarrow^* e'$ and $\mathcal{E} : (e' \Uparrow)$ and $\mathcal{E}$ is compatible with Ψ*

2. *If $D^{ha(e))}$ is right-normal and $\Psi \vdash D^{ha(e))}$ then there exists e' such that $e \longrightarrow^* e'$ and $\mathcal{E} : (e' \Downarrow)$ and $\mathcal{E}$ is compatible with Ψ*

3. *If $\mathcal{D}^{wh^\tau(e_1,e_2)}$ is right-normal and $\Psi \vdash \mathcal{D}^{wh^\tau(e_1,e_2)}$ then there exists $\mathcal{E} : (e_1 \longrightarrow e_2)$ and $\mathcal{E}$ is compatible with Ψ*

Proof: By induction on the structure of the given proof terms, using Theorem 2.2.1.
□

We are now ready to prove extraction for Gödel's T, thus filling the last piece of the puzzle in the normalization theorem begun in Section 2.2.

Theorem 4.2.13 (Extraction for Gödel's T) *For all Ψ, if Ψ is a variable context then:*

1. *If $\Psi \vdash D^{hc(e))}$ then there exists e' such that $e \longrightarrow^* e'$ and $\mathcal{E} : (e' \Uparrow)$ and $\mathcal{E}$ is compatible with Ψ*
2. *If $\Psi \vdash D^{ha(e))}$ then there exists e' such that $e \longrightarrow^* e'$ and $\mathcal{E} : (e' \Downarrow)$ and $\mathcal{E}$ is compatible with Ψ*
3. *If $\Psi \vdash \mathcal{D}^{wh^{\tau}(e_1,e_2)}$ then there exists $\mathcal{E} : (e_1 \longrightarrow e_2)$ and $\mathcal{E}$ is compatible with Ψ*

Proof: By Theorem 4.2.11 and Lemma 4.2.12 □

Chapter 5

Proofs as Logic Programs

In Section 3.2.3, we saw that any syntactically finitary proof can be represented as a number-theoretic function that is provably total in a fragment of Peano Arithmetic. Although this characterization is useful for calculating bounds on the expressivity of syntactic finitism—fragments of first-order arithmetic have been well studied by proof theorists—it is not a very satisfying formalization. This is partly due to the fact that Peano Arithmetic is a classical theory[1], although in principle this complaint can be ameliorated by noting that the provably total functions of Heyting and Peano arithmetic are one and the same. More fundamentally, Gödel numbering is simply too clumsy a tool to represent basic syntactic manipulations with the gracefulness they deserve.

In this chapter, we aim to give a more syntactic formalization of the nature of syntactically finitary proofs. We begin with the aforementioned observation: syntactically finitary proofs are all essentially first-order functions from tuples of derivations to tuples of derivations. The defining equations for such functions can be represented

[1]Syntactic finitism is intended characterize a lower bound on the reasoning principles generally accepted by programming languages researchers, many of whom consider the law of excluded middle to be controversial.

judgmentally in the same manner as any other function; we have already seen several examples in Chapter 4. Recall that we consider a judgment $J\langle \mathit{inputs}; \mathit{outputs}\rangle$—where *inputs* and *outputs* are names for the syntactic categories that J depends on—to represent a function from *inputs* to *outputs* if, and only if, it passes the following test: given concrete terms t_{in} from the syntactic categories of *inputs*, we can always find a concrete terms t_{out} from the syntactic categories of *outputs* and a derivation of $\mathcal{D} : J\langle t_{in}; t_{out}\rangle$. One especially natural way to find t_{out} and $\mathcal{D}$ given t_{in} is to perform a depth-first search on derivations.

For example, given the inputs $s\,z$ and $s\,z$ for the judgment *ack* defined in Section 1.2, we might represent the search problem "can we find an n such that $\mathit{ack}(s\,z;\ s\,z;\ n)$ is inhabited?" using the following notation.

$$\boxed{\mathcal{D}} : \mathit{ack}(s\,z;\ s\,z;\ \boxed{n})$$

Intuitively, boxes are intended to represent holes in the derivation, where depth-first search is used to fill them. We can model these holes using hypothetical variables that we refer to as *logic variables.*

Of the inference rules for *ack*, only the rule *ackss* can possibly be used to fill a hole of the above form. We proceed by filling in $\boxed{\mathcal{D}}$ with *ackss*, resulting in a derivation with some new holes.

$$\dfrac{\boxed{\mathcal{D}_0} : (\mathit{ack}(s\,z;\ z;\ \boxed{m})) \quad \boxed{\mathcal{D}_1} : (\mathit{ack}(z;\ \boxed{m};\ \boxed{n}))}{\mathit{ack}(s\,z;\ s\,z;\ \boxed{n})}\ \scriptstyle{ackss}$$

We proceed by considering the left-most derivation hole $\boxed{\mathcal{D}_0}$. Here, the only eligible rule is *acksz*; using it results in the search state represented below. In this particular situation, we could have just as easily proceeded by filling $\boxed{\mathcal{D}_1}$, but we consider doing so to be ill-advised because one of its input arguments is a logic variable. In general, we only require search to be well-behaved when all of the input arguments to a judgment are *ground* terms (i.e. they do not contain logic variables); this will be remarked upon further in Section 5.2.4. In the case of *ack*, if search follows the strategy of always considering the left-most derivation hole, then that hole's input arguments are guaranteed to be ground. We say that judgments with this property are *well moded*, the significance of which will be discussed subsequently.

$$\dfrac{\dfrac{\boxed{\mathcal{D}_0'} : (ack(z;\ s\,z;\ \boxed{m}))}{ack(s\,z;\ z;\ \boxed{m})}\ \textit{acksz} \qquad \boxed{\mathcal{D}_1} : (ack(z;\ \boxed{m};\ \boxed{n}))}{ack(s\,z;\ s\,z;\ \boxed{n})}\ \textit{ackss}$$

At this point, we again attempt to fill the leftmost derivation hole, this time using the inference rule *ackz*. In doing so, we find a concrete instantiation for the logic variable m by *unifying* it with $s\,s\,z$.

$$\dfrac{\dfrac{\dfrac{}{ack(z;\ s\,z;\ s\,s\,z)}\ \textit{ackz}}{ack(s\,z;\ z;\ s\,s\,z)}\ \textit{acksz} \qquad \boxed{\mathcal{D}_1} : (ack(z;\ s\,s\,z;\ \boxed{n}))}{ack(s\,z;\ s\,z;\ \boxed{n})}\ \textit{ackss}$$

We are now able to fill the remaining subgoal, whose input arguments have been grounded by the substitution of $s\,s\,z$ for m. We fill this goal with the only rule that can be applied, *ackz*, thus completing our search.

$$\dfrac{\dfrac{\dfrac{}{ack(z;\ s\,z;\ s\,s\,z)}\ \textit{ackz}}{ack(s\,z;\ z;\ s\,s\,z)}\ \textit{acksz} \qquad \dfrac{}{ack(z;\ s\,s\,z;\ s\,s\,s\,z)}\ \textit{ackz}}{ack(s\,z;\ s\,z;\ s\,s\,s\,z)}\ \textit{ackss}$$

Clearly, the search procedure sketched above, when generalized to work on arbitrary judgments, can be used to implement any computable function; for example, performing search on a judgmentally-specified big-step structural operational semantics is a particularly natural way to define an interpretor. We refer to this computational paradigm as *logic programming*, where judgments such as *ack* are *logic programs* and search problems such as $\boxed{\mathcal{D}} : ack(s\,z;\ s\,z;\ \boxed{n})$ are *queries*. Our notion of logic program differs somewhat from the standard: usually, logic programs are atomic predicates in first order logic whose axioms are Horn clauses, where queries are expressed in terms of proof search on Σ_1 formulas.We justify ourselves by noting that Horn clauses can be naturally be represented in terms of judgments, where depth-first search on derivations using logic variables faithfully models proof-search for the corresponding Σ_1 formulas. Thus, we sometimes abuse terminology by referring to depth-first search over judgments as *proof search*. We can use the logic programming interpretation of judgments to help us characterize the provably total functions of syntactic finitism. We proceed as follows.

In Section 5.1, we define a particular syntactic category whose terms can be used to *adequately* represent arbitrary derivations of arbitrary judgments. We refer to such a syntactic category as a *logical framework*; the logical framework we are most interested in is *LF* [HHP87], although, for reasons that will be discussed later, we present a spine-calculus/Herbelin-style[CP03, Her95] variant of Canonical LF[HL07]. In Section 5.2, we define a big-step operational semantics for proof search over the terms of the logical framework, thus providing a syntactic specification for the notion of logic programming. However, not all logic programs are total functions: they must be well-moded, terminating and cover all cases. In Section 5.2.5, we carve out a subset of well-behaved logic programs by judgmentally specifying an algorithm—inspired by the one sketched in [RP96, Roh96]—that can guarantee the well-modedness and termination of proof search. The mode/termination analysis is modular in the ordering used on LF terms, meaning that it can, in principle, capture syntactically finitary reasoning based on both the subterm ordering and lexicographic path ordering. We prove the correctness of the mode/termination checker using syntactically finitary methods extended by a principle of well-founded induction based on this modular ordering. We leave a syntactic characterization of a coverage checker, such as [SP03], to future work.

5.1 Canonical Spine LF

In Section 1.1, we saw how BNF-style equations can be used to define syntactic categories, where each clause of a definition specifies both the name of a term constructor, and the syntactic categories that the term constructor can be applied to. Informally, if we think of syntactic categories as being atomic types, the applicability of term constructors can be captured by the notion of function-type. This observation inspires us to view the following type-theory inspired definition of the natural numbers as being nothing more than a notational variation on the usual BNF-style definition.

$$\begin{aligned}&\mathit{nat} : \mathit{type}\\ &z : \mathit{nat}\\ &s : \mathit{nat} \rightarrow \mathit{nat}\end{aligned}$$

We have also seen that, by allowing syntactic categories to depend on one another, any judgment that can be defined using inference rules can also be defined using BNF-style definitions. We refer to this observation as the *judgments-as-types* principle because, in general, judgments and (dependent) syntactic categories can be expressed using type-theoretic declarations like the one above, extended with *dependent types*. For example, we view the following definition as being nothing more than a notational variation on the usual definitions of the judgments (or, alternatively, syntactic categories) $\mathit{even}(n)$ and $\mathit{odd}(n)$ (or, alternatively, E^n and O^n).

$$\begin{aligned}&\mathit{even} : \mathit{nat} \rightarrow \mathit{type}\\ &\mathit{odd} : \mathit{nat} \rightarrow \mathit{type}\\ &\mathit{evenz} : \mathit{even}\, z\\ &\mathit{evens} : \Pi n{:}\mathit{nat}.\, \mathit{odd}\, n \rightarrow \mathit{even}(s\, n)\\ &\mathit{odds} : \Pi n{:}\mathit{nat}.\, \mathit{even}\, n \rightarrow \mathit{odd}(s\, n)\end{aligned}$$

In Section 1.3, we introduced the concept of hypothetical judgment, where inference rules and term constructors can bind variables, and and hypothetical judgments (such as $\vdash \tau\ nd$) are equivalent to syntactic categories (such as e^τ). We express such hypothetical definitions in type-theoretic notation using higher-order types, where functions of type $\rightarrow$ and Π can do no more than perform substitutions for bound variables. We define the syntactic categories τ and e^τ using this notation below.

$$
\begin{aligned}
&tp : type\\
&o : tp\\
&\Rightarrow : tp \rightarrow tp \rightarrow tp. \qquad\qquad \text{(we treat $\Rightarrow$ as an infix operator)}\\
&exp : \Pi t_1{:}tp.\Pi t_2{:}tp.((exp\ t_1) \rightarrow (exp\ t_2)) \rightarrow exp\ (t_1 \Rightarrow t_2)\\
&app : \Pi t_1{:}tp.\Pi t_2{:}tp.(exp\ (t_1 \Rightarrow t_2)) \rightarrow (exp\ t_1) \rightarrow (exp\ t_2)
\end{aligned}
$$

We denote variable-binding functions using λ-notation, meaning that the term expressed in BNF-style notation as $(lam\, x^o.x^o)^{o\Rightarrow o}$ would be expressed in type-theory inspired notation as $lam\ o\ o\ (\lambda x{:}o.x)$, which can be accurately represented by a term of the same name in a dependently-typed λ-calculus with constants of the appropriate types.

This leads us to the following observation. Given a dependently-typed λ-calculus, it should be straightforward to adequately encode judgments as types and inference rules as constants, such that every derivation is isomorphic to a *canonical* λ-calculus term of the appropriate type. The logical framework LF [HHP87] is just such a λ-calculus, where canonical forms are β-short and η-long. The proof that every LF term can be converted to a canonical form involves a nontrivial proof by logical relations [HP05]. Although we have seen in Chapter 2 that logical relations proofs are compatible with syntactic finitism, converting expressing this result in terms of structural logical relations would be prohibitively complex for the purpose of defining proof search. Instead, we rely on a formalization of LF similar to *canonical LF* [HL07], whose metatheory is clearly syntactically finitary. Here, we define a spine calculus variation of canonical LF because we feel that this formulation leads to a more natural account of proof search. We provide a technical treatment of the metatheory canonical spine LF, closely mirroring the technical development of canonical LF in [HL07], in the following subsections. We will not formally address the issue of proving adequacy theorems for this formulation; instead we defer to the treatment provided in the aforementioned references.

5.1.1 Terms and Typing Rules

Below, we present the abstract syntax for a spine-calculus [CP03] variation of canonical LF [HL07]. This presentation closely mimics the way that LF is represented internally by the programming language Twelf, and simplifies the presentation of proof-search for LF terms in Section 5.2. It should also be noted that this presentation of LF closely resembles the cut-free proof terms for Herbelin's LJT [Her95], and so we adopt some of the notational conventions from this paper.

Kinds	K, L	$::=$	$\mathit{type} \mid \Pi x{:}A.K$
Type Families	A, B	$::=$	$a\,\overline{M} \mid \Pi x{:}A.B$
Canonical Terms	M, N	$::=$	$x\,\overline{M} \mid c\,\overline{M} \mid \lambda x{:}A.M$
Canonical Spines	$\overline{M}, \overline{N}$	$::=$	$\cdot \mid N :: \overline{M}$
Contexts	Γ	$::=$	$\cdot \mid \Gamma, x{:}A$
Signatures	Σ	$::=$	$\cdot \mid \Sigma, a{:}K \mid \Sigma, c{:}A$

Note that, unlike the presentation of canonical LF in [HL07], we include type-labels on λ-abstractions; although these type-labels are unnecessary, having them makes the specification of the mode/termination checker somewhat simpler.

For convenience, we use the letter h to stand for heads of terms (i.e. either x or c).A term that would be written as $h\ M_0\ \ldots\ M_n$ in conventional presentations of LF would be written as $h\,(M_0 :: \ldots :: M_n :: \cdot)$ here (although we sometimes write the former as shorthand for the latter). For any syntactic category E among $K, A, \overline{M}$ and Γ, we write $x \in FV(E)$ to refer to the judgment that characterizes when x occurs freely in E, and write $x\#E$ to refer to the judgment that characterizes when x is not among the free variables of E (we inherit this convention from [HL07]). We sometimes write $\Pi x{:}A.B$ as $A \to B$ when $x\#B$ holds. In general, we require $x\#\Gamma$ to hold in order for $\Gamma, x{:}A$ to be well-formed; this side condition can typically be satisfied by the tacit renaming of bound variables. Similarly, we require that each term- or family-level constant may be declared at most once in Σ.

We use the notion of simple types to help define hereditary substitutions; although we reuse the symbols σ and τ, the simple types defined below not to be confused with the similar notion of types defined in Section1.3.

$$\textbf{Simple Types} \quad \tau, \sigma \quad ::= \quad a \mid \tau \to \sigma$$

The mapping of type-families to simple types is straightforward.

$$\begin{aligned} (a\,\overline{M})^- &= a \\ (\Pi x{:}A.B)^- &= (A^-) \to (B^-) \end{aligned}$$

We define the notion of *head* for type families and simple types below.

$$\begin{aligned} \texttt{hd}(a\,\overline{M}) &= a \\ \texttt{hd}(\Pi x{:}A.B) &= \texttt{hd}(B) \\ \texttt{hd}(a) &= a \\ \texttt{hd}(\tau \to \sigma) &= \texttt{hd}(\sigma) \end{aligned}$$

Clearly, for every A, $\texttt{hd}(A) = \texttt{hd}(A^-)$.

An LF specification is parameterized not only by a signature Σ (not to be confused with the notion of signature from Chapter 4), but also by a (decidable) binary *subordination relation*, $\sqsubseteq$, on the family-level constants in Σ; we lift $\sqsubseteq$ to arbitrary families by writing $A \sqsubseteq B$ as shorthand for $\mathtt{hd}(A) \sqsubseteq \mathtt{hd}(B)$. Intuitively, $A \sqsubseteq B$ holds whenever terms of type A can occur inside of terms of type B. The typing, well-formedness and substitution judgments for LF are listed along with their informal meanings below.

$\Gamma \vdash_{\Sigma,\sqsubseteq} K : \mathit{kind}$	K is a valid kind
$\Gamma \vdash_{\Sigma,\sqsubseteq} A : K$	A is a family of kind K
$\Gamma; K \vdash_{\Sigma,\sqsubseteq} \overline{M} : K'$	$\overline{M}$ is a spine that transforms the kind K into K'
$\Gamma \vdash_{\Sigma,\sqsubseteq} M : A$	M is a term of type A
$\Gamma; A \vdash_{\Sigma,\sqsubseteq} \overline{M} : B$	$\overline{M}$ is a spine that transforms A into B
$\vdash_{\Sigma,\sqsubseteq} \Gamma : \mathtt{ctx}$	Γ is a valid context
$\vdash_{\sqsubseteq} \Sigma : \mathtt{sig}$	Σ is a valid signature
$[M/x]_\tau K = K'$	substituting M for x in K results in K'
$[M/x]_\tau A = A'$	substituting M for x in A results in A'
$[M/x]_\tau N = N'$	substituting M for x in N results in N'
$[M/x]_\tau \overline{M} = \overline{M'}$	substituting M for x in $\overline{M}$ results in $\overline{M'}$
$[M/x]_\tau \Gamma = \Gamma'$	substituting M for x in Γ results in Γ'
$\mathtt{reduce}_\tau(M, \overline{M}) = N$	applying each element of the spine $\overline{M}$ to M results in N

All of the typing rules for kinds, families, spines and terms have a premiss of the form $\vdash_{\Sigma,\sqsubseteq} \Gamma : \mathtt{ctx}$; we will not explicitly write this premiss for the sake of readability. We begin specifying the inference rules for these judgments below.

$$\frac{}{\Gamma \vdash_{\Sigma,\sqsubseteq} \mathit{type} : \mathit{kind}}\ \textit{KIND-TYPE} \qquad \frac{\Gamma \vdash_{\Sigma,\sqsubseteq} A : \mathit{type} \quad \Gamma, x{:}A \vdash_{\Sigma,\sqsubseteq} K : \mathit{kind}}{\Gamma \vdash_{\Sigma,\sqsubseteq} \Pi x{:}A.K : \mathit{kind}}\ \textit{KIND-PI}$$

$$\frac{\Gamma \vdash_{\Sigma,\sqsubseteq} A : \mathit{type} \quad \Gamma, x{:}A \vdash_{\Sigma,\sqsubseteq} B : \mathit{type} \quad \mathtt{hd}(A) \sqsubseteq \mathtt{hd}(B)}{\Gamma \vdash_{\Sigma,\sqsubseteq} \Pi x{:}A.B : \mathit{type}}\ \textit{FAM-PI}$$

$$\frac{a{:}K \text{ in } \Sigma \quad \Gamma;\, K \vdash_{\Sigma,\sqsubseteq} \overline{M} : \mathit{type}}{\Gamma \vdash_{\Sigma,\sqsubseteq} a\,\overline{M} : \mathit{type}}\ \textit{FAM-ATM} \qquad \frac{}{\Gamma;\, K \vdash_{\Sigma,\sqsubseteq} \cdot : K}\ \textit{FAM-LIST-NIL}$$

$$\frac{\Gamma \vdash_{\Sigma,\sqsubseteq} M : A \quad [M/x]_{A^-}K = K' \quad \Gamma;\, K' \vdash_{\Sigma,\sqsubseteq} \overline{M} : L}{\Gamma;\, \Pi x{:}A.K \vdash_{\Sigma,\sqsubseteq} M :: \overline{M} : L}\ \textit{FAM-LIST-CONS}$$

$$\frac{\Gamma \vdash_{\Sigma,\sqsubseteq} A : \mathit{type} \quad \Gamma, x{:}A \vdash_{\Sigma,\sqsubseteq} M : B}{\Gamma \vdash_{\Sigma,\sqsubseteq} \lambda x{:}A.M : \Pi x{:}A.B}\ \textit{TERM-LAM} \qquad \frac{c{:}A \text{ in } \Sigma \quad \Gamma;\, A \vdash_{\Sigma,\sqsubseteq} \overline{M} : a\,\overline{N}}{\Gamma \vdash_{\Sigma,\sqsubseteq} c\,\overline{M} : a\,\overline{N}}\ \textit{TERM-ATM-C}$$

$$\frac{x{:}A \text{ in } \Gamma \quad \Gamma;\, A \vdash_{\Sigma,\sqsubseteq} \overline{M} : a\,\overline{N}}{\Gamma \vdash_{\Sigma,\sqsubseteq} x\,\overline{M} : a\,\overline{N}}\ \textit{TERM-ATM-V} \qquad \frac{}{\Gamma;\, A \vdash_{\Sigma,\sqsubseteq} \cdot : A}\ \textit{TERM-LIST-NIL}$$

$$\frac{\Gamma \vdash_{\Sigma,\sqsubseteq} M : A \quad [M/x]_{A^-}B = B' \quad \Gamma;\, B' \vdash_{\Sigma,\sqsubseteq} \overline{M} : A'}{\Gamma;\, \Pi x{:}A.B \vdash_{\Sigma,\sqsubseteq} M :: \overline{M} : A'}\ \textit{TERM-LIST-CONS}$$

$$\frac{}{\vdash_{\sqsubseteq} \cdot : \mathtt{sig}}\ \textit{SIG-EMPTY} \qquad \frac{\vdash_{\sqsubseteq} \Sigma : \mathtt{sig} \quad \Gamma \vdash_{\Sigma,\sqsubseteq} A : \mathit{type} \quad c\#\Sigma}{\vdash_{\sqsubseteq} \Sigma, c{:}A : \mathtt{sig}}\ \textit{SIG-TERM}$$

$$\frac{\vdash_{\sqsubseteq} \Sigma : \mathtt{sig} \quad \Gamma \vdash_{\Sigma,\sqsubseteq} K : \mathit{kind} \quad a\#\Sigma}{\vdash_{\sqsubseteq} \Sigma, a{:}K : \mathtt{sig}}\ \textit{SIG-FAM}$$

$$\frac{}{\vdash_{\Sigma,\sqsubseteq} \cdot : \mathtt{ctx}}\ \textit{CTX-EMPTY} \qquad \frac{\vdash_{\Sigma,\sqsubseteq} \Gamma : \mathtt{ctx} \quad \Gamma \vdash_{\Sigma,\sqsubseteq} A : \mathit{type} \quad x\#\Gamma}{\vdash_{\Sigma,\sqsubseteq} \Gamma, x{:}A : \mathtt{ctx}}$$

The notion of hereditary substitution $[M/x]_\tau E = E'$ defined below is not to be confused with the ordinary substitutions used in earlier chapters; our spine calculus has the flavor of a cut-free sequent calculus, and performing a hereditary substitution is akin to using *cut* as an admissible rule. In particular, hereditary substitutions can be viewed as partial functions on arbitrary syntactic categories, and as total functions on well-formed syntactic categories. We define hereditary substitution judgmentally below.

$$\frac{}{[M/x]_\tau type = type} \qquad \frac{[M/x]_\tau A = A' \quad [M/x]_\tau K = K' \quad x \# y}{[M/x]_\tau \Pi y{:}A.K = \Pi y{:}A'.K'}$$

$$\frac{[M/x]_\tau \overline{M} = \overline{N}}{[M/x]_\tau a\,\overline{M} = a\,\overline{N}} \qquad \frac{[M/x]_\tau A = A' \quad [M/x]_\tau B = B' \quad x \# y}{[M/x]_\tau \Pi y{:}A.B = \Pi y{:}A'.B'}$$

$$\frac{h \# x \quad [M/x]_\tau \overline{N} = \overline{N'}}{[M/x]_\tau h\,\overline{N} = h\,\overline{N'}} \qquad \frac{[M/x]_\tau \overline{N} = \overline{N'}}{[M/x]_\tau a\,\overline{N} = a\,\overline{N'}} \qquad \frac{[M/x]_\tau \overline{N} = \overline{N'} \quad \texttt{reduce}_\tau(M, \overline{N'}) = M'}{[M/x]_\tau x\,\overline{N} = M'}$$

$$\frac{[M/x]_\tau A = A' \quad [M/x]_\tau N = N' \quad x \# y}{[M/x]_\tau \lambda y{:}A.N = \lambda y{:}A'.N'}$$

$$\frac{}{[M/x]_\tau \cdot = \cdot} \qquad \frac{[M/x]_\tau N = N' \quad [M/x]_\tau \overline{N} = \overline{N'}}{[M/x]_\tau N :: \overline{N} = N' :: \overline{N'}}$$

$$\frac{[N/x]_\tau M = M' \quad \texttt{reduce}_\sigma(M', \overline{N}) = N'}{\texttt{reduce}_{\tau \to \sigma}(\lambda x{:}A.M, N :: \overline{N}) = N'} \qquad \frac{}{\texttt{reduce}_\tau(M, \cdot) = M}$$

$$\frac{}{[M/x]_\tau \cdot = \cdot} \qquad \frac{[M/x]_\tau \Gamma = \Gamma' \quad [M/x]_\tau A = B \quad x \# y \quad y \# M}{[M/x]_\tau \Gamma, y{:}A = \Gamma', y{:}B}$$

The subordination $\sqsubseteq$ relation specified along with the signature Σ;we expect this specification to be both syntactic and decidable. Just as some signatures are considered ill-formed, some subordination relations also considered ill-formed. We define well-formed subordination relations in the style of [HL07], as opposed to [Vir99]), below, although the distinction between these two approaches does not affect our presentation.

Definition 5.1.1 (Subordination Relation) *Given Σ, the subordination relation $\sqsubseteq$ is well formed iff it satisfies all of the following properties:*

1. $\vdash_\sqsubseteq \Sigma : \texttt{sig}$
2. *For all $a{:}\Pi x{:}A_1 \ldots \Pi x{:}A_n.\ type$ in Σ, $\texttt{hd}(A_i) \sqsubseteq a$ (for all $i \in 1, \ldots, n$)*
3. *$\sqsubseteq$is reflexive*

4. *$\sqsubseteq$ is transitive*

Note that condition 2 can be expressed judgmentally (i.e. without the use of ellipses); we omit this characterization for the sake of readability.

We write $\sqsubset$ for the strict portion of $\sqsubseteq$ (i.e. $a \sqsubset b$ iff $a \sqsubseteq b$ and $b \not\sqsubseteq a$) and $\equiv$ for the reflexive portion of $\sqsubseteq$ (i.e. $a \equiv b$ iff $a \sqsubseteq b$ and $b \sqsubseteq a$). We say that a and b are mutually recursive *whenever $a \equiv b$.*

Although we follow [HL07] by viewing $\sqsubseteq$ as part of an LF specification, in practice the "strongest subordination relation" can be computed from Σ. From now on, we will always assume that fixed, well-formed specifications of $\sqsubseteq$ and Σ have been given; thus, for the sake of readability, we will use $\vdash$ in place of $\vdash_{\Sigma,\sqsubseteq}$ whenever there is no ambiguity.

5.1.2 Hereditary Substitutions

Here, as in canonical LF, hereditary substitutions play the dual role of substitution and normalization. In this section, we prove that Hereditary substitutions are applied to well-formed terms are well behaved.

Lemma 5.1.2 (Decidability of Substitution)

1. *For any E in $\{K, A, M, \overline{M}, \Gamma\}$, for every M, x and τ there either exists an E' such that $[M/x]_\tau E = E'$ or no such E' exists*

2. *For every $M, \overline{M}$ and τ either there exists an N such that $\texttt{reduce}_\tau(M, \overline{M}) = N$ or no such N exists*

Proof: By mutual induction: τ followed by E for 1, and τ followed by $\overline{M}$ for 2. $\square$

Lemma 5.1.3 (Uniqueness of Substitutions and Formation)

1. *For any E in $\{K, A, M, \overline{M}\}$ if $[M/x]_\tau E = E'$ and $[M/x]_\tau E = E''$ then $E' = E''$*

2. *If $\texttt{reduce}_\tau(M, \overline{M}) = N$ and $\texttt{reduce}_\tau(M, \overline{M}) = N'$ then $N = N'$*

3. *If $\Gamma \vdash A : K$ and $\Gamma \vdash A : K'$ then $K = K'$*

4. *If $\Gamma \vdash M : A$ and $\Gamma \vdash M : B$ then $A = B$*

5. *If $\Gamma; A \vdash \overline{M} : P$ and $\Gamma; A \vdash \overline{M} : P'$ then $P = P'$*

Proof: By straightforward mutual inductions on the structures of the given derivations. □

Lemma 5.1.4 (Substitutions Preserve Freshness)

1. *Given E among $\{K, A, M, \overline{M}\}$, if $[M/x]_\tau E = E'$ and $y\#M$ and $y\#E$ then $y\#E'$*

2. *If $\texttt{reduce}_\tau(M, \overline{M}) = N$ and $x\#M$ and $x\#\overline{M}$ then $x\#N$*

Proof: By straightforward mutual induction on the given substitution derivations. □

Lemma 5.1.5 (Contexts Contain Free Variables) *If $x\#\Gamma$ then:*

1. *if $\Gamma \vdash K : kind$ then $x\#K$*

2. *if $\Gamma; K \vdash \overline{M} : K'$ then $x\#K$ and $x\#\overline{M}$ and $x\#K'$*

3. *if $\Gamma \vdash A : type$ then $x\#A$*

4. *if $\Gamma; A \vdash \overline{M} : B$ then $x\#A$ and and $x\#\overline{M}$ and $x\#B$*

5. *if $\Gamma \vdash M : A$ then $x\#M$ and $x\#A$*

Proof: By straightforward mutual induction on the given typing derivations, using Lemma 5.1.4 in the *TERM-LIST-CONS* and *FAM-LIST-CONS* cases. $\Box$

Lemma 5.1.6 (Weakening for Typing)

1. *If* $\Gamma, \Gamma' \vdash M : A$ *and* $\Gamma \vdash B : \mathit{type}$ *then* $\Gamma, x{:}B, \Gamma' \vdash M : A$

2. *If* $\Gamma, \Gamma' \vdash A : K$ *and* $\Gamma \vdash B : \mathit{type}$ *then* $\Gamma, x{:}B, \Gamma' \vdash A : K$

3. *If* $\Gamma, \Gamma'; A \vdash \overline{M} : A'$ *and* $\Gamma \vdash B : \mathit{type}$ *then* $\Gamma, x{:}B, \Gamma'; A \vdash \overline{M} : A'$

4. *If* $\Gamma, \Gamma'; K \vdash \overline{M} : K'$ *and* $\Gamma \vdash B : \mathit{type}$ *then* $\Gamma, x{:}B, \Gamma'; K \vdash \overline{M} : K'$

Proof: By straightforward mutual inductions. $\Box$

Strengthening isn't used to prove the substitution lemma, but it will be useful later on.

Lemma 5.1.7 (Strengthening for Typing)

1. *If* $\Gamma, x{:}B, \Gamma' \vdash M : A$ *and* $x \# \Gamma'$ *and* $x \# M$ *then* $\Gamma, \Gamma' \vdash M : B$

2. *If* $\Gamma, x{:}B, \Gamma' \vdash A : K$ *and* $x \# \Gamma'$ *then* $\Gamma, \Gamma' \vdash A : K$

3. *If* $\Gamma, x{:}B, \Gamma'; A \vdash \overline{M} : A'$ *and* $x \# \Gamma'$ *and* $x \# \overline{M}$ *then* $\Gamma, \Gamma'; A \vdash \overline{M} : A'$

4. *If* $\Gamma, x{:}B, \Gamma'; K \vdash \overline{M} : K'$ *and* $x \# \Gamma'$ *and* $x \# K$ *and* $x \# \overline{M}$ *then* $\Gamma, \Gamma'; K \vdash \overline{M} : K'$

Proof: By straightforward mutual inductions, using Lemma 5.1.4 in 3 and 4. $\Box$

Lemma 5.1.8 (Vacuous Substitutions) *Given* M, x, τ *and* E *among* $\{K, A, M, \overline{M}\}$, *if* $x \# E$ *then* $[M/x]_\tau E = E$

Proof: By induction on the structure of E. $\Box$

Lemma 5.1.9 (Substitutions Consume Variables) *Given M, x, τ and E among $\{K, A, M, \overline{M}\}$, if $[M/x]_\tau E = E'$ and $x\#M$ then $x\#E'$*

Proof: By induction the given substitution derivation. $\square$

Lemma 5.1.10 (Erasure is Invariant Under Substitutions) *If $[M/x]_\tau A = A'$ then $A^- = A'^-$*

Proof: By straightforward induction over the structure of the substitution derivation. $\square$

The following lemma can be thought of both as a kind of commutativity lemma for hereditary substitutions, and form of Church-Rosser.

Lemma 5.1.11 (Composition of Substitutions)

1. *For all E in $\{K, A, M, \overline{M}\}$, if $\mathcal{D} : ([M_2/x]_{\tau_2} E_1 = E)$ and $\mathcal{E} : ([M_0/x_0]_{\tau_0} E_1 = E_1')$ and $\mathcal{F} : ([M_0/x_0]_{\tau_0} M_2 = M_2')$ and $x\#x_0$ and $x\#M_0$ and $x\#M_2$ then there exists and E' such that $[M_0/x_0]_{\tau_0} E = E'$ and $[M_2'/x]_{\tau_2} E_1' = E'$*

2. *If $\mathcal{D} : (\texttt{reduce}_{\tau_2}(M_1, \overline{M_2}) = M')$ and $\mathcal{E} : ([M_0/x_0]_{\tau_0} M_1 = M_1')$ and $\mathcal{F} : ([M_0/x_0]_{\tau_0} \overline{M_2} = \overline{M_2'})$ and $x_0\#M_0$ then there exists an M' such that $\texttt{reduce}_{\tau_0}(M_1', \overline{M_2'}) = M'$ and $[M_0/x_0]_{\tau_0} M = M'$*

Proof: By mutual induction on the (commutative generalization of) the simultaneous ordering of τ_0 and τ_2, followed by $\mathcal{D}$. Most of the cases are straightforward; however the cases in 1 where $E_1 = x_0\,\overline{M_1}$ and $E_1 = x\,\overline{M_1}$ are somewhat tricky, as is the substitution case of 2. We show them here.

1.

Case:

$$\mathcal{D} \;=\; \cfrac{\overset{\mathcal{D}_0}{[M_2/x]_{\tau_2}\overline{M_1} = \overline{M}} \qquad \overset{\mathcal{D}_1}{\mathtt{reduce}_{\tau_2}(M_2, \overline{M}) = M}}{[M_2/x]_{\tau_2} x\,\overline{M_1} = M}$$

$$\mathcal{E} \;=\; \cfrac{\overset{\mathcal{E}_0}{[M_0/x_0]_{\tau_0}\overline{M_1} = \overline{M_1'}} \qquad x\#x_0}{[M_0/x_0]_{\tau_0} x\,\overline{M_1} = x\,\overline{M_1'}}$$

$[M_0/x_0]_{\tau_0} M_2 = M_2'$ and $x\#M_0$ and $x\#x_0$ and $x\#M_2'$ given

$[M_0/x_0]_{\tau_0}\overline{M} = \overline{M'}$ and $[M_2'/x]_{\tau_2}\overline{M} = \overline{M'}$ by IH 1 on τ_0, τ_2 and $\mathcal{D}_0$

$\mathtt{reduce}_{\tau_2}(M_2', \overline{M}') = M'$ and $[M_0/x_0]_{\tau_0} M = M'$ by IH 2 on τ_0, τ_2 and $\mathcal{D}_1$

$[M_2'/x]_{\tau_2} x\,\overline{M_1'} = M'$ by rule

Case:

$$\mathcal{D} \;=\; \cfrac{\overset{\mathcal{D}_0}{[M_2/x]_{\tau_2}\overline{M_1} = \overline{M}} \qquad x\#x_0}{[M_2/x]_{\tau_2} x_0\,\overline{M_1} = x_0\,\overline{M}}$$

$$\mathcal{E} \;=\; \cfrac{\overset{\mathcal{E}_0}{[M_0/x_0]_{\tau_0}\overline{M_1} = \overline{M_1'}} \qquad \overset{\mathcal{E}_1}{\mathtt{reduce}_{\tau_0}(M_0, \overline{M_1'}) = M_1'}}{[M_0/x_0]_{\tau_0} x_0\,\overline{M_1} = M_1'}$$

$[M_0/x_0]_{\tau_0} M_2 = M_2'$ and $x\#M_0$ and $x\#x_0$ and $x\#M_2'$ given

$[M_0/x_0]_{\tau_0}\overline{M} = \overline{M'}$ and $[M_2'/x]_{\tau_2}\overline{M_1'} = \overline{M'}$ by IH 1 on τ_0, τ_2 and $\mathcal{E}_1$

$[M_2'/x]_{\tau_2} M_0 = M_0$ by lemma 5.1.8

$\mathtt{reduce}_{\tau_0}(M_0, \overline{M'}) = M'$ and $[M_2'/x]_{\tau_2} M_1' = M'$ by IH 2 on τ_2, τ_0 and $\mathcal{D}_0$

$[M_0/x_0]_{\tau_0} x_0\,\overline{M_1} = M'$ by rule

2.

Case:

$$\mathcal{D} \quad = \quad \dfrac{\overset{\mathcal{D}_0}{[M_2/x]_{\tau_2} M_1 = N} \qquad \overset{\mathcal{D}_1}{\texttt{reduce}_{\sigma_2}(N, \overline{M_2}) = M}}{\texttt{reduce}_{\tau_2 \to \sigma}(\lambda x{:}A_1.M_1, M_2 :: \overline{M_2}) = M}$$

$$\mathcal{E} \quad = \quad \dfrac{\overset{\mathcal{E}_0}{[M_0/x_0]_{\tau_0} A_1 = A_1'} \qquad \overset{\mathcal{E}_1}{[M_0/x_0]_{\tau_0} M_1 = M_1'} \qquad x\#x_0}{[M_0/x_0]_{\tau_0} \lambda x{:}A_1.M_1 = \lambda x{:}A_1'.M_1'}$$

$$\mathcal{F} \quad = \quad \dfrac{\overset{\mathcal{F}_0}{[M_0/x_0]_{\tau_0} M_2 = M_2'} \qquad \overset{\mathcal{F}_1}{[M_0/x_0]_{\tau_0} \overline{M_2} = \overline{M_2'}}}{[M_0/x_0]_{\tau_0} M_2 :: \overline{M_2} = M_2' :: \overline{M_2'}}$$

$x_0\#M_0$	given
$x_0\#M_2'$	by Lemma 5.1.9
$[M_0/x_0]_{\tau_0} N = N'$ and $[M_2'/x]_{\tau_2} M_1' = N'$	by IH 1 on τ_0, τ_2 and $\mathcal{D}_0$
$\texttt{reduce}_{\sigma_2}(N', \overline{M_2'}) = M'$ and $[M_0/x_0]_{\tau_0} M = M'$	by IH 2 on σ_2, τ_0 and $\mathcal{D}_1$

□

We are now ready to show that hereditary substitutions are total functions on well-formed syntactic categories.

Theorem 5.1.12 (Hereditary Substitutions)

1. *If* $\Gamma; A \vdash \overline{M} : B$ *and* $\Gamma \vdash M : A$ *and* $A^- = \tau$ *then* $\texttt{reduce}_\tau(M, \overline{M}) = M'$ *and* $\Gamma \vdash M' : B$

2. *If* $\Gamma_0 \vdash M : A$ *and* $A^- = \tau$ *then:*

   (a) If $\vdash \Gamma_0, x{:}A, \Gamma_1 : \texttt{ctx}$ *then* $[M/x]_\tau \Gamma_1 = \Gamma_1'$ *and* $\vdash \Gamma_0, \Gamma_1' : \texttt{ctx}$

(b) *If* $\Gamma_0, x{:}A, \Gamma_1 \vdash K : kind$ *and* $[M/x]_\tau \Gamma_1 = \Gamma_1'$ *then* $[M/x]_\tau K = K'$ *and* $\Gamma_0, \Gamma_1' \vdash K' : ki$

(c) *If* $\Gamma_0, x{:}A, \Gamma_1 \vdash B : type$ *and* $[M/x]_\tau \Gamma_1 = \Gamma_1'$ *then* $[M/x]_\tau B = B'$ *and* $\Gamma_0, \Gamma_1' \vdash B' : typ$

(d) *If* $\Gamma_0, x{:}A, \Gamma_1;\ K \vdash \overline{M} : type$ *and* $[M/x]_\tau \Gamma_1 = \Gamma_1'$ *and* $[M/x]_\tau K = K'$ *then* $[M/x]_\tau \overline{M} = \overline{M'}$ *and* $\Gamma_0, \Gamma_1';\ K' \vdash \overline{M'} : type$

(e) *If* $\Gamma_0, x{:}A, \Gamma_1 \vdash N : B$ *and* $[M/x]_\tau \Gamma_1 = \Gamma_1'$ *then* $[M/x]_\tau N = N'$ *and* $[M/x]_\tau B = B'$ *and* $\Gamma_0, \Gamma_1' \vdash N' : B'$

(f) *If* $\Gamma_0, x{:}A, \Gamma_1;\ B \vdash \overline{M} : a\,\overline{N}$ *and* $[M/x]_\tau \Gamma_1 = \Gamma_1'$ *and* $[M/x]_\tau B = B'$ *then* $[M/x]_\tau \overline{M} = \overline{M'}$ *and* $[M/x]_\tau \overline{N} = \overline{N'}$ *and* $\Gamma_0, \Gamma_1';\ B' \vdash \overline{M'} : a\,\overline{N'}$

Proof: By simultaneous induction on τ followed by $\Gamma;\ A \vdash M : a\,\overline{N}$ for 1, and τ followed by the typing derivation of the syntactic category being substituted into for 2. In 1, the $\tau \to \sigma$ case uses Uniqueness of Substitutions and Formation. In 2, the *FAM-ATM* and *TERM-ATM-C* cases use the Vacuous Substitution Lemma; the *FAM-LIST-NIL* and *TERM-LIST-NIL* cases are direct; the *FAM-LIST-CONS* and *TERM-LIST-CONS* cases use Composition of Substitutions and Uniqueness of Substitutions; and the non-trivial *TERM-ATM-V* case uses Weakening for Typing (repeated inductively on Γ_1'). All other cases are straightforward. $\square$

The following lemma is a useful corollary to the Hereditary substitution theorem.

Lemma 5.1.13 (Regularity for Typing)

1. *If* $\vdash_\sqsubseteq \Sigma : \texttt{sig}$ *and* $a{:}K \in \Sigma$ *then* $\cdot \vdash K : kind$

2. *If* $\vdash_\sqsubseteq \Sigma : \texttt{sig}$ *and* $c{:}A \in \Sigma$ *then* $\cdot \vdash A : type$

3. *If* $\vdash \Gamma : \texttt{ctx}$ *and* $x{:}A \in \Gamma$ *then* $\Gamma \vdash A : type$

4. *If* $\Gamma \vdash K : kind$ *and* $\Gamma;\ K \vdash \overline{M} : K'$ *then* $\Gamma \vdash K : kind$

5. *If* $\Gamma \vdash A : K$ *then* $\Gamma \vdash K : kind$

6. *If* $\Gamma \vdash A : type$ *and* $\Gamma; A \vdash \overline{M} : B$ *then* $\Gamma \vdash B : type$

7. *If* $\Gamma \vdash M : A$ *then* $\Gamma \vdash A : type$

Proof: By straightforward induction on the structure of the given typing derivations.1 and 2 require an analog of Weakening for Typing for signatures, 3 uses Weakening for Typing, 4 and 6 use Theorem 5.1.12. $\square$

5.1.3 Eta Expansion

Unlike the conventionally defined λ-calculi, here heads (i.e. constants and variables) are not considered first-class expressions. However, any given head can be η-expanded into a first-class term. We find the notion of spine concatenation, defined below, useful for defining this notion of expansion.

$$\frac{}{(\cdot)@M = M :: \cdot} \qquad \frac{\overline{N}@M = \overline{M}}{(N :: \overline{N})@M = N :: \overline{M}}$$

$$\frac{}{\overline{M}@\cdot = \overline{M}} \qquad \frac{\overline{M}@N = \overline{M'} \quad \overline{M'}@\overline{N} = \overline{N'}}{\overline{M}@(N :: \overline{N}) = \overline{N'}}$$

Lemma 5.1.14 (Empty Concatenation) *For every* $\overline{M}$, $(\cdot)@\overline{M} = \overline{M}$

Proof: By a straightforward induction on the structure of $\overline{M}$ $\square$

Lemma 5.1.15 (Spines Are Concatenatable)

1. *For every* $\overline{M}$ *and* M *there exists* $\overline{N}$ *s.t.* $\overline{M}@M = \overline{N}$

2. *For every* $\overline{M_0}$ *and* $\overline{M_1}$ *there exists* $\overline{N}$ *s.t.* $\overline{M_0}@\overline{M_1} = \overline{N}$

Proof: 1 follows by straightforward induction on the structure of $\overline{M}$; 2 follows by straightforward induction on the structure of $\overline{M_1}$, using 1. $\square$

Lemma 5.1.16 (Uniqueness of Concatenation)

1. *If* $\overline{M}@M = N$ *and* $\overline{M}@M = N'$ *then* $N = N'$

2. *If* $\overline{M_0}@\overline{M_1} = N$ *and* $\overline{M_0}@\overline{M_1} = N'$ *then* $N = N'$

Proof: 1 follows by straightforward induction on the structure of $\overline{M}$; 2 follows by straightforward induction on the structure of $\overline{M_1}$, using 1. $\square$

Lemma 5.1.17 (Concatenation Preserves Freshness)

1. *If* $\overline{M_0}@M_1 = N$ *then* ($x\#\overline{M_0}$ *and* $x\#M_1$) *iff* $x\#N$

2. *If* $\overline{M_0}@\overline{M_1} = N$ *then* ($x\#\overline{M_0}$ *and* $x\#\overline{M_1}$) *iff* $x\#N$

Proof: 1 is by straightforward induction on $\overline{M_0}$, 2 is by straightforward induction on M_1 using 1. $\square$

Lemma 5.1.18 (Spine Concatenation Substitution)

1. *If* $[M/x]_\tau\overline{N} = \overline{N'}$ *and* $[M/x]_\tau N = N'$ *and* $\overline{N}@N = \overline{M}$ *then there exists* $\overline{M'}$ *s.t.* $[M/x]_\tau\overline{M} = \overline{M'}$ *and* $\overline{N}@N' = \overline{M'}$

2. *If* $[M/x]_\tau\overline{N_0} = \overline{N_0'}$ *and* $[M/x]_\tau\overline{N_1} = \overline{N_1'}$ *and* $\overline{N_0}@\overline{N_1} = \overline{N}$ *then there exists* $\overline{M}$ *such that* $[M/x]_\tau\overline{N} = \overline{M}$ *and* $\overline{N_0'}@\overline{N_1'} = \overline{M}$

Proof: 1 follows by a straightforward induction on the structure of $\overline{N}$, and 2 follows by a straightforward induction on the structure of $\overline{N_1}$, using 1. $\square$

Lemma 5.1.19 (Spine Concatenation Typing)

1. *If* $\Gamma; A \vdash \overline{M} : \Pi x{:}B_1.B_2$ *and* $\Gamma \vdash M : B_1$ *and* $[M/x]_{B_1^-}B_2 = B$ *and* $\overline{M}@M = \overline{M'}$ *then* $\Gamma; A \vdash \overline{M'} : B$

2. *If* $\Gamma;\, A \vdash \overline{M} : A'$ *and* $\Gamma;\, A' \vdash \overline{N} : B$ *and* $\overline{M} @ \overline{N} = \overline{M'}$ *then* $\Gamma;\, A \vdash \overline{M'} : B$

Proof: 1 follows by a straightforward induction on the structure of $\overline{M}$, 2 follows from a straightforward induction on the structure of $\overline{N}$, using 1. $\square$

We are now ready to define expansion.

Definition 5.1.20 (Expansion) *Expansion is implemented as a function on heads, types and spines, using the judgment* $\mathtt{expand}_h(A; \overline{M}) = M$*, whose rules are defined below.*

$$\frac{}{\mathtt{expand}_h(a\,\overline{M}; \overline{N}) = h\,\overline{N}}$$

$$\frac{\mathtt{expand}_x(A; \cdot) = N \quad x \# \overline{N} \quad \overline{N} @ N = \overline{M} \quad \mathtt{expand}_h(B; \overline{M}) = M}{\mathtt{expand}_h(\Pi x{:}A.B; \overline{N}) = \lambda x{:}A.M}$$

Theorem 5.1.25 summarizes the intuitive meaning of $\mathtt{expand}$.

As usual, we show that the judgment $\mathtt{expand}_h(A; \overline{M}) = N$ can be viewed as a (deterministic) function by means of case analysis and induction.

Lemma 5.1.21 (Existence, Uniqueness of Eta Expansion)

1. *For every* $h, \overline{M}$ *and* A*, there exists an* M *such that* $\mathtt{expand}_h(A; \overline{M}) = M$
2. *If* $\mathtt{expand}_h(A; \overline{M}) = M$ *and* $\mathtt{expand}_h(A; \overline{M}) = M'$ *then* $M = M'$

Proof: Both are by straightforward inductions on A^-. In 1's induction step, the freshness condition can always be satisfied by the renaming of bound variables; 2's induction step uses Lemma 5.1.16 $\square$

Lemma 5.1.22 (Expansion Preserves Freshness)

1. *If* $\mathcal{D} : (\mathtt{expand}_h(A; \overline{M}) = M)$ *and* $x \# h$ *and* $x \# \overline{M}$ *then* $x \# M$
2. *If* $\mathcal{D} : (\mathtt{expand}_h(A; \overline{M}) = M)$ *and* $x \# M$ *then* $x \# h$ *and* $x \# \overline{M}$

Proof: Both cases are by straightforward induction on the structure of $\mathcal{D}$ using 5.1.17. □

Lemma 5.1.23 (Trivial Substitution on Eta Expansion)
If $\mathcal{D} : (\texttt{expand}_h(A; \overline{M}) = M)$ *and* $x \# h$ *and* $[N/x]_\tau A = A'$ *and* $[N/x]_\tau \overline{M} = \overline{M'}$ *then* $[N/x]_\tau M = M'$ *and* $\texttt{expand}_h(A'; \overline{M'}) = M'$*.*

Proof: By straightforward induction on the structure of $\mathcal{D}$, using Lemma 5.1.18. □

Lemma 5.1.24 (Undoing Concatenation in Reduction)
If $\mathcal{D} : (\Gamma; A \vdash \overline{M_0} : \Pi x{:}B_1.B_2)$ *and* $\mathcal{E} : (\Gamma \vdash M : A)$ *and* $\Gamma \vdash M_1 : B_1$ *and* $\mathcal{F} : \overline{M_0} @ M_1 = \overline{M}$ *and* $\mathcal{G} : (\texttt{reduce}_{A^-}(M, \overline{M}) = N)$ *then* $\texttt{reduce}_{A^-}(M, \overline{M_0}) = \lambda x{:}B_1.N'$ *and* $[M_1/x]_{B_1^-} N' = N$

Proof: By induction on the structure of $\overline{M_0}$; the reasoning is straightforward, but the case analysis is not. We show both cases below.

Case:

$$\mathcal{D} \quad = \quad \dfrac{}{\Gamma; \Pi x{:}B_1.B_2 \vdash \cdot : \Pi x{:}B_1.B_2}\ \textit{TERM-LIST-NIL}$$

$$\mathcal{E} \quad = \quad \dfrac{\overset{\mathcal{E}_0}{\Gamma \vdash B_1 : \textit{type}} \qquad \overset{\mathcal{E}_1}{\Gamma, x{:}B_1 \vdash M : B_2}}{\Gamma \vdash \lambda x{:}B_1.M : \Pi x{:}B_1.B_2}\ \textit{TERM-LAM}$$

$$\mathcal{F} \quad = \quad \dfrac{}{\cdot\, @ M_1 = M_1 :: \cdot}$$

$$\mathcal{G} \quad = \quad \dfrac{\overset{\mathcal{G}_0}{[M_1/x]_{B_1^-} M = N} \qquad \dfrac{}{\texttt{reduce}_{B_2^-}(N, \cdot) = N}}{\texttt{reduce}_{B_1^- \to B_2^-}(\lambda x{:}B_1.M, M_1 :: \cdot) = N'}$$

$\texttt{reduce}_\tau(\lambda x{:}B_1.M, \cdot) = \lambda x{:}B_1.M$ by rule

$[M_1/x]_{B_1^-} M = N$ given ($\mathcal{G}_0$)

Case:

$$\mathcal{D} \;=\; \dfrac{\overset{\mathcal{D}_0}{\Gamma \vdash M_0 : A_1} \qquad \overset{\mathcal{D}_1}{[M_0/x]_{A_1^-} A_2 = A_2'} \qquad \overset{\mathcal{D}_2}{\Gamma;\, A_2' \vdash \overline{M_0} : \Pi x{:}B_1.B_2}}{\Gamma;\, \Pi y{:}A_1.A_2 \vdash M_0 :: \overline{M_0} : \Pi x{:}B_1.B_2}$$

$$\mathcal{E} \;=\; \dfrac{\overset{\mathcal{E}_0}{\Gamma \vdash A_1 : \mathit{type}} \qquad \overset{\mathcal{E}_1}{\Gamma, y{:}A_1 \vdash M : A_2}}{\Gamma \vdash \lambda y{:}A_1.M : \Pi y{:}A_1.A_2}\ \textit{TERM-LAM}$$

$$\mathcal{F} \;=\; \dfrac{\overset{\mathcal{F}_0}{\overline{M_0} @ M_1 = \overline{M}}}{(M_0 :: \overline{M_0}) @ M_1 = M_0 :: \overline{M}}$$

$$\mathcal{G} \;=\; \dfrac{\overset{\mathcal{G}_0}{[M_0/y]_{A_1^-} M = M'} \qquad \overset{\mathcal{G}_1}{\texttt{reduce}_{A_2^-}(M', \overline{M}) = N}}{\texttt{reduce}_{A_1^- \to A_2^-}(\lambda y{:}A_1.M, M_0 :: \overline{M}) = N}$$

$\Gamma \vdash M' : A_2'$ by Theorem 5.1.12 and Lemma 5.1.3

$\texttt{reduce}_{A_2^-}(M', \overline{M_0}) = \lambda x{:}B_1.N'$

and $[M_1/x]_{B^1-} N' = N$ by IH on $\overline{M_0}$

$\texttt{reduce}_{A_1^- \to A_2^-}(\lambda y{:}A_1.M, M_0 :: \overline{M_0}) = \lambda x{:}B_1.N'$ by rule

$\square$

Theorem 5.1.25 (Soundness of Eta Expansion)

1. *If* $h{:}A \in \Gamma$ *or* $h{:}A \in \Sigma$ *and* $\Gamma;\, A \vdash \overline{M} : B$ *and* $\mathcal{D} : (\texttt{expand}_h(B; \overline{M}) = M)$ *then* $\Gamma \vdash M : B$

2. *If* $\texttt{expand}_x(B; \cdot) = M$ *then:*

   (a) if $\mathcal{D} : (\Gamma \vdash K : \mathit{kind})$ *then* $[M/x]_{B^-} K = K$

(b) *if* $\mathcal{D} : (\Gamma;\ K \vdash \overline{N} : K')$ *then* $[M/x]_{B^-}\overline{N} = \overline{N}$

(c) *if* $\mathcal{D} : (\Gamma \vdash A : \mathit{type})$ *then* $[M/x]_{B^-}A = A$

(d) *if* $\mathcal{D} : (\Gamma;\ A \vdash \overline{N} : A')$ *then* $[M/x]_{B^-}\overline{N} = \overline{N}$

(e) *if* $\mathcal{D} : (\Gamma \vdash N : A)$ *then* $[M/x]_{B^-}N = N$

3. *If* $\mathcal{D} : (\mathtt{expand}_h(B; \overline{M_0}) = M)$ *and* $\mathcal{E} : (\Gamma;\ B \vdash \overline{M_1} : a\,\overline{N})$ *and* $\mathcal{F} : (\overline{M_0}@\overline{M_1} = \overline{M})$ *then* $\mathtt{reduce}_{B^-}(M, \overline{M_1}) = h\,\overline{M}$

4. *If* $\Gamma \vdash N : A$ *and* $\Gamma;\ A \vdash \overline{M} : B$ *and* $x\#A$ *and* $x\#\overline{M}$ *and* $x\#B$ *and* $\mathcal{D} : \mathtt{expand}_x(B; \overline{M}) = M$ *then* $\mathtt{reduce}_{A^-}(N, \overline{M}) = M'$ *and* $[N/x]_{A^-}M = M'$

Proof: By mutual induction. 2 is by induction on B^- followed by the structure of the thing being substituted into, 3 is by induction on the structure of B^- followed by $\overline{M_0}$, and all of the other cases are by induction B^- followed by a dummy ordering. We show some representative cases below. $\square$

1.

Case:

$$\mathcal{D} \;=\; \dfrac{\overset{\mathcal{D}_0}{\mathtt{expand}_x(B_1; \cdot) = M_1} \quad x\#\overline{M} \quad \overline{M}@M_1 = \overline{M_2} \quad \overset{\mathcal{D}_1}{\mathtt{expand}_h(B_2; \overline{M_2}) = M}}{\mathtt{expand}_h(\Pi x{:}B_1.B_2; \overline{M}) = \lambda x{:}B_1.M}$$

$h{:}A \in \Gamma$ or $h{:}A \in \Sigma$ and $\Gamma;\ A \vdash \overline{M} : \Pi x{:}B_1.B_2$	given
$\Gamma \vdash \Pi x{:}B_1.B_2 : \mathit{type}$	by Regularity for Typing
$\Gamma \vdash B_1 : \mathit{type}$ and $\Gamma, x{:}B_1 \vdash B_2 : \mathit{type}$	by inversion for typing
$\Gamma;\ B_1 \vdash \cdot : B_1$	by rule (TERM-LIST-NIL)
$\Gamma, x{:}B_1 \vdash M_1 : B_1$	by IH 1 on B_1^-
$[M_1/x]_{B_1^-}B_2 = B_2$	by IH 2 on B_1^-

$\Gamma, x{:}B_1;\ A \vdash \overline{M} : \Pi x{:}B_1.B_2$ by Weakening for Typing

$\Gamma, x{:}B_1;\ A \vdash \overline{M_2} : B_2$ by Lemma 5.1.18

$\Gamma, x{:}B_1 \vdash M : B_2$ by IH 1 on B_2^-

$\Gamma \vdash \lambda x{:}B_1.M : \Pi x{:}B_1.B_2$ by rule (*TERM-LAM*)

2.

Case:

$$\mathcal{D} \quad = \quad \dfrac{\overset{\mathcal{D}_0}{\Gamma \vdash N : A_1} \qquad \overset{\mathcal{D}_1}{[N/y]_{A_1^-} A_2 = A_2'} \qquad \overset{\mathcal{D}_2}{\Gamma;\ A_2' \vdash \overline{N} : A_1'}}{\Gamma;\ \Pi y{:}A_1.A_2 \vdash N :: \overline{N} : A_1'}\ \textit{TERM-LIST-CONS}$$

$\mathtt{expand}_h(B; \cdot) = M$ given

$[M/x]_{B^-} N = N$ by IH 2 on B^- and N

$[M/x]_{B^-} \overline{N} = \overline{N}$ by IH 2 on B^- and $\overline{N}$

$[M/x]_{B^-} (N :: \overline{N}) = N :: \overline{N}$ by rule

Case:

$$\mathcal{D} \quad = \quad \dfrac{x{:}B \text{ in } \Gamma \qquad \overset{\mathcal{D}_0}{\Gamma;\ B \vdash \overline{M} : a\,\overline{N}}}{\Gamma \vdash x\,\overline{M} : a\,\overline{N}}\ \textit{TERM-ATM-V}$$

$\mathtt{expand}_x(B; \cdot) = M$

$[M/x]_{B^-} \overline{M} = \overline{M}$ by IH 2 on B^- and $\overline{M}$

$\Gamma;\ B \vdash \cdot : B$ by rule (*TERM-LIST-NIL*)

$\overline{M} @ \cdot = \overline{M}$ by rule

$\mathtt{reduce}_{B^-}(M, \overline{M}) = x\,\overline{M}$ by IH 3 on B^- and $\overline{M}$

$[M/x]_{B^-} x\,\overline{M} = x\,\overline{M}$ by rule

3.

Case:

$$\mathcal{D} = \dfrac{\begin{array}{cccc} \mathcal{D}_0 & & \mathcal{D}_1 & \mathcal{D}_2 \\ \mathtt{expand}_x(B_1;\cdot) = M_0 & x\#\overline{M_0} & \overline{M_0}@M_0 = \overline{M} & \mathtt{expand}_h(B_2;\overline{M}) = M \end{array}}{\mathtt{expand}_h(\Pi x{:}B_1.B_2;\overline{M_0}) = \lambda x{:}B_1.M}$$

$$\mathcal{E} = \dfrac{\begin{array}{ccc} \mathcal{E}_0 & \mathcal{E}_1 & \mathcal{E}_2 \\ \Gamma \vdash M_1 : B_1 & [M_1/x]_{B_1^-} B_2 = B_2' & \Gamma;\, B_2' \vdash \overline{M_1} : a\,\overline{N_0} \end{array}}{\Gamma;\, \Pi x{:}B_1.B_2 \vdash M_1 :: \overline{M_1} : a\,\overline{N_0}}\ \textit{TERM-LIST-CONS}$$

$$\mathcal{F} = \dfrac{\begin{array}{cc} \mathcal{F}_0 & \mathcal{F}_1 \\ \overline{M_0}@M_1 = \overline{M_0'} & \overline{M_0'}@\overline{M_1} = \overline{N} \end{array}}{\overline{M_0}@(M_1 :: \overline{M_1}) = \overline{N}}$$

$[M_1/x]_{B_1^-}\overline{M_0} = \overline{M_0}$	by Lemma 5.1.8
$\Gamma;\, B_1 \vdash \cdot : B_1$	by rule (*TERM-LIST-NIL*)
$\mathtt{reduce}_{B_1^-}(M_1,\cdot) = M_1$	by rule
$x\#B_1$	by the well-formedness of $\Pi x{:}B_1.B_2$
$[M_1/x]_{B_1^-}M_0 = M_1$	by IH 4 on B_1^- and Lemma 5.1.3
$[M_1/x]_{B_1^-}\overline{M} = \overline{M_0'}$	by Lemma 5.1.18 and Lemma 5.1.16
$\mathtt{expand}_h(B_2';\overline{M_0'}) = M'$ and $[M_1/x]_{B_1^-}M = M'$	by Lemma 5.1.23
$B_2'^- = B_2^-$	by Lemma 5.1.10
$\mathtt{reduce}_{B_2'^-}(M',\overline{M_1}) = h\,\overline{N}$	by IH 3 on $B_2'^-$ and $\overline{M_1}$
$\mathtt{reduce}_{B_1^- \to B_2^-}(\lambda x{:}B_1.M, M_1 :: \overline{M_1}) = h\,\overline{N}$	by rule

4.

Case:

$$\dfrac{\overset{\mathcal{D}_0}{\texttt{expand}_y(B_1;\cdot) = M_1} \quad y\#\overline{M} \quad \overset{\mathcal{D}_1}{\overline{M}@M_1 = \overline{M_2}} \quad \overset{\mathcal{D}_2}{\texttt{expand}_x(B_2;\overline{M_2}) = M}}{\texttt{expand}_x(\Pi y{:}B_1.B_2;\overline{M}) = \lambda y{:}B_1.M}$$

$\Gamma \vdash N : A$ and $\Gamma;\, A \vdash \overline{M} : \Pi y{:}B_1.B_2$

and $x\#A$ and $x\#\overline{M}$ and $x\#\Pi y{:}B_1.B_2$ — given

$\Gamma, y{:}B_1;\, B_1 \vdash \cdot : B_1$ — by rule (*TERM-LIST-NIL*)

$\Gamma, y{:}B_1 \vdash M_1 : B_1$ — by IH 1 on B_1^-

$\Gamma, y{:}B_1;\, A \vdash \overline{M} : \Pi y{:}B_1.B_2$ — by Weakening for Typing

$\Gamma \vdash A : \mathit{type}$ — by Regularity for Typing

$\Gamma \vdash \Pi y{:}B_1.B_2 : \mathit{type}$ — by Regularity for Typing

$\Gamma, y{:}B_1 \vdash B_2 : \mathit{type}$ — by inversion for typing

$[M_1/y]_{B_1^-} B_2 = B_2$ — by IH 2 on B_1^-

$\Gamma, y{:}B_1;\, A \vdash \overline{M_2} : B_2$ — by Lemma 5.1.19

$x\#y$ — by the renamability of bound variables

$x\#B_1$ and $x\#B_2$ — by def of fresh

$x\#M_1$ — by Lemma 5.1.22

$x\#\overline{M_2}$ — by Lemma 5.1.17

$\texttt{reduce}_{A^-}(N,\overline{M_2}) = M'$ and $[N/x]_{A^-} M = M'$ — by IH 4 on B_2^-

$\Gamma, y{:}B_1 \vdash N : A$ — by Weakening for Typing

$\texttt{reduce}_{A^-}(N,\overline{M}) = \lambda y{:}B_1.M''$

and $[M_1/x]_{B_1^-} M'' = M'$ — by Lemma 5.1.24

$\Gamma \vdash \lambda y{:}B_1.M'' : \Pi y{:}B_1.B_2$ — by Theorem 5.1.12 and Lemma 5.1.3

$[M_1/x]_{B_1^-} M'' = M''$ — by IH 2 on B_1^-

$M'' = M'$ by Lemma 5.1.3

$\texttt{reduce}_{A^-}(N, \overline{M_2}) = \lambda y{:}B_1.M'$ by equality

$[N/x]_{A^-} B_1 = B_1$ by Lemma 5.1.8

$[N/x]_{A^-} \lambda y{:}B_1.M = \lambda y{:}B_1.M'$ by rule

5.2 Logic Programming

At the beginning of this chapter, we sketched a search procedure for finding a derivation of a given judgment. In this section, we will formalize this idea by describing a procedure that finds LF terms of a given LF type.

5.2.1 Mixed Prefix Contexts and Generalized Substitutions

In order to define proof search, we need some way to model logic variables. One possibility would be to define logic variables using contextual modal type theory [NPP08], but this would involve a non-trivial extension to our spine calculus presentation of canonical LF. Instead, we model logic variables using ordinary LF variables, but we have to be careful. If y is considered a logic variable in a query of the form $\Gamma, y{:}B \vdash \boxed{?} : \Pi x{:}A_1.A_2$ (i.e. y is a placeholder for an as-of-yet undetermined LF term), then we expect proof search to proceed by solving the query $\Gamma, y{:}B, x{:}A_1 \vdash \boxed{?} : A_2$. But, unlike y, x is not intended to serve as a placeholder for anything; it is an actual variable that will be bound by a λ. In other words x and y are very different kinds of variables, and need to be treated as such. To this end, we express queries in terms of LF contexts that have been augmented with flags to distinguish the two notions of variable. Such augmented contexts are known as *mixed-prefix* contexts [Mil92b] and have previously been used in a setting similar to this one in [RP96, Roh96].

$$\textbf{Mixed-prefix Context} \quad \Delta \quad ::= \quad \cdot \mid \Delta, \forall x{:}A \mid \Delta, \exists x{:}A$$

Intuitively, existentially-quantified variables can be thought of as logic variables, and universally quantified variables can be thought of as actual variables. As with ordinary LF contexts, we assume that all variable declarations in a mixed prefix context are unique; this can typically be achieved by the tacit renaming of bound variables. Likewise, we assume that generalized substitutions act on any given variable at most once. A mixed-prefix context can be converted to an ordinary context using $|\Delta|$, and an ordinary context can be converted to a mixed-prefix context using $\exists\Gamma$ and $\forall\Gamma$.

$$\begin{array}{rcl}
|\cdot| & = & \cdot \\
|\Delta, \forall \mathrm{x}{:}\mathrm{A}| & = & |\Delta|, x{:}A \\
|\Delta, \exists \mathrm{x}{:}\mathrm{A}| & = & |\Delta|, x{:}A \\
\forall(\cdot) & = & \cdot \\
\forall(\Gamma, x{:}A) & = & (\forall\Gamma), \forall x{:}A \\
\exists(\cdot) & = & \cdot \\
\exists(\Gamma, x{:}A) & = & (\exists\Gamma), \exists x{:}A
\end{array}$$

We say that a mixed prefix context is *raised* if all of its existentially quantified variables occur at the beginning of the context. We formalize this concept, which will be useful later sections, with the judgment below.

$$\frac{}{\vdash \exists\Gamma \ \texttt{raised}} \qquad \frac{\vdash \Delta \ \texttt{raised}}{\vdash \Delta, \forall x{:}A \ \texttt{raised}}$$

The notion of logic variable instantiation is captured by the notion of generalized substitution, defined below.

$$\textbf{Generalized Substitutions} \quad \theta \quad ::= \quad \cdot \mid \theta, y/x \mid \theta, M{:}A/x$$

Generalized substitutions can be applied to LF variables, terms, spines, type families and kinds. Applying generalized substitutions to LF variables yields either a variable (in the case of universally quantified variables), or a term with its type (in the case of existentially quantified variables). We present the defining equations for generalizes substitution below. Note that, as usual, these equations should be viewed as being nothing more than a more compact notation for inference rules. Although generalized substitution application is decidable, it may, in general, be undefined.

$$\begin{array}{rcll}
(\theta, M{:}A/x)\, x &=& M{:}A & \\
(\theta, M{:}A/x)\, y &=& (\theta)y & \text{if } y \neq x \\
(\theta, y/x)\, x &=& y & \\
(\theta, y/x)\, x' &=& (\theta)x' & \text{if } x' \neq x \\
\theta\,(c\,\overline{M}) &=& c\,\overline{N} & \text{if } \theta\overline{M} = \overline{N} \\
\theta\,(x\,\overline{M}) &=& \begin{cases} y\,\overline{N} & \text{if } \theta\, x = y \text{ and } \theta\overline{M} = \overline{N} \\ \texttt{reduce}_{A^-}(M, \overline{N}) & \text{if } \theta\, x = M{:}A \text{ and } \theta\overline{M} = \overline{N} \end{cases} & \\
\theta\,(\lambda x{:}A.M) &=& \lambda y{:}B.N & \text{if } \theta A = B \text{ and } (\theta, y/x)M = N \text{ and } y \# \theta \\
\theta\,(\cdot) &=& \cdot & \\
\theta(M :: \overline{M}) &=& N :: \overline{N} & \text{if } \theta M = N \text{ and } \theta\overline{M} = \overline{N} \\
\theta\,(a\,\overline{M}) &=& a\,\overline{N} & \text{if } \theta\overline{M} = \overline{N} \\
\theta\,(\Pi x{:}A.B) &=& \Pi y{:}A'.B' & \text{if } \theta A = A' \text{ and } (\theta, y/x)B = B' \text{ and } y \# \theta \\
\theta\,(\mathit{type}) &=& \mathit{type} & \\
\theta\,(\Pi x{:}A.K) &=& \Pi y{:}A'.K' & \text{if } \theta A = A' \text{ and } (\theta, y/x)K = K' \text{ and } y \# \theta
\end{array}$$

The typing rules for generalized substitutions are defined below.

$$\frac{}{\cdot \vdash \cdot : \cdot}\ \textit{gsub-nil} \qquad \frac{\Delta' \vdash \theta : \Delta \quad \theta A = A' \quad |\Delta'| \vdash A' : \textit{type}}{\Delta', \forall y{:}A' \vdash \theta, y/x : \Delta, \forall x{:}A}\ \textit{gsub-avar}$$

$$\frac{\Delta' \vdash \theta : \Delta \quad \theta A = A' \quad |\Delta'| \vdash M : A'}{\Delta' \vdash \theta, M{:}A'/x : \Delta, \exists x{:}A}\ \textit{gsub-evar}$$

$$\frac{\Delta' \vdash \theta : \Delta \quad |\Delta'| \vdash A : \textit{type}}{\Delta', \exists x{:}A \vdash \theta : \Delta}\ \textit{gsub-weak}$$

We proceed by proving some useful properties of generalized substitutions.

Lemma 5.2.1 (Typing Inversion for Generalized Substitutions)

1. *If* $\Delta' \vdash \theta : \Delta, \forall x{:}A$ *then* $\theta = \theta_0, y/x$ *and* $\Delta' = \Delta'_0, \forall y{:}A', \exists\Gamma$ *and* $\Delta'_0 \vdash \theta_0 : \Delta$ *and* $\theta_0 A = A'$

2. *If* $\Delta' \vdash \theta, y/x : \Delta$ *then* $\Delta = \Delta_0, \forall x{:}A$ *and* $\Delta' = \Delta'_0, \forall y{:}A', \exists\Gamma$ *and* $\Delta'_0 \vdash \theta : \Delta_0$ *and* $\theta_0 A = A'$ *and* $|\Delta'_0| \vdash A' : \textit{type}$

3. *If* $\Delta', \forall y{:}A' \vdash \theta : \Delta$ *then* $\theta = \theta_0, y/x$ *and* $\Delta = \Delta_0, \forall x{:}A$ *and* $\Delta' \vdash \theta_0 : \Delta_0$ *and* $\theta_0 A = A'$ *and* $|\Delta'| \vdash A' : \textit{type}$.

4. *If* $\Delta' \vdash \theta : \Delta, \exists x{:}A$ *then* $\theta = \theta_0, M{:}A'/x$ *and* $\Delta' \vdash \theta : \Delta$ *and* $\theta_0 A = A'$ *and* $|\Delta'| \vdash M : A'$

5. *If* $\Delta' \vdash \theta, M{:}A/x : \Delta$ *then* $\Delta = \Delta_0, \exists x{:}A$ *and* $\Delta' \vdash \theta : \Delta_0$ *and* $|\Delta'| \vdash M : A$

6. *If* $\Delta', \exists y{:}A \vdash \theta : \Delta$ *then* $\Delta' \vdash \theta : \Delta$ *and* $|\Delta'| \vdash A : \textit{type}$

7. *If* $\Delta' \vdash \theta : \cdot$ *then* $\theta = \cdot$ *and* $\Delta' = \exists\Gamma$

8. *If* $\Delta' \vdash \cdot : \Delta$ *then* $\Delta = \cdot$ *and* $\Delta' = \exists\Gamma$

9. *If* $\cdot \vdash \theta : \Delta$ *then* $\Delta = \cdot$ *and* $\theta = \cdot$

Proof: 1, 2, 4, 5, 7 and 8 are by straightforward inductions on the structures of the given typing derivations; the others are direct, by cases. □

Lemma 5.2.2 (Properties of Generalized Substitutions) *Let E be any of $\{K, A, M, \overline{M}\}$*

1. *If $\theta E = E'$ and $\theta E = E''$ then $E' = E''$*

2. *If $(\theta, X/x, Y/y)E = E'$ and $x\#y$, where $X = x'$ or $X = M{:}A$ and $Y = y'$ or $y = N{:}A$ then $(\theta, Y/y, X/x)E = E'$*

3. *If $x\#E$ and either $(\theta, y/x)E = E'$ or $(\theta, M{:}A/x)E = E'$ then $\theta E = E'$*

4. *If the free variables of E are among $x_1, \ldots, x_n$, then $(\theta, x_1/x_1, \ldots, x_n/x_n)E = E$*

5. *If $x\#\theta$ and $\theta E = E'$ then $x\#E'$*

Proof: By straightforward inductions over the given derivations of generalized substitution application. Note that 4 can be described judgmentally, but that we use the more informal ellipses for the sake of readability. □

Lemma 5.2.3 (Generalized and Hereditary Substitutions Commute) *Let E be any of $\{K, A, M, \overline{M}\}$*

1. *If $\mathcal{D} : [M/x]_\tau E = E'$ and $(\theta, y/x)E = E_0$ and $\theta M = M_0$ then there exists some E_0' s.t. $[M_0/y]_\tau E_0 = E_0'$ and $\theta E' = E_0'$*

2. *If $\mathcal{D}$: $\texttt{reduce}_\tau(M, \overline{N}) = N'$ and $\theta M = M_0$ and $\theta\overline{N} = \overline{N_0}$ then there exists some N_0' s.t. $\texttt{reduce}_\tau(M_0, \overline{N_0}) = N_0'$ and $\theta N' = N_0'$.*

Proof: By mutual induction $\mathcal{D}$. We show some representative cases below.

1.

Case:

$$\mathcal{D} \;=\; \dfrac{\begin{array}{cc}\mathcal{D}_0 & \mathcal{D}_1\\ [M/x]_\tau\overline{N} = \overline{N'} & \texttt{reduce}_\tau(M, \overline{N'}) = N'\end{array}}{[M/x]_\tau x\,\overline{N} = N'}$$

$(\theta, y/x)\,(x\,\overline{N}) = y\,\overline{N_0}$ and $(\theta, y/x)\,M = M_0$ given

$(\theta, y/x)\,\overline{N} = \overline{N_0}$ by inversion on gsub application

$\texttt{reduce}_\tau(M_0, \overline{N_0}) = N_0'$ and $\theta N' = N_0'$ by IH 2 on $\mathcal{D}_1$

$[M_0/y]_\tau y\,\overline{N_0} = N_0'$ by rule

Case:

$$\mathcal{D} \;=\; \dfrac{\begin{array}{ccc}\mathcal{D}_0 & \mathcal{D}_1 & \\ [M/x]_\tau A = A' & [M/x]_\tau B = B' & x\#z\end{array}}{[M/x]_\tau \Pi z{:}A.B = \Pi z{:}A'.B'}$$

$(\theta, y/x)\,\Pi z{:}A.B = \Pi z_0{:}A_0.B_0$ and $(\theta, y/x)M = M_0$ given

$(\theta, y/x)A = A_0$ and $(\theta, y/x, z_0/z)B = B_0$

by inversion on gsub application

$[M_0/y]_\tau A_0 = A_0'$ and $\theta A' = A_0'$ by IH 1 on $\mathcal{D}_0$

$(\theta, z_0/z, y/x)B = B_0$ by Lemma 5.2.2

$[M_0/y]_\tau B_0 = B_0'$ and $(\theta, z_0/z)B' = B_0'$ by IH 1 on $\mathcal{D}_1$

$[M_0/y]_\tau \Pi z_0{:}A_0.B_0 = \Pi z_0{:}A_0'.B_0'$ by rule

$\theta\Pi z{:}A'.B' = \Pi z_0{:}A_0'.B_0'$ by rule

2.

Case:

$$\mathcal{D} = \dfrac{\overset{\mathcal{D}_0}{[N/x]_\tau M = M'} \qquad \overset{\mathcal{D}_1}{\texttt{reduce}_\beta(M', \overline{N}) = N'}}{\texttt{reduce}_{\tau\to\beta}(\lambda x{:}A.M, N :: \overline{N}) = N'}$$

$\theta(\lambda x{:}A.M) = \lambda y{:}A_0.M_0$ and $\theta(N :: \overline{N}) = N_0 :: \overline{N_0}$ given

$\theta A = A_0$ and $(\theta, y/x)M = M_0$ and $\theta N = N_0$ and $\theta\overline{N} = \overline{N_0}$

by inversion on gsub application

$[N_0/x]_\tau M_0 = M_0'$ and $\theta M' = M_0'$ by IH 1 on $\mathcal{D}_0$

$\texttt{reduce}_\tau(M_0', N_0) = N_0'$ and $\theta N' = N_0'$ by IH 2 on $\mathcal{D}_1$

$\texttt{reduce}_{\tau\to\beta}(\lambda y{:}A_0.M_0, N_0 :: \overline{N_0}) = N_0'$ by rule

□

Lemma 5.2.4 (Generalized Substitutions are Simultaneous) *If $A^- = \tau$ and E is any of $\{K, A, \overline{M}, M\}$ $(\theta, M'{:}A/x)E_0 = E'$ and $(\theta, y/x)E_0 = E$ then $[M'/y]_\tau E = E'$*

Proof: By straightforward induction on the structure of E_0, using Lemma 5.2.3 clause 2. □

Lemma 5.2.5 (Weakening for Generalized Substitutions) *For E in $\{K, A, \overline{M}, M\}$, if $y\#E$ and $\theta E = E'$ then $(\theta, y/x)E = E'$ and $(\theta, M{:}B/x)E = E'$*

Proof: By a straightforward induction on the derivation of $\theta E = E'$; note that $x\#\theta$ must hold in order for $\theta, y/x$ and $\theta, M{:}A/x$ to be well formed. □

Lemma 5.2.6 (Generalized Substitutions on Variables) *If $\mathcal{D} : \Delta' \vdash \theta : \Delta$ then:*

1. *If $\forall x{:}A \in \Delta$ then $\theta x = y$ and $\theta A = B$ and $\forall y{:}B \in \Delta'$*

2. *If* $\exists x{:}A \in \Delta$ *then* $\theta x = M{:}B$ *and* $\theta A = B$ *and* $|\Delta'| \vdash M : B$

3. *If* $\forall x{:}A \in \Delta$ *and* $\forall y{:}B \in \Delta$ *and* $x \# y$ *then* $\theta x \# \theta y$

Proof: Each follows from a straightforward induction on the structure of $\mathcal{D}$, using Lemma 5.2.5 for both 1 and 2, and Weakening for Typing in 2. 3 uses the fact that mixed prefix contexts cannot contain duplicate variable-name declarations. □

Lemma 5.2.7 (Generalized Substitutions) *If* $\Delta' \vdash \theta : \Delta$ *then:*

1. *if* $\vdash |\Delta| : \texttt{ctx}$ *then* $\vdash |\Delta'| : \texttt{ctx}$

2. *if* $\mathcal{D} : |\Delta| \vdash K : \mathit{kind}$ *then* $\theta K = K'$ *and* $|\Delta'| \vdash A' : \mathit{kind}$

3. *if* $\mathcal{D} : |\Delta|;\, A \vdash \overline{M} : \mathit{type}$ *and* $\theta A = A'$ *then* $\theta\overline{M} = \overline{M'}$ *and* $|\Delta'|;\, A' \vdash \overline{M'} : \mathit{type}$

4. *if* $\mathcal{D} : |\Delta| \vdash A : \mathit{type}$ *then* $\theta A = A'$ *and* $|\Delta'| \vdash A' : \mathit{type}$

5. *if* $\mathcal{D} : |\Delta|;\, A \vdash \overline{M} : B$ *and* $\theta A = A'$ *then* $\theta\overline{M} = \overline{M'}$ *and* $\theta B = B'$ *and* $|\Delta'|;\, A' \vdash \overline{M'} : B'$

6. *if* $\mathcal{D} : |\Delta| \vdash M : A$ *then* $\theta M = M'$ *and* $\theta A = A'$ *and* $|\Delta'| \vdash M' : A'$

Proof: 1 is by induction on the structure of $\Delta' \vdash \theta : \Delta$; 2-6 are by mutual inductions on the structure of $\mathcal{D}$. The non-trivial *TERM-ATM-V* case uses Theorem 5.1.12; the *FAM-LIST-CONS* and *TERM-LIST-CONS* cases use Lemma 5.2.3. □

Definition 5.2.8 (Composition of Generalized substitutions) *Generalized substitutions can be composed as follows.*

$$\begin{array}{rcll}
\theta \circ \cdot & = & \theta & \\
(\theta, z/y) \circ (\theta', y/x) & = & (\theta \circ \theta'), z/x & \\
(\theta, M{:}A/z) \circ (\theta', y'/x) & = & \theta' \circ (\theta', y/x) & \text{if } y \# z \text{ and } \theta' y = y' \\
\theta \circ (\theta', M{:}A/x) & = & (\theta \circ \theta'), N{:}B/x & \text{if } \theta M = N \text{ and } \theta A = B
\end{array}$$

Lemma 5.2.9 (Composition of Generalized Substitutions) *Let E be any of $\{K, A, \overline{M}, M\}$*

1. *If $\theta_2 \circ \theta_1 = \theta$ then*
   (a) *$\theta_2(\theta_1\ x) = y$ iff $\theta x = y$*
   (b) *$\theta_2(\theta_1\ x) = M{:}A$ iff $\theta x = M{:}A$*
   (c) *$\theta_2(\theta_1 E) = E'$ iff $\theta E = E'$ where E is any of $\{K, A, M, \overline{M}\}$*
2. *If $\Delta'' \vdash \theta_2 : \Delta'$ and $\Delta' \vdash \theta_1 : \Delta$ then there exists some θ s.t. $\theta_2 \circ \theta_1 = \theta$ and $\Delta'' \vdash \theta : \Delta$*
3. *If $\theta_0 \circ \theta_1 = \theta$ and $\theta_0 \circ \theta_1 = \theta'$ then $\theta = \theta'$*

Proof: 1 (a) and (b) follow by a straightforward induction on the derivation of $\theta_2 \circ \theta_1 = \theta$, whereas 1 (c) follows by induction on E. 2 follows by induction on the natural sum of θ_1 and θ_2, using Lemma 5.2.1. 3 is by straightforward induction on the derivation of $\theta_0 \circ \theta_1$. $\square$

Lemma 5.2.10 (Generalized Substitutions and Eta)

1. *If $\mathcal{D} : \overline{M}@M = N$ and $\theta\overline{M} = \overline{M'}$ and $\theta M = M'$ then there exists N' s.t. $\overline{M'}@M' = N$ and $\theta N = N'$*
2. *If $\mathcal{D} : \texttt{expand}_h(A; \overline{M}) = N$ and $x \# h$ $x \# A$ and $x \# \overline{M}$ then $x \# M$*
3. *If $\mathcal{D} : \texttt{expand}_x(A; \overline{M}) = M_0$ and $\theta x = y$ and $\theta A = B$ and $\theta\overline{M} = \overline{N}$ then there exists an N_0 s.t. $\theta M_0 = N_0$ and $\texttt{expand}_y(B; \overline{N}) = N_0$*

Proof: By straightforward induction on the structure of $\mathcal{D}$. 1 and 2 are self-contained (i.e. the induction is not mutual), whereas 3 uses 1 and 2. We show the induction step for 3 below.

$$\mathcal{D} = \dfrac{\overset{\mathcal{D}_0}{\mathtt{expand}_z(A_1;\cdot) = M} \quad z\#\overline{M} \quad \overset{\mathcal{D}_1}{\overline{M}@M = \overline{M'}} \quad \overset{\mathcal{D}_2}{\mathtt{expand}_x(A_2;\overline{M'}) = M_0}}{\mathtt{expand}_x(\Pi z{:}A_1.A_2;\overline{M}) = \lambda z{:}A_1.M_0}$$

$\theta x = y$ and $\theta(\Pi z{:}A_1.A_2) = \Pi z'{:}B_1.B_2$ and $\theta\overline{M} = \overline{N}$	given
$x\#z$	by renamability of bound variables
$\theta A_1 = B_1$ and $(\theta, z'/z)A_2 = B_2$ and $z'\#\theta$	by inversion on gsub app
$z'\#\overline{N}$	by Lemma 5.2.2 no. 5
$\theta\cdot = \cdot$	by rule
$\mathtt{expand}_x(B_1;\cdot) = N$ and $\theta M = N$	by IH 3 on $\mathcal{D}_0$
$\overline{N}@N = \overline{N'}$ and $\theta\overline{M'} = \overline{N'}$	by IH 1 on $\mathcal{D}_1$
$(\theta, z'/z)M_0 = N_0$ and $\mathtt{expand}_x(B_2;\overline{N'}) = N_0$	by IH 3 on $\mathcal{D}_1$
$(\theta)\lambda z{:}A_1.M_0 = \lambda z'{:}B_1.N_0$	by def of gsub app
$\mathtt{expand}_x(\Pi z'{:}B_1.B_2;\overline{N}) = \lambda z'{:}B_1.N_0$	by rule

□

5.2.2 The Semantics of Logic Programming

Given LF terms M and N that are well-formed in the mixed-prefix context Δ, it is natural to ask whether there is some instantiation in the logic variables of Δ such that M and N are indistinguishable. We refer to such a problem as a *unification equation*, which we write $\Delta \vdash M \overset{\bullet}{=} N$. The solution to such an equation is a *unifier*, which we define below. Unification will play an important role in defining proof search.

Definition 5.2.11 (Unifiers) *$\Delta' \vdash \theta : \Delta$ is a* unifier *(or* solution*) for the unifica-*

tion equation $\Delta \vdash E \stackrel{\bullet}{=} E'$ *(for any* E *in* $\{A, M, \overline{M}, K\}$*) iff* $\theta E = E''$ *and* $\theta E' = E''$ *and* E *is well-formed in* $|\Delta|$. *A unifier* $\Delta' \vdash \theta : \Delta$ *is a* most general unifier *for the equation* $\Delta \vdash E \stackrel{\bullet}{=} E'$ *iff, for all unifiers* $\Delta'' \vdash \theta' : \Delta$*, there exists* $\Delta'' \vdash \theta'' : \Delta'$ *s.t.* $\theta'' \circ \theta = \theta'$. *We sometimes refer to* θ *as being a (most general) unifier when* θ*'s type is clear from the context.*

Unfortunately, unification is an undecidable problem for LF terms, and the existence of a solution to a unification equation does not guarantee the existence of a most general unifier. However, there are algorithms that can solve a decidable subset of the unification equations for LF, known as *higher-order pattern matching* for which the existence of a unifier implies the existence of a most general unifier[Mil91]. We assume that we are given the judgmental specification of such an algorithm, written here as $\texttt{unify}(\Delta \vdash E \stackrel{\bullet}{=} E') \stackrel{\Delta'}{\Longrightarrow} \theta$ when the algorithm succeeds in finding a most general unifier $\Delta' \vdash \theta : \Delta$ for E and E', and $\texttt{unify}(\Delta \vdash E \stackrel{\bullet}{=} E') \Longrightarrow \texttt{fail}$ when the algorithm fails to find such a unifier. We view the following proposition as part of the specification of unification. Its proof should be syntactically finitary.

Proposition 5.2.12 (Decidability, Well-Formedness of Unification) *For every* E *in* $\{M, \overline{M}, A, K\}$

1. *either* $\texttt{unify}(\Delta \vdash E \stackrel{\bullet}{=} E') \stackrel{\Delta'}{\Longrightarrow} \theta$ *or* $\texttt{unify}(\Delta \vdash E \stackrel{\bullet}{=} E') \Longrightarrow \texttt{fail}$
2. *If* $\texttt{unify}(\Delta \vdash E \stackrel{\bullet}{=} E') \stackrel{\Delta'}{\Longrightarrow} \theta$ *then* $\Delta' \vdash \theta : \Delta$ *and* $\vdash \Delta'$ $\texttt{raised}$ *and* θ *is a (most general)*[2] *unifier for* E *and* E'.

It should be noted that, although we do not require the unification algorithm to be complete (indeed, no such algorithm exists), we do expect it to be nontrivial in

[2]The fact that θ is most general will not be used in any of our theorems; thus, we do not have to worry about the quantifier alternations introduced by the notion of most general unifier taking us out of the realm of the syntactically finitary.

that it is capable of deciding an interesting subset of unification problems, such as higher-order pattern matching.

Below we introduce the notion of proof search on LF terms as a model of computation. That is, given a mixed-prefix context Δ and a *goal type* A (possibly containing some logic variables), is it possible to find a term M and a generalized substitution θ such that $\Delta' \vdash \theta : \Delta$ and $\Delta' \vdash M : \theta A$? Such a search problem is referred to as a *query*, which we write $\Delta \vdash \boxed{?} : A$. In practice, queries performed on atomic-types are the most interesting. This is because family-level constants can be thought of as defining relations, where the terms $M_1, \ldots, M_n$ are "related" whenever $a\, M_1\, \ldots\, M_n$ is inhabited. If we consider some elements of these relations to be *inputs* and others to be *outputs*, LF type-families can provide convenient notations for functions. We refer to the classification of the input and output arguments for a family-level constant as its *mode declaration.* Mode declarations are provided by the user (i.e. at the same level as the declaration of the signature Σ and the subordination relation $\sqsubseteq$), and every family-level constant that we wish to interpret as a logic program must be given such a classification. Arguments that are classified as inputs are said to have *positive mode* and arguments that are classified as outputs are said to have *negative mode.*

More precisely, if $a{:}K \in \Sigma$, then K must have the form $\Pi x_0{:}A_0 \ldots \Pi x_n{:}A_n.\mathit{type}$; a mode declaration for a consists of a user-level assignment of *positive* or *negative* to each i in $0, \ldots, n$ with the following restriction: if i has positive mode and j has negative mode, then $x_j \# A_i$ (that is, the types of inputs cannot depend on the types of outputs). If a has a mode declaration, and the atomic type $a\,\overline{M}$ is well formed, then $\overline{M} = M_0 :: \ldots :: M_n :: \cdot$. We can then partition the spine $\overline{M}$ into two spines, $\texttt{inputs}_a(\overline{M})$ and $\texttt{outputs}_a(\overline{M})$, where M_i is in $\texttt{inputs}_a(\overline{M})$ iff i is has positive mode, and M_i is in $\texttt{outputs}_a(\overline{M})$ iff i has negative mode. Formally, we consider $\texttt{inputs}$

and `outputs` to be user-specified syntactically finitary functions. It should be noted that the fact that input arguments cannot depend on out put arguments guarantees that both $\mathtt{inputs}_a(\overline{M})$ and $\mathtt{outputs}_a(\overline{M})$ are well-formed whenever $\overline{M}$ is. Thus, we assume the following properties hold.

Lemma 5.2.13 (Inputs and Outputs)

1. $\mathtt{inputs}_a(\overline{M}) = \mathtt{inputs}_a(\overline{N})$ *and* $\mathtt{outputs}_a(\overline{N}) = \mathtt{outputs}_a(\overline{M})$ *iff* $\overline{N} = \overline{M}$

2. *If* $\theta\overline{M} = \overline{N}$ *then* $\theta(\mathtt{inputs}_a(\overline{M})) = \mathtt{inputs}_a(\overline{N})$ *and* $\theta(\mathtt{outputs}_a(\overline{M})) = \mathtt{outputs}_a(\overline{N})$

Proof: Direct, by the definitions of `inputs` and `outputs`. $\square$

It should be noted that it is possible for a mode declaration to be invalid; that is, the term-level constants in Σ may not respect the mode declarations given by the user. We postpone discussing mode checking until section 5.2.5.

We are finally ready to define the semantics of proof search. The judgment $\mathtt{fillTerm}(\Delta_0 \vdash \boxed{?} : A_0) \overset{\theta}{\Longrightarrow} \langle\Delta;\, M;\, A\rangle$ formalizes the notion of search for terms: it can be thought of as being recursive on the structure of A_0, where in the base case—i.e. $A_0 = a\,\overline{N}$—it nondeterministically chooses either a variable or constant whose type has head a, then searches for a spine. The judgment $\mathtt{fillHead}(\Delta; \Sigma; a) \Longrightarrow h{:}A$ formalizes the nondeterministic choice of a variable or constant whose type A satisfies $\mathtt{hd}(A) = a$: it can be thought of as being recursive on the structure of Σ and Δ. The judgment $\mathtt{fillSpine}(\Delta_0; A_0 \vdash \boxed{?} : a\,\overline{N_0}) \overset{\theta}{\Longrightarrow} \langle\Delta;\, A;\, \overline{M};\, a\,\overline{N}\rangle$ the formalizes notion of search for spines: it can be thought of as being recursive on the structure of A_0 For each of these judgments, the syntactic categories to the left of the arrow can be thought of as inputs, and everything else as outputs. The intuition behind each judgment is made more precise by Lemma 5.2.14 below. In essence, we are formalizing the logic-programming interpretation of judgments and inference rules (here implemented as LF types and terms) using judgments and inference rules that, intuitively, can be interpreted as logic programs. This sort of circularity is inevitable in all foundational investigations, and ultimately must be justified by intuition.

$$\frac{\texttt{unify}(\Delta_0 \vdash \texttt{inputs}_a(\overline{N_1}) \stackrel{\bullet}{=} \texttt{inputs}_a(\overline{N_0})) \overset{\Delta}{\Longrightarrow} \theta \quad \theta(a\,\overline{N_1}) = a\,\overline{N}}{\texttt{fillSpine}(\Delta_0; a\,\overline{N_1} \vdash \boxed{?} : a\,\overline{N_0}) \overset{\theta}{\Longrightarrow} \langle \Delta;\ a\,\overline{N};\ \cdot\ ;\ a\,\overline{N}\rangle}\ \textit{fs-atm}$$

$$\frac{\texttt{fillSpine}(\Delta_0, \exists x_0{:}A_0; B_0 \vdash \boxed{?} : a\,\overline{N_0}) \overset{\theta, M:A/x_0}{\Longrightarrow} \langle \Delta;\ B';\ \overline{M};\ a\,\overline{N}\rangle \quad (\theta, x/x_0)B_0 = B}{\texttt{fillSpine}(\Delta_0; \Pi x_0{:}A_0.B_0 \vdash \boxed{?} : a\,\overline{N_0}) \overset{\theta}{\Longrightarrow} \langle \Delta;\ (\Pi x{:}A.B);\ (M :: \overline{M});\ a\,\overline{N}\rangle}\ \textit{fs-pi}$$

$$\frac{\begin{array}{c}\texttt{fillSpine}(\Delta; B_0 \vdash \boxed{?} : a\,\overline{N_0}) \overset{\theta}{\Longrightarrow} \langle \Delta;\ B;\ \overline{M};\ a\,\overline{N}\rangle \quad \theta A_0 = A \\ \texttt{fillTerm}(\Delta \vdash \boxed{?} : A) \overset{\theta'}{\Longrightarrow} \langle \Delta';\ M';\ A'\rangle \quad \theta'\overline{M} = \overline{M'} \quad \theta' B = B' \quad \theta' a\,\overline{N} = a\,\overline{N'}\end{array}}{\texttt{fillSpine}(\Delta_0; A_0 \to B_0 \vdash \boxed{?} : a\,\overline{N_0}) \overset{\theta'\circ\theta}{\Longrightarrow} \langle \Delta';\ A' \to B';\ (M' :: \overline{M'});\ a\,\overline{N'}\rangle}\ \textit{fs-arr}$$

$$\frac{\forall x{:}A \in \Delta \quad \texttt{hd}(A) = a}{\texttt{fillHead}(\Delta; \Sigma'; a) \Longrightarrow x{:}A}\ \textit{fh-v} \qquad \frac{c{:}A \in \Sigma' \quad \texttt{hd}(A) = a}{\texttt{fillHead}(\Delta; \Sigma'; a) \Longrightarrow c{:}A}\ \textit{fh-c}$$

$$\frac{\begin{array}{c}\texttt{fillHead}(\Delta; \Sigma; a) \Longrightarrow h{:}A_0 \quad \texttt{fillSpine}(\Delta_0; A_0 \vdash \boxed{?} : a\,\overline{N_0}) \overset{\theta}{\Longrightarrow} \langle \Delta;\ A;\ \overline{M};\ a\,\overline{N}\rangle \\ \theta(a\,\overline{N_0}) = a\,\overline{N'} \quad \texttt{unify}(\Delta \vdash \texttt{outputs}_a(\overline{N}) \stackrel{\bullet}{=} \texttt{outputs}_a(\overline{N'})) \overset{\Delta'}{\Longrightarrow} \theta' \\ \theta'\overline{M} = \overline{M'} \quad \theta'\overline{N'} = \overline{N''}\end{array}}{\texttt{fillTerm}(\Delta_0 \vdash \boxed{?} : a\,\overline{N_0}) \overset{\theta'\circ\theta}{\Longrightarrow} \langle \Delta';\ h\,\overline{M'};\ a\,\overline{N''}\rangle}\ \textit{ft-atm}$$

$$\frac{\texttt{fillTerm}(\Delta_0, \forall x_0{:}A_0 \vdash \boxed{?} : B_0) \overset{\theta, x/x_0}{\Longrightarrow} \langle \Delta, \forall x{:}A;\ M;\ B\rangle}{\texttt{fillTerm}(\Delta_0 \vdash \boxed{?} : \Pi x_0{:}A_0.B_0) \overset{\theta}{\Longrightarrow} \langle \Delta;\ \lambda x{:}A.M;\ \Pi x{:}A.B\rangle}\ \textit{ft-pi}$$

We think of the rule *fs-pi* as having the implicit side condition that x occurs freely within B_0 (i.e. $\Pi x{:}A.B$ cannot be written as $A \to B$), which we omit both for the sake of brevity. In essence, the difference between *fs-pi* and *fs-arr* is the difference between creating a subgoal to be found via proof search, and creating a logic variable to be instantiated via unification; these rules are inspired by the Πl and $\to l$ rules of the system U of [PW90].

It should be noted that the above semantics for proof search is *mode aware*; that is, the mode declarations given by the user affect the semantics of proof search. This is a departure from what is actually implemented in other accounts of Logic Programming based on LF [PW90, Pfe91, RP96, PS98], where unification would be invoked only in the *fs-atm* rule, and on all of $\overline{N_0}$ and $\overline{N_1}$ rather than just the input arguments. This simplifies the development of the metatheory of logic programming: to do otherwise would require us to specify a unification algorithm that postpones unsolvable unification problems as constraints, which would, in order to prove the analog of Lemma 5.2.14, require a development of LF where the typing rules are parameterized by equality constraints (see [Ree09] for a detailed sketch of what would be involved). The mode-awareness of our semantics does not meaningfully affect the execution of any of the programs considered in this dissertation.

Lemma 5.2.14 (Soundness of Filling)

1. *If* $\mathcal{D} : \texttt{fillSpine}(\Delta_0; A_0 \vdash \overline{\boxed{?}} : a\,\overline{N_0}) \overset{\theta}{\Longrightarrow} \langle \Delta;\ A;\ \overline{M};\ a\,\overline{N}\rangle$ *and* $|\Delta_0| \vdash A_0 : \mathit{type}$ *and* $|\Delta_0| \vdash a\,\overline{N_0} : \mathit{type}$ *then* $\Delta \vdash \theta : \Delta_0$ *and* $\vdash \Delta\ \texttt{raised}$ *and* $\theta A_0 = A$ *and* $\theta(\texttt{inputs}_a(\overline{N_0})) = \texttt{inputs}_a(\overline{N})$ *and* $|\Delta|;\ A \vdash \overline{M} : a\,\overline{N}$
2. *If* $\mathcal{D} : \texttt{fillTerm}(\Delta_0 \vdash \boxed{?} : A_0) \overset{\theta}{\Longrightarrow} \langle \Delta;\ M;\ A\rangle$ *and* $|\Delta_0| \vdash A_0 : \mathit{type}$ *then* $\Delta \vdash \theta : \Delta_0$ *and* $\vdash \Delta\ \texttt{raised}$ *and* $\theta A_0 = A$ *and* $|\Delta| \vdash M : A$

Proof: By mutual inductions on the structure of $\mathcal{D}$. We show some representative cases below.

Case:

$$\mathcal{D} = \cfrac{\overset{\mathcal{D}_0}{\texttt{fillSpine}(\Delta_0, \exists x_0{:}A_0; B_0 \vdash \overline{\boxed{?}} : a\,\overline{N_0}) \overset{\theta, M:A/x_0}{\Longrightarrow} \langle \Delta;\ B';\ \overline{M};\ a\,\overline{N}\rangle} \quad (\theta, x/x_0)B_0 = B}{\texttt{fillSpine}(\Delta_0; \Pi x_0{:}A_0.B_0 \vdash \overline{\boxed{?}} : a\,\overline{N_0}) \overset{\theta}{\Longrightarrow} \langle \Delta;\ (\Pi x{:}A.B);\ (M :: \overline{M});\ a\,\overline{N}\rangle}\ \textit{fs-pi}$$

$|\Delta_0| \vdash \Pi x{:}A_0.B_0 : \mathit{type}$ and $|\Delta_0| \vdash a\,\overline{N_0} : \mathit{type}$ given

$|\Delta_0| \vdash A_0 : \mathit{type}$ and $|\Delta_0|, x_0{:}A_0 \vdash B_0 : \mathit{type}$ by inversion for typing

$|\Delta_0|, x_0{:}A_0 \vdash a\,\overline{N_0} : \mathit{type}$ by weakening for typing

$\Delta \vdash (\theta, M{:}A/x_0) : \Delta_0, \exists x_0{:}A_0$ and $\vdash \Delta$ `raised`

and $(\theta, M{:}A/x_0)B_0 = B'$ and $(\theta, M{:}A/x_0)\mathtt{inputs}_a(\overline{N_0}) = \mathtt{inputs}_a(\overline{N})$

and $|\Delta|;\, B' \vdash \overline{M} : a\,\overline{N}$ by IH on $\mathcal{D}_0$

$|\Delta| \vdash M : A$ and $\theta A_0 = A$ and $\Delta \vdash \theta : \Delta_0$ by Lemma 5.2.1

$\theta\,(\Pi x_0{:}A_0.B_0) = \Pi x{:}A.B$ by rule

$x \# a\,\overline{N_0}$ by Lemma 5.1.5

$\theta(\mathtt{inputs}_a(\overline{N_0})) = \mathtt{inputs}_a(\overline{N})$ by Lemma 5.2.2, clause 3

$[M/x]_{A^-}B = B'$ by Lemma 5.2.4 and Lemma 5.2.2 clause 1

$|\Delta|;\, \Pi x{:}A.B \vdash M :: \overline{M} : a\,\overline{N}$ by rule

Case:

$$\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_0\\ \mathtt{fillTerm}(\Delta_0, \forall x_0{:}A_0 \vdash \boxed{?} : B_0) \overset{\theta, x/x_0}{\Longrightarrow} \langle \Delta, \forall x{:}A;\, M;\, B\rangle\end{array}}{\mathtt{fillTerm}(\Delta_0 \vdash \boxed{?} : \Pi x_0{:}A_0.B_0) \overset{\theta}{\Longrightarrow} \langle \Delta;\, (\lambda x{:}A.M);\, (\Pi x{:}A.B)\rangle}\ \textit{ft-pi}$$

$|\Delta_0| \vdash \Pi x_0{:}A_0.B_0 : \mathit{type}$ given

$|\Delta_0| \vdash A_0 : \mathit{type}$ and $|\Delta_0|, x_0{:}A_0 \vdash B_0 : \mathit{type}$ by inversion on typing

$\Delta, \forall x{:}A \vdash \theta, x/x_0 : \Delta_0, \forall x_0{:}A_0$ and $\vdash \Delta, \forall x{:}A$ `raised`

and $(\theta, x/x_0)B_0 = B$ and $|\Delta, \forall \mathrm{x{:}A}| \vdash M : B$ by IH on $\mathcal{D}_0$

$\Delta \vdash \theta : \Delta_0$ and $\theta A_0 = A$ by Lemma 5.2.1

$\vdash \Delta$ `raised` by inversion on `raised`

$\theta(\Pi x_0{:}A_0.B_0) = \Pi x{:}A.B$ by rule

$|\Delta| \vdash A : \mathit{type}$ by Lemmas 5.2.7 and 5.2.2

$|\Delta| \vdash \lambda x{:}A.M : \Pi x{:}A.B$

$\square$

5.2.3 Groundness and Abstract Substitutions

We are interested in identifying a subset of well-behaved logic programs for the purpose of proving their termination. In order to prove the termination of a query, it we will need to be able to define a size metric on goal formulas, where the "size" of a logic variable is considered infinite, since it can be replaced with any term.

To this end, we say that a term or spine is *ground* iff it contains no existentially-quantified variables, excluding any that may occur in the type-labels of λ-abstractions (we do not consider type labels to influence the sizes of terms). This concept is formalized judgmentally below.

$$\frac{\Delta, \forall x{:}A \vdash M \,\underline{\dot{=}}}{\Delta \vdash \lambda x{:}A.M \,\underline{\dot{=}}} \qquad \frac{(h{:}A \in \Sigma \text{ or } \forall h{:}A \in \Delta) \quad \Delta \vdash \overline{M} \,\underline{\dot{=}}}{\Delta \vdash h\,\overline{M} \,\underline{\dot{=}}}$$

$$\frac{}{\Delta \vdash \cdot \,\underline{\dot{=}}} \qquad \frac{\Delta \vdash M \,\underline{\dot{=}} \quad \Delta \vdash \overline{M} \,\underline{\dot{=}}}{\Delta \vdash M :: \overline{M} \,\underline{\dot{=}}}$$

Logic programming can be thought of as giving a computational interpretation to relations; a program consists of searching for the inhabitants of a specified relation. The mode checker is, in essence, an abstract interpretation that tells us when the elements of the relation are treated consistently as the inputs to, or outputs from, a function. We can think of generalized substitutions as being the value returned by a successful query. As is typically the case with abstract interpretation, we need to define an abstract domain of values. To this end, we define abstract substitutions in the same manner as in [RP96] below.

Abstract Substitutions $\eta \;::=\; \cdot \mid \eta, \forall x{:}A \mid \eta, \exists^{?} x{:}A \mid \eta, \exists^{\underline{\dot{=}}} x{:}A$

In one sense are just mixed prefix contexts with a flag that determines which existential variables are assumed to be ground. In another sense, abstract substitutions can be thought of as sets of generalized substitutions that obey these flags. In the first sense, the judgment $\Delta \vdash \theta : \eta$, whose rules are listed below, is merely a refinement to the typing rules for generalized substitutions. In the second sense, $\Delta \vdash \theta : \eta$ can be thought of as meaning "$\theta \in \eta$."

$$\frac{}{\cdot \vdash \cdot : \cdot} \qquad \frac{\Delta \vdash \theta : \eta \quad \theta A = B \quad |\Delta| \vdash B : \mathit{type}}{\Delta, \forall y{:}B \vdash \theta, y/x : \eta, \forall x{:}A}$$

$$\frac{\Delta \vdash \theta : \eta \quad \theta A = B \quad |\Delta| \vdash M : B \quad \Delta \vdash M \stackrel{\perp}{=}}{\Delta \vdash \theta, M{:}B/x : \eta, \exists^{\stackrel{\perp}{=}} x{:}A}$$

$$\frac{\Delta \vdash \theta : \eta \quad \theta A = B \quad |\Delta| \vdash M : B \quad \exists y{:}B \in \Delta \quad \mathtt{expand}_y(B; \cdot) = M}{\Delta \vdash \theta, M{:}B/x : \eta, \exists^{?} x{:}A}$$

$$\frac{\Delta \vdash \theta : \eta \quad \theta A = B \quad |\Delta| \vdash M : B \quad \Delta \vdash M \stackrel{\perp}{=}}{\Delta \vdash \theta, M{:}B/x : \eta, \exists^{?} x{:}A} \qquad \frac{\Delta \vdash \theta : \eta \quad |\Delta| \vdash A : \mathit{type}}{\Delta, \exists x{:}A \vdash \theta : \eta}$$

We should note that the judgment $\Delta \vdash \theta : \eta$ requires that θ be well-behaved in that it can only map existential variables to terms which are either ground, or the eta-expansion of another existential variable; baking this invariant into this judgment simplifies the development of the metatheory of the mode-checker.

If $\Delta' \vdash \theta : \Delta$ then we say that η *approximates* θ if $\Delta' \vdash \theta : \eta$, whenever Δ' is either arbitrary, or clear from the context.

We can convert between mixed prefix contexts and abstract substitutions in the obvious ways: η_Δ assigns the "unknown" flag to each existential variable in Δ, and Δ_η strips the flags in η. Under the set-interpretation of η, Δ_η tells us the domain of the generalized substitutions in η, and η_Δ defines the set of all well-behaved generalized substitutions whose domain is Δ. Both interpretations are reflected in Lemma 5.2.16.

$$
\begin{array}{rcl}
\Delta_{\cdot} & = & \cdot \\
\Delta_{\eta,\forall x{:}A} & = & \Delta_{\eta}, \forall x{:}A \\
\Delta_{\eta,\exists^{?} x{:}A} & = & \Delta_{\eta}, \exists x{:}A \\
\Delta_{\eta,\exists^{\doteq} x{:}A} & = & \Delta_{\eta}, \exists x{:}A
\end{array}
$$

$$
\begin{array}{rcl}
\eta_{\cdot} & = & \cdot \\
\eta_{\Delta,\forall x{:}A} & = & (\eta_{\Delta}), \forall x{:}A \\
\eta_{\Delta,\exists x{:}A} & = & (\eta_{\Delta}), \exists^{?} x{:}A
\end{array}
$$

An abstract substitution can be converted to an ordinary LF context in much the same way that a mixed prefix context can.

$$
\begin{array}{rcl}
|\cdot| & = & \cdot \\
|\eta, \forall \mathrm{x{:}A}| & = & |\eta|, x{:}A \\
|\eta, \exists^{?} \mathrm{x{:}A}| & = & |\eta|, x{:}A \\
|\eta, \exists^{\doteq} \mathrm{x{:}A}| & = & |\eta|, x{:}A
\end{array}
$$

The notion of groundness for abstract substitutions is the same as for mixed prefix contexts, but with the following extra rule, which, under the set interpretation of η, says that for every θ in η, θM is ground. This is justified by Lemma 5.2.16.

$$
\frac{\exists^{\doteq} x{:}A \in \eta \quad \eta \vdash \overline{M} \doteq}{\eta \vdash x\,\overline{M} \doteq}
$$

We also find it useful to define the judgment $\eta' \vdash \theta : \eta$. If we interpret abstract substitutions as contexts, then the interpretation of the judgment is straightforward. If we interpret abstract substitutions as sets, then we can think of $\eta' \vdash \theta : \eta$ as saying that, for any θ' in η', $\theta' \circ \theta$ is in η.

$$\frac{}{\cdot \vdash \cdot : \cdot} \qquad \frac{\eta' \vdash \theta : \eta \quad \theta A = B \quad |\eta'| \vdash B : \mathit{type}}{\eta', \forall y{:}B \vdash \theta, y/x : \eta, \forall x{:}A}$$

$$\frac{\eta' \vdash \theta : \eta \quad \theta A = B \quad |\eta'| \vdash M : B \quad \eta' \vdash M \doteq}{\eta' \vdash \theta, M{:}B/x : \eta, \exists^{\doteq} x{:}A}$$

$$\frac{\eta' \vdash \theta : \eta \quad \theta A = B \quad |\eta'| \vdash M : B \quad \exists^{?} y{:}B \in \eta' \quad \mathtt{expand}_y(B; \cdot) = M}{\eta' \vdash \theta, M{:}B/x : \eta, \exists^{?} x{:}A}$$

$$\frac{\eta' \vdash \theta : \eta \quad \theta A = B \quad |\eta'| \vdash M : B \quad \eta' \vdash M \doteq}{\eta' \vdash \theta, M{:}B/x : \eta, \exists^{?} x{:}A} \qquad \frac{\eta' \vdash \theta : \eta \quad |\eta'| \vdash A : \mathit{type}}{\eta', \exists x{:}A \vdash \theta : \eta}$$

Lemma 5.2.15

1. *For every Δ, $|\Delta| = |\eta_\Delta|$*

2. *For every η, $|\eta| = |\Delta_\eta|$*

Proof: By straightforward induction on the structure of the given mixed-prefix context or abstract substitution. □

Lemma 5.2.16 (Abstract and Generalized Substitutions)

1. *If $\Delta' \vdash \theta : \Delta$ then $\Delta' \vdash \theta : \eta_\Delta$*

2. *If $\Delta' \vdash \theta : \eta$ then $\eta_{\Delta'} \vdash \theta : \eta$*

3. *If $\Delta' \vdash \theta : \eta$ then $\Delta \vdash \theta : \Delta_\eta$*

4. *If $\eta' \vdash \theta : \eta$ then $\Delta_{\eta'} \vdash \theta : \Delta_\eta$*

Proof: Each is by straightforward induction on the given derivation, using Lemma 5.2.15.
□

Lemma 5.2.17 (Abstract Inclusion) *If* $\Delta' \vdash \theta : \Delta$ *and* $\Delta' \vdash \theta : \eta$ *then* $\Delta_\eta = \Delta$

Proof: By a straightforward induction on either of the typing derivations. $\square$

Lemma 5.2.18 (Weakening for Groundness)

1. *If* Δ *can be obtained from* Δ' *by deleting some number of declarations then:*

   (a) *If* $\Delta \vdash M \doteq$ *then* $\Delta' \vdash M \doteq$

   (b) *If* $\Delta \vdash \overline{M} \doteq$ *then* $\Delta' \vdash \overline{M} \doteq$

2. *If* η *can be obtained from* η' *by deleting some number of declarations then:*

   (a) *If* $\eta \vdash M \doteq$ *then* $\eta' \vdash M \doteq$

   (b) *If* $\eta \vdash \overline{M} \doteq$ *then* $\eta' \vdash \overline{M} \doteq$

Proof: By straightforward simultaneous induction on the structure of the given groundness derivation. $\square$

Lemma 5.2.19 (Hereditary Substitutions Preserve Groundness)

1. *If* $\Delta \vdash M \doteq$ *then:*

   (a) *If* $\Delta \vdash \overline{M} \doteq$ *and* $\texttt{reduce}_\tau(M, \overline{M}) = N$ *then* $\Delta \vdash N \doteq$

   (b) *If* $\Delta \vdash M' \doteq$ *and* $[M/x]_\tau M' = N$ *then* $\Delta \vdash M' \doteq$

   (c) *If* $\Delta \vdash \overline{M} \doteq$ *and* $[M/x]_\tau \overline{M} = \overline{N}$ *then* $\Delta \vdash \overline{N} \doteq$

   (d) *If* $\Delta \vdash A \doteq$ *and* $[M/x]_\tau A = B$ *then* $\Delta \vdash B \doteq$

2. *If* $\eta \vdash M \doteq$ *then:*

   (a) *If* $\eta \vdash \overline{M} \doteq$ *and* $\texttt{reduce}_\tau(M, \overline{M}) = N$ *then* $\eta \vdash N \doteq$

   (b) *If* $\eta \vdash M' \doteq$ *and* $[M/x]_\tau M' = N$ *then* $\eta \vdash M' \doteq$

(c) *If* $\eta \vdash \overline{M} \mathrel{\underline{\underline{\perp}}}$ *and* $[M/x]_\tau \overline{M} = \overline{N}$ *then* $\eta \vdash \overline{N} \mathrel{\underline{\underline{\perp}}}$

(d) *If* $\eta \vdash A \mathrel{\underline{\underline{\perp}}}$ *and* $[M/x]_\tau A = B$ *then* $\eta \vdash B \mathrel{\underline{\underline{\perp}}}$

Proof: 1 is by straightforward simultaneous induction on the derivation of the given hereditary substitution, using Lemma 5.2.18; 2 analogous. □

Lemma 5.2.20 (Generalized Substitutions Preserve Groundness)

1. *If* $\Delta' \vdash \theta : \Delta$ *then:*

   (a) *If* $\Delta \vdash M \mathrel{\underline{\underline{\perp}}}$ *and* $\theta M = N$ *then* $\Delta' \vdash N \mathrel{\underline{\underline{\perp}}}$

   (b) *If* $\Delta \vdash \overline{M} \mathrel{\underline{\underline{\perp}}}$ *and* $\theta \overline{M} = N$ *then* $\Delta' \vdash \overline{N} \mathrel{\underline{\underline{\perp}}}$

2. *If* $\Delta \vdash \theta : \eta$ *then:*

   (a) $\eta \vdash M \mathrel{\underline{\underline{\perp}}}$ *and* $\theta M = N$ *then* $\Delta \vdash N \mathrel{\underline{\underline{\perp}}}$

   (b) $\eta \vdash \overline{M} \mathrel{\underline{\underline{\perp}}}$ *and* $\theta \overline{M} = \overline{N}$ *then* $\Delta \vdash \overline{N} \mathrel{\underline{\underline{\perp}}}$

3. *If* $\eta' \vdash \theta : \eta$ *then:*

   (a) $\eta \vdash M \mathrel{\underline{\underline{\perp}}}$ *and* $\theta M = N$ *then* $\eta' \vdash N \mathrel{\underline{\underline{\perp}}}$

   (b) $\eta \vdash \overline{M} \mathrel{\underline{\underline{\perp}}}$ *and* $\theta \overline{M} = \overline{N}$ *then* $\eta' \vdash \overline{N} \mathrel{\underline{\underline{\perp}}}$

Proof: 1 is by straightforward induction on the derivation of the given hereditary substitution, using Lemma 5.2.19. 2 and 3 are analogous.

□

Lemma 5.2.21 (Expansion and Groundness) *If* $\mathcal{D} : \texttt{expand}_h(A; \overline{M}) = M$ *then:*

1. *if* $\Delta \vdash \overline{M} \mathrel{\underline{\underline{\perp}}}$ *and* $h = c$ *or* ($h = x$ *and* $\forall x{:}B \in \Delta$) *then* $\Delta \vdash M \mathrel{\underline{\underline{\perp}}}$

2. *if* $\eta \vdash \overline{M} \mathrel{\underline{\underline{\perp}}}$ *and* $h = c$ *or* ($h = x$ *and either* $\forall x{:}B \in \Delta$ *or* $\exists x{:}B \in \Delta$) *then* $\eta \vdash M \mathrel{\underline{\underline{\perp}}}$

Proof: Each case is by straightforward induction on the structure of $\mathcal{D}$. $\square$

Lemma 5.2.22 (Groundness of Inputs and Outputs)

1. $\Delta \vdash \overline{M} \stackrel{\perp}{=}$ *iff* $\Delta \vdash \mathtt{inputs}_a(\overline{M}) \stackrel{\perp}{=}$ *and* $\Delta \vdash \mathtt{outputs}_a(\overline{M}) \stackrel{\perp}{=}$

2. $\Delta \vdash \overline{M} \stackrel{\perp}{=}$ *iff* $\Delta \vdash \mathtt{inputs}_a(\overline{M}) \stackrel{\perp}{=}$ *and* $\Delta \vdash \mathtt{outputs}_a(\overline{M}) \stackrel{\perp}{=}$

Proof: Direct, by definition of `inputs` and `outputs`. $\square$

Given a well-typed generalized substitution θ, it may be the case that θ is approximated by more than one η; for example, if $\Delta \vdash \theta : \eta, \exists^{\stackrel{\perp}{=}} x{:}A, \eta'$ then $\Delta \vdash \theta : \eta, \exists^{?} x{:}A, \eta'$.

Lemma 5.2.23 (Subtyping for Abstract Substitutions)

1. *If* $\Delta \vdash \theta : \eta_0, \exists^{\stackrel{\perp}{=}} x{:}A, \eta_1$ *then* $\Delta \vdash \theta : \eta_0, \exists^{?} x{:}A, \eta_1$

2. *If* $\eta \vdash \theta : \eta_0, \exists^{\stackrel{\perp}{=}} x{:}A, \eta_1$ *then* $\eta \vdash \theta : \eta_0, \exists^{?} x{:}A, \eta_1$

3. *If* $\eta_0, \exists^{?} x{:}A, \eta_1 \vdash \theta : \eta$ *then* $\eta_0, \exists^{\stackrel{\perp}{=}} x{:}A, \eta_1 \vdash \theta : \eta$

Given two abstract substitutions η and η' which approximate the same substitution θ, we can conclude that any existentially-quantified variable which is marked as ground in either η or η' must be ground. Thus, we define $\eta \sqcap \eta'$ as the meet of the groundness information in η and η'; under the set interpretation, we can think of $\sqcap$

as intersection.

$$\begin{array}{rcl}
\cdot \sqcap \cdot & = & \cdot \\
(\eta, \forall x{:}A) \sqcap (\eta', \forall x{:}A) & = & (\eta \sqcap \eta'), \forall x{:}A \\
(\eta, \exists^{?} x{:}A) \sqcap (\eta', \exists^{?} x{:}A) & = & (\eta \sqcap \eta'), \exists^{?} x{:}A \\
(\eta, \exists^{\doteq} x{:}A) \sqcap (\eta', \exists^{?} x{:}A) & = & (\eta \sqcap \eta'), \exists^{\doteq} x{:}A \\
(\eta, \exists^{?} x{:}A) \sqcap (\eta', \exists^{\doteq} x{:}A) & = & (\eta \sqcap \eta'), \exists^{\doteq} x{:}A \\
(\eta, \exists^{\doteq} x{:}A) \sqcap (\eta', \exists^{\doteq} x{:}A) & = & (\eta \sqcap \eta'), \exists^{\doteq} x{:}A
\end{array}$$

Lemma 5.2.24 (Properties of $\sqcap$)

1. *(a) If $\eta \vdash M \doteq$ then $\eta \sqcap \eta' \vdash M \doteq$*

   (b) If $\eta \vdash \overline{M} \doteq$ then $\eta \sqcap \eta' \vdash \overline{M} \doteq$

2. *If $\eta = \eta_0, \eta_1$ and $\eta' = \eta'_0, \eta'_1$ then $\eta \sqcap \eta' = \eta''$ iff $\eta'' = \eta''_0, \eta''_1$ and $\eta_0 \sqcap \eta'_0 = \eta''_0$ and $\eta_1 \sqcap \eta'_1 = \eta''_1$*

3. *$\eta = \eta \sqcap \eta$*

4. *$\eta \sqcap \eta' = \eta' \sqcap \eta$*

5. *If $\eta \sqcap \eta_\Delta = \eta'$ then $\eta' = \eta$*

Proof: 1 is by a straightforward simultaneous induction on the structure of $\doteq$; the others are by induction on the structure of η □

Lemma 5.2.25 (Soundness of $\sqcap$)

1. *If $\Delta \vdash \theta : \eta$ and $\Delta \vdash \theta : \eta'$ then $\Delta \vdash \theta : \eta \sqcap \eta'$*

2. *If* $\eta_0 \vdash \theta : \eta_1$ *and* $\eta_0 \vdash \theta : \eta_1'$ *then* $\eta_0 \vdash \theta : \eta_1 \sqcap \eta_1'$

3. *If* $\eta_0 \vdash \theta : \eta_1$ *and* $\eta_0' \vdash \theta : \eta_1$ *then* $\eta_0 \sqcap \eta_0' \vdash \theta : \eta_1$

4. *If* $\eta_0 \vdash \theta : \eta_1$ *and* $\eta_0' \vdash \theta : \eta_1'$ *then* $\eta_0 \sqcap \eta_0' \vdash \theta : \eta_1 \sqcap \eta_1'$

Proof: Each case is by straightforward induction on one of the given typing derivations. □

The notion of a pattern spine, inspired by a similar definition in [Roh96] will be important in our discussion of the properties of unification and in the definition of the mode/termination checker. We define pattern spines judgmentally below. Note that, unlike most of our judgments, $\Delta \vdash \overline{M}$ `pattern` and $\eta \vdash \overline{M}$ `pattern` can be inhabited when Δ and η are ill-formed, and moreover, $\overline{M}$ may contain free variable not declared in Δ or η.

$$\frac{}{\Delta \vdash \cdot\ \texttt{pattern}} \qquad \frac{\forall x{:}A \in \Delta \quad \texttt{expand}_x(A; \cdot) = M \quad x \# \overline{M} \quad \Delta \vdash \overline{M}\ \texttt{pattern}}{\Delta \vdash M :: \overline{M}\ \texttt{pattern}}$$

$$\frac{}{\eta \vdash \cdot\ \texttt{pattern}} \qquad \frac{\forall x{:}A \in \eta \quad \texttt{expand}_x(A; \cdot) = M \quad x \# \overline{M} \quad \eta \vdash \overline{M}\ \texttt{pattern}}{\eta \vdash M :: \overline{M}\ \texttt{pattern}}$$

Lemma 5.2.26 (Patterns are Ground)

1. *If* $\Delta \vdash \overline{M}$ `pattern` *then* $\Delta \vdash \overline{M} \mathrel{\underline{\perp}}$

2. *If* $\eta \vdash \overline{M}$ `pattern` *then* $\eta \vdash \overline{M} \mathrel{\underline{\perp}}$

Proof: By straightforward induction on the given derivations, using Lemma 5.2.21. □

Lemma 5.2.27 (Pattern Conversion)

1. *If* $\Delta \vdash \overline{M}$ `pattern` *then* $\eta_\Delta \vdash \overline{M}$ `pattern`

2. *If* $\eta \vdash \overline{M}$ `pattern` *then* $\Delta_\eta \vdash \overline{M}$ `pattern`

Proof: Each case is by a straightforward induction on the given derivation. $\square$

Lemma 5.2.28 (Weakening for Patterns)

1. *If* $\Delta \vdash \overline{M}$ `pattern` *then* $\Delta', \Delta \vdash \overline{M}$ `pattern`

2. *If* $\eta \vdash \overline{M}$ `pattern` *then* $\eta', \eta \vdash \overline{M}$ `pattern`

Proof: By induction on the structure of the given derivations. $\square$

Lemma 5.2.29 (Pattern Freshness)

1. *If* $x\#\Delta$ *and* $\Delta \vdash \overline{M}$ `pattern` *then* $x\#\overline{M}$

2. *If* $x\#\eta$ *and* $\eta \vdash \overline{M}$ `pattern` *then* $x\#\overline{M}$

3. *If* $x\#\overline{M}$ *and* $\Delta, \forall x{:}A, \Delta' \vdash \overline{M}$ `pattern` *then* $\Delta, \Delta' \vdash \overline{M}$ `pattern`

4. *If* $x\#\overline{M}$ *and* $\eta, \forall x{:}A, \eta' \vdash \overline{M}$ `pattern` *then* $\eta, \eta' \vdash \overline{M}$ `pattern`

Proof: Each case is by straightforward induction on the structure of $\overline{M}$, using Lemma 5.1.22 in 1 and 2 $\square$

Lemma 5.2.30 (Substitution Splitting)

1. *If* $\mathcal{D} : (\Delta' \vdash \theta : \Delta_0, \Delta_1)$ *then* $\Delta' = \Delta'_0, \Delta'_1$ *and* $\theta = \theta_0, \theta_1$ *and* $\Delta'_0 \vdash \theta_0 : \Delta_0$

2. *If* $\mathcal{D} : (\Delta \vdash \theta : \eta_0, \eta_1)$ *then* $\Delta = \Delta_0, \Delta_1$ *and* $\theta = \theta_0, \theta_1$ *and* $\Delta_0 \vdash \theta_0 : \eta_0$

3. *If* $\mathcal{D} : (\eta' \vdash \theta : \eta_0, \eta_1)$ *then* $\eta = \eta'_0, \eta'_1$ *and* $\theta = \theta_0, \theta_1$ *and* $\eta'_0 \vdash \theta_0 : \eta_0$

Proof: Each case is by a straightforward induction on the structure of $\mathcal{D}$. $\square$

Lemma 5.2.31 (Abstract Substitutions on Variables)

1. *If $\mathcal{D} : (\Delta \vdash \theta : \eta)$ then:*

   (a) *If $\forall x{:}A \in \eta$ then $\theta x = y$, and $\theta A = B$ and $\forall y{:}B \in \Delta$*

   (b) *If $\exists^{?}x{:}A \in \eta$ then $\theta x = M{:}B$ and $\theta A = B$ and $|\Delta| \vdash M : B$*

   (c) *If $\exists^{\doteq}x{:}A \in \eta$ then $\theta x = M{:}B$ and $\theta A = B$ and $|\Delta| \vdash M : B$ and $\Delta \vdash M \doteq$*

   (d) *If $\forall x{:}A \in \eta$ and $\forall y{:}B \in \eta$ and $x\#y$ then $\theta x\#\theta y$*

2. *If $\mathcal{D} : (\eta' \vdash \theta : \eta)$ then:*

   (a) *If $\forall x{:}A \in \eta$ then $\theta x = y$, and $\theta A = B$ and $\forall y{:}B \in \eta'$*

   (b) *If $\exists^{?}x{:}A \in \eta$ then $\theta x = M{:}B$ and $\theta A = B$ and $|\eta'| \vdash M : B$*

   (c) *If $\exists^{\doteq}x{:}A \in \eta$ then $\theta x = M{:}B$ and $\theta A = B$ and $|\eta'| \vdash M : B$ and $\eta' \vdash M \doteq$*

   (d) *If $\forall x{:}A \in \eta$ and $\forall y{:}B \in \eta$ and $x\#y$ then $\theta x\#\theta y$*

Proof: Analogous to Lemma 5.2.6. Each case is by induction on $\mathcal{D}$. All cases use Lemma 5.2.5, the (b) and (c) cases use Weakening for Typing and the (c) cases uses Weakening for Groundness. The (d) cases use the fact that mixed-prefix contexts and abstract substitutions cannot contain duplicate variable name declarations.

□

Lemma 5.2.32 (Properties of Split Substitutions)

1. *If $\mathcal{D} : (\Delta_0', \Delta_1' \vdash \theta_0, \theta_1 : \Delta_0, \Delta_1)$ and $\Delta_0' \vdash \theta_0 : \Delta_0$ then:*

   (a) *If $\Delta_0'' \vdash \theta_0 : \Delta_0$ then $\Delta_0'', \Delta_1' \vdash \theta_0, \theta_1 : \Delta_0, \Delta_1$*

   (b) *If $\forall x{:}A \in \Delta_1$ then $\theta x = y$ and $\theta A = B$ and $\forall y{:}B \in \Delta_1'$*

2. *If $\mathcal{D} : (\Delta_0', \Delta_1' \vdash \theta_0, \theta_1 : \eta_0, \eta_1)$ and $\Delta_0' \vdash \theta_0 : \eta_0$ then:*

   (a) *If $\Delta_0'' \vdash \theta_0 : \eta_0$ then $\Delta_0'', \Delta_1' \vdash \theta_0, \theta_1 : \eta_0, \eta_1$*

(b) *If* $\forall x{:}A \in \eta_1$ *then* $\theta x = y$ *and* $\theta A = B$ *and* $\forall y{:}B \in \Delta_1'$

3. *If* $\mathcal{D} : (\eta_0', \eta_1' \vdash \theta_0, \theta_1 : \eta_0, \eta_1)$ *and* $\eta_0' \vdash \theta_0 : \eta_0$ *then:*

   (a) *If* $\eta_0'' \vdash \theta_0 : \eta_0$ *then* $\eta_0'', \eta_1' \vdash \theta_0, \theta_1 : \eta_0, \eta_1$

   (b) *If* $\forall x{:}A \in \eta_1$ *then* $\theta x = y$ *and* $\theta A = B$ *and* $\forall y{:}B \in \eta_1'$

Proof: Each case is by straightforward induction on $\mathcal{D}$; the (b) cases use weakening for Typing and Lemma 5.2.2.

□

Lemma 5.2.33 (Substitutions Respect Patterns)

1. *If* $\Delta' \vdash \theta : \Delta$ *then:*

   (a) *If* $\forall x{:}A \in \Delta$ *and* $x \# \overline{M}$ *and* $\Delta \vdash \overline{M}\ \texttt{pattern}$ *then* $\theta x \# \theta \overline{M}$

   (b) *If* $\Delta = \Delta_0, \Delta_1$ *and* $\Delta' = \Delta_0', \Delta_1'$ *and* $\theta = \theta_0, \theta_1$ *and* $\Delta_0' \vdash \theta_0 : \Delta_0$ *and* $\Delta_1 \vdash \overline{M}\ \texttt{pattern}$ *and* $\theta \overline{M} = \overline{N}$ *then* $\Delta_1 \vdash \overline{N}\ \texttt{pattern}$

2. *If* $\Delta \vdash \theta : \eta$ *then:*

   (a) *If* $\forall x{:}A \in \eta$ *and* $x \# \overline{M}$ *and* $\eta \vdash \overline{M}\ \texttt{pattern}$ *then* $\theta x \# \theta \overline{M}$

   (b) *If* $\eta = \eta_0, \eta_1$ *and* $\Delta = \Delta_0, \Delta_1$ *and* $\theta = \theta_0, \theta_1$ *and* $\Delta_0 \vdash \theta_0 : \eta_0$ *and* $\eta_1 \vdash \overline{M}\ \texttt{pattern}$ *and* $\theta \overline{M} = \overline{N}$ *then* $\Delta_1 \vdash \overline{N}\ \texttt{pattern}$

3. *If* $\eta' \vdash \theta : \eta$ *then:*

   (a) *If* $\forall x{:}A \in \eta$ *and* $x \# \overline{M}$ *and* $\eta \vdash \overline{M}\ \texttt{pattern}$ *then* $\theta x \# \theta \overline{M}$

   (b) *If* $\eta = \eta_0, \eta_1$ *and* $\Delta' = \Delta_0', \Delta_1'$ *and* $\theta = \theta_0, \theta_1$ *and* $\eta_0' \vdash \theta_0 : \eta_0$ *and* $\eta_1 \vdash \overline{M}\ \texttt{pattern}$ *and* $\theta \overline{M} = \overline{N}$ *then* $\eta_1' \vdash \overline{N}\ \texttt{pattern}$

Proof: The (a) cases are all by induction on the given derivation of `pattern`, using Lemmas 5.1.22, Lemma 5.2.10, either Lemma 5.2.6 or Lemma 5.2.31, and Lemma 5.2.10. The (b) cases are also by induction on the given derivation of `pattern`, using Lemma 5.2.32, the appropriate part (a) and Lemma 5.2.10. $\square$

Lemma 5.2.34 (Strengthening for Groundness)

1. *If Δ can be obtained from Δ' by deleting some number of declarations then:*

   (a) *if $|\Delta| \vdash M : A$ and $\Delta' \vdash M \doteq$ then $\Delta \vdash M \doteq$*

   (b) *if $|\Delta|; A \vdash \overline{M} : B$ and $\Delta' \vdash \overline{M} \doteq$ then $\Delta \vdash \overline{M} \doteq$*

2. *If η can be obtained from η' by deleting some number of declarations then:*

   (a) *if $|\eta| \vdash M : A$ and $\eta' \vdash M \doteq$ then $\eta \vdash M \doteq$*

   (b) *if $|\eta|; A \vdash \overline{M} : B$ and $\eta' \vdash \overline{M} \doteq$ then $\eta \vdash \overline{M} \doteq$*

Proof: 1 is by a straightforward simultaneous induction over the structure of the given typing derivations. 2 is analogous.

$\square$

In some instances, we can reason backwards about the behavior of generalized substitutions with respect to groundness information. For example, if it is known that θM is a ground term, then it must be the case that, for every logic variable x that occurs rigidly in M, θx is ground. Thus, given Δ and M, we can use this observation define an abstract-substitution that characterizes the set of generalized substitutions that ground M. We write this judgment $\Delta \vdash M \doteq\!\!\Longrightarrow \eta$, and specify its rules (taken directly from [Roh96]) below.

$$\frac{\Delta, \forall x{:}A \vdash M \mathrel{\underline{\perp}}\!\Longrightarrow \eta, \forall x{:}A}{\Delta \vdash \lambda x{:}A.M \mathrel{\underline{\perp}}\!\Longrightarrow \eta} \qquad \frac{\forall x{:}\in\Delta \quad \Delta \vdash \overline{M} \mathrel{\underline{\perp}}\!\Longrightarrow \eta}{\Delta \vdash x\,\overline{M} \mathrel{\underline{\perp}}\!\Longrightarrow \eta}$$

$$\frac{c{:}A \in \Sigma \quad \Delta \vdash \overline{M} \mathrel{\underline{\perp}}\!\Longrightarrow \eta}{\Delta \vdash c\,\overline{M} \mathrel{\underline{\perp}}\!\Longrightarrow \eta}$$

$$\frac{\Delta' \vdash \overline{M}\ \texttt{pattern} \quad |\Delta, \exists x{:}\mathrm{A}, \Delta'|;\ A \vdash \overline{M} : a\,\overline{N}}{\Delta, \exists x{:}A, \Delta' \vdash x\,\overline{M} \mathrel{\underline{\perp}}\!\Longrightarrow \eta_\Delta, \exists^{\underline{\perp}} x{:}A, \eta_{\Delta'}}$$

$$\frac{}{\Delta, \exists x{:}A, \Delta' \vdash x\,\overline{M} \mathrel{\underline{\perp}}\!\Longrightarrow \eta_\Delta, \exists^{?} x{:}A, \eta_{\Delta'}}$$

$$\frac{}{\Delta \vdash \cdot \mathrel{\underline{\perp}}\!\Longrightarrow \eta_\Delta} \qquad \frac{\Delta \vdash M \mathrel{\underline{\perp}}\!\Longrightarrow \eta \quad \Delta \vdash \overline{M} \mathrel{\underline{\perp}}\!\Longrightarrow \eta'}{\Delta \vdash M :: \overline{M} \mathrel{\underline{\perp}}\!\Longrightarrow (\eta \sqcap \eta')}$$

We lifting this judgment to abstract substitutions below.

$$\frac{\eta, \forall x{:}A \vdash M \mathrel{\underline{\perp}}\!\Longrightarrow \eta', \forall x{:}A}{\eta \vdash \lambda x{:}A.M \mathrel{\underline{\perp}}\!\Longrightarrow \eta'} \qquad \frac{\forall x{:}A \in \eta \quad \eta \vdash \overline{M} \mathrel{\underline{\perp}}\!\Longrightarrow \eta'}{\eta \vdash x\,\overline{M} \mathrel{\underline{\perp}}\!\Longrightarrow \eta'}$$

$$\frac{\eta_1 \vdash \overline{M}\ \texttt{pattern} \quad |\eta_0, \exists^{?}\mathrm{x{:}A}, \eta_1|;\ A \vdash \overline{M} : a\,\overline{N}}{\eta_0, \exists^{?} x{:}A, \eta_1 \vdash x\,\overline{M} \mathrel{\underline{\perp}}\!\Longrightarrow \eta_0, \exists^{\underline{\perp}} x{:}A, \eta_1} \qquad \frac{}{\eta_0, \exists^{?} x{:}A, \eta_1 \vdash x\,\overline{M} \mathrel{\underline{\perp}}\!\Longrightarrow \eta_0, \exists^{?} x{:}A, \eta_1}$$

$$\frac{\exists^{\underline{\perp}} x{:}A \in \eta}{\eta \vdash x\,\overline{M} \mathrel{\underline{\perp}}\!\Longrightarrow \eta} \qquad \frac{}{\eta \vdash \cdot \mathrel{\underline{\perp}}\!\Longrightarrow \eta} \qquad \frac{\eta \vdash M \mathrel{\underline{\perp}}\!\Longrightarrow \eta' \quad \eta \vdash \overline{M} \mathrel{\underline{\perp}}\!\Longrightarrow \eta''}{\eta \vdash M :: \overline{M} \mathrel{\underline{\perp}}\!\Longrightarrow \eta' \sqcap \eta''}$$

Lemma 5.2.35 (Groundness Inversion for Patterns)

If Δ and Δ' are disjoint and $\mathcal{D} : (|\Delta| \vdash M : A)$ and $\mathcal{E} : (\texttt{reduce}_{A^-}(M, \overline{N}) = M')$ and $\mathcal{F} : (\Delta' \vdash \overline{N}\ \texttt{pattern})$ and $\Delta, \Delta' \vdash M' \mathrel{\underline{\perp}}$ then $\Delta \vdash M \mathrel{\underline{\perp}}$.

Proof: By induction on the structure of $\mathcal{E}$.

Case:

$$\mathcal{D} = \dfrac{\begin{array}{cc} \mathcal{D}_0 & \mathcal{D}_1 \\ |\Delta| \vdash A_1 : \mathit{type} & |\Delta, \forall \text{x:A}_1| \vdash M : A_2 \end{array}}{|\Delta| \vdash \lambda x{:}A_1.M : \Pi x{:}A_1.A_2}$$

$$\mathcal{E} = \dfrac{\begin{array}{cc} \mathcal{E}_0 & \mathcal{E}_1 \\ [N/x]_{A_1^-} M = N' & \texttt{reduce}_{A_2^-}(N', \overline{N}) = M' \end{array}}{\texttt{reduce}_{A_1^- \to A_2^-}(\lambda x{:}A_1.M, N :: \overline{N}) = M'}$$

$$\mathcal{F} = \dfrac{\begin{array}{cccc} & \mathcal{F}_0 & & \mathcal{F}_1 \\ \forall x{:}A \in \Delta', \forall x{:}A, \Delta'' & \texttt{expand}_x(A; \cdot) = N & x \# \overline{N} & \Delta', \forall x{:}A, \Delta'' \vdash \overline{N}\ \texttt{pattern} \end{array}}{\Delta', \forall x{:}A, \Delta'' \vdash N :: \overline{N}\ \texttt{pattern}}$$

Note that the x in $\lambda x{:}A_1.M$ and the x in $\Delta', \forall x{:}A, \Delta''$ can be assumed to be the same, by the renamability of bound variables and the fact that x doesn't occur in Δ.

Δ is disjoint from $\Delta', \forall x{:}A, \Delta''$	
and $\Delta, \Delta', \forall x{:}A, \Delta'' \vdash M' \doteq$	given
$\vert\Delta\vert, x{:}A_1 \vdash N : A_1$	by Theorem 5.1.25
$[N/x]_{A_1^-} M = M$	by Theorem 5.1.25
$N' = M$	by Lemma 5.1.3
$\Delta', \Delta'' \vdash \overline{N}\ \texttt{pattern}$	by Lemma 5.2.29
$\Delta, \forall x{:}A_1 \vdash M \doteq$	by IH on $\mathcal{E}_1$
$\Delta \vdash \lambda x{:}A_1.M \doteq$	by rule

Case:

$$\mathcal{E} \quad = \quad \dfrac{}{\texttt{reduce}_\tau(M, \cdot) = M}$$

Δ disjoint from Δ' and $|\Delta| \vdash M : A$

and $\Delta' \vdash \cdot$ `pattern` and $\Delta, \Delta' \vdash M \doteq$ given

$\Delta \vdash M \doteq$ by Lemma 5.2.34

$\square$

Lemma 5.2.36 (Reasoning Backwards About Groundness)

1. *If $\Delta' \vdash \theta : \Delta$ then:*

   (a) *if $\mathcal{D} : (\Delta \vdash M \doteq\Longrightarrow \eta)$ and $\theta M = N$ and $\Delta' \vdash N \doteq$ then $\Delta \vdash \theta : \eta$*

   (b) *If $\mathcal{D} : (\Delta \vdash \overline{M} \doteq\Longrightarrow \eta)$ and $\theta\overline{M} = \overline{N}$ and $\Delta' \vdash \overline{N} \doteq$ then $\Delta \vdash \theta : \eta$*

2. *If $\Delta \vdash \theta : \eta$ then:*

   (a) *if $\mathcal{D} : (\eta \vdash M \doteq\Longrightarrow \eta')$ and $\theta M = N$ and $\Delta \vdash N \doteq$ then $\Delta \vdash \theta : \eta'$*

   (b) *If $\mathcal{D} : (\eta \vdash \overline{M} \doteq\Longrightarrow \eta')$ and $\theta\overline{M} = \overline{N}$ and $\Delta \vdash \overline{N} \doteq$ then $\Delta \vdash \theta : \eta'$*

Proof: Each case is by simultaneous induction on the structure of $\mathcal{D}$, where the proofs for 1 and 2 are essentially the same. We show the interesting cases of 2 below.

1.

Case:

$$\mathcal{D} = \cfrac{\overset{\mathcal{D}_0}{\eta_1 \vdash \overline{M}\ \texttt{pattern}} \quad \overset{\mathcal{D}_1}{|\eta_0, \exists^? \text{x:A}, \eta_1|;\ A \vdash \overline{M} : a\,\overline{N}}}{\eta_0, \exists^? x{:}A, \eta_1 \vdash x\,\overline{M} \doteq\Longrightarrow \eta_0, \exists^{\doteq} x{:}A, \eta_1}$$

$\Delta \vdash \theta : \eta_0, \exists^? x{:}A, \eta_1$ and $\theta(x\,\overline{M}) = N$ and $\Delta \vdash N \doteq$ given

$\theta x = N_0{:}B$ and $\theta A = B$ by Lemma 5.2.31

$\theta\overline{M} = \overline{N_0}$ and $\texttt{reduce}_{B^-}(N_0, \overline{N_0}) = N$ by inversion on gsub app

$\Delta = \Delta_0, \Delta_1$ and $\theta = \theta_0, M{:}B/x, \theta_1$ and $\Delta_0 \vdash \theta_0' : \eta_0, \exists^? x{:}A$

by Lemma 5.2.30

$\theta'_0 = \theta_0, N_0{:}B/x$ and $|\Delta_0| \vdash M : B$ and $\Delta_0 \vdash \theta_0 : \eta_0$

by inversion on abs sub typing and Lemma 5.2.2

$\Delta_1 \vdash \overline{N_0}$ `pattern` by Lemma 5.2.33

$\Delta_0 \vdash N_0 \doteqdot$ by Lemma 5.2.35

$\Delta_0 \vdash \theta_0 : \eta_0, \exists^{\doteqdot}x{:}A$ by rule

$\Delta \vdash \theta : \eta_0, \exists^{\doteqdot}x{:}A, \eta_1$ by Lemma 5.2.32

2.

Case:

$$\mathcal{D} = \dfrac{\overset{\mathcal{D}_0}{\eta \vdash M \doteqdot\!\Longrightarrow \eta'} \qquad \overset{\mathcal{D}_1}{\eta \vdash \overline{M} \doteqdot\!\Longrightarrow \eta''}}{\eta \vdash M :: \overline{M} \doteqdot\!\Longrightarrow \eta' \sqcap \eta''}$$

$\Delta \vdash \theta : \eta$ and $\theta(M :: \overline{M}) = N :: \overline{N'}$ and $\Delta \vdash N :: \overline{N} \doteqdot$ given

$\theta M = N$ and $\theta\overline{M} = \overline{N}$ by inversion on gsub app

$\Delta \vdash N \doteqdot$ and $\Delta \vdash \overline{N} \doteqdot$ by inversion on groundness

$\Delta \vdash \theta : \eta'$ by IH on $\mathcal{D}_0$

$\Delta \vdash \theta : \eta''$ by IH on $\mathcal{D}_1$

$\Delta \vdash \theta : \eta' \sqcap \eta''$ by Lemma 5.2.25

□

Lemma 5.2.37 (Ground Terms Yield Trivial Information)

1. (a) *If $\Delta \vdash M \doteqdot$ and $\Delta \vdash M \doteqdot\!\Longrightarrow \eta$ then $\eta = \eta_\Delta$*

   (b) *If $\Delta \vdash \overline{M} \doteqdot$ and $\Delta \vdash \overline{M} \doteqdot\!\Longrightarrow \eta$ then $\eta = \eta_\Delta$*

2. (a) *If $\eta \vdash M \doteqdot$ and $\eta \vdash M \doteqdot\!\Longrightarrow \eta'$ then $\eta' = \eta$*

   (b) *If $\eta \vdash \overline{M} \doteqdot$ and $\eta \vdash \overline{M} \doteqdot\!\Longrightarrow \eta'$ then $\eta' = \eta$*

Proof: 1 is by straightforward simultaneous induction on the given derivation of groundness, using Lemma 5.2.24; 2 is analogous. □

The following lemma plays a crucial role in the soundness of our termination/mode checker.

Lemma 5.2.38 (Abstract Unification)

1. *If* $\Delta' \vdash \theta : \Delta$ *then:*
   (a) *if* $\mathcal{D} : (\Delta \vdash M \mathrel{\underline{\perp}\!\!\Longrightarrow} \eta)$ *and* $\theta M = M'$ *then* $\Delta' \vdash M' \mathrel{\underline{\perp}\!\!\Longrightarrow} \eta'$ *and* $\eta \vdash \theta : \eta'$

   (b) *if* $\mathcal{D} : (\Delta \vdash \overline{M} \mathrel{\underline{\perp}\!\!\Longrightarrow} \eta)$ *and* $\theta\overline{M} = \overline{M'}$ *then* $\Delta' \vdash \overline{M'} \mathrel{\underline{\perp}\!\!\Longrightarrow} \eta'$ *and* $\eta \vdash \theta : \eta'$

2. *If* $\Delta \vdash \theta : \eta$ *then:*
   (a) *if* $\mathcal{D} : (\eta \vdash M \mathrel{\underline{\perp}\!\!\Longrightarrow} \eta')$ *and* $\theta M = M'$ *then* $\Delta \vdash M' \mathrel{\underline{\perp}\!\!\Longrightarrow} \eta''$ *and* $\eta'' \vdash \theta : \eta'$

   (b) *if* $\mathcal{D} : \eta \vdash \overline{M} \mathrel{\underline{\perp}\!\!\Longrightarrow} \eta'$ *and* $\theta\overline{M} = \overline{M'}$ *then* $\Delta \vdash \overline{M'} \mathrel{\underline{\perp}\!\!\Longrightarrow} \eta''$ *and* $\eta'' \vdash \theta : \eta'$

3. *If* $\eta_0' \vdash \theta : \eta_0$ *then:*
   (a) *If* $\mathcal{D} : (\eta_0 \vdash M \mathrel{\underline{\perp}\!\!\Longrightarrow} \eta_1)$ *and* $\theta M = M'$ *then* $\eta_0' \vdash M' \mathrel{\underline{\perp}\!\!\Longrightarrow} \eta_1'$ *and* $\eta_1' \vdash \theta : \eta_1$

   (b) *If* $\mathcal{D} : (\eta_0 \vdash \overline{M} \mathrel{\underline{\perp}\!\!\Longrightarrow} \eta_1)$ *and* $\theta\overline{M} = \overline{M'}$ *then* $\eta_0' \vdash \overline{M'} \mathrel{\underline{\perp}\!\!\Longrightarrow} \eta_1'$ *and* $\eta_1' \vdash \theta : \eta_1$

Proof: Each numbered case is by simultaneous induction on $\mathcal{D}$, where the reasoning is essentially the same each time. We show the interesting cases of 2 below.

1(a)

Case:

$$\mathcal{D} = \dfrac{\begin{matrix}\mathcal{D}_0 \\ \eta_1 \vdash \overline{M}\ \texttt{pattern}\end{matrix} \qquad \begin{matrix}\mathcal{D}_1 \\ |\eta_0, \exists^? \text{x:A}, \eta_1|;\, A \vdash \overline{M} : a\,\overline{N}\end{matrix}}{\eta_0, \exists^? x{:}A, \eta_1 \vdash x\,\overline{M} \mathrel{\overset{+}{=}}\!\!\Longrightarrow \eta_0, \exists^{\overset{+}{=}} x{:}A, \eta_1}$$

$\Delta \vdash \theta : \eta_0, \exists^? x{:}A, \eta_1$ and $\theta(x\,\overline{M}) = M'$ given

$\theta x = N{:}B$ and $\theta\overline{M} = \overline{M'}$ and $\texttt{reduce}_{B^-}(N, \overline{M'}) = M'$

by inversion on gsub app

$\Delta = \Delta_0, \Delta_1$ and $\theta = \theta_0', \theta_1$ and $\Delta_0 \vdash \theta_0' : \eta_0, \exists^? x{:}A$ by Lemma 5.2.30

$\theta_0' = \theta_0, N{:}B/x$ and $\Delta_0 \vdash \theta_0 : \eta_0$ and $|\Delta_0| \vdash N : B$ and $\theta A = B$

and either $\Delta_0 \vdash N \overset{+}{=}$ or ($\exists y{:}B \in \Delta_0$ and $\texttt{expand}_y(B; \cdot) = N$)

by inversion on abs sub typing

Assume $\Delta_0 \vdash N \overset{+}{=}$ Case 1

let $\eta'' = \eta_\Delta$

$\Delta_1 \vdash \overline{M'}\ \texttt{pattern}$ by Lemma 5.2.33

$\Delta \vdash \overline{M'}\ \texttt{pattern}$ by Lemma 5.2.28

$\Delta \vdash \overline{M'} \overset{+}{=}$ by Lemma 5.2.26

$\Delta \vdash M' \overset{+}{=}$ by Lemma 5.2.20

$\Delta \vdash M' \mathrel{\overset{+}{=}}\!\!\Longrightarrow \eta''$ by Lemma 5.2.37

$\Delta_0 \vdash (\theta_0, N{:}B/x) : (\eta_0, \exists^{\overset{+}{=}} x{:}A)$ by rule

$\Delta \vdash \theta : \eta_0, \exists^{\overset{+}{=}} x{:}A, \eta_1$ by Lemma 5.2.32

$\eta'' \vdash \theta : \eta_0, \exists^{\overset{+}{=}} x{:}A, \eta_1$ by Lemma 5.2.16

Assume $\exists y{:}B \in \Delta_0$ and $\texttt{expand}_y(B; \cdot) = N$ Case 2

$N = y\,\overline{M'}$ by Theorem 5.1.25 and Lemma 5.1.3

$\Delta_0 = \Delta_0', \exists y{:}B, \Delta_0''$ by inversion on $\exists y{:}B \in \Delta_0$

Let $\eta'' = \eta_{\Delta_0'}, \exists^{\pm} y{:}B, \eta_{\Delta_0''}, \eta_{\Delta_1}$

$\Delta_1 \vdash \overline{M'}$ `pattern` by Lemma 5.2.33

$\Delta_0'', \Delta_1 \vdash \overline{M'}$ `pattern` by Lemma 5.2.28

$\eta_{\Delta_0''}, \eta_{\Delta_1} \vdash \overline{M'}$ `pattern` by Lemma 5.2.27

$\Delta \vdash \theta : \Delta_{\eta_0, \exists^? x:A, \eta_1}$ By Lemma 5.2.16

$\theta(a\,\overline{N}) = a\,\overline{N'}$ and $|\Delta|;\, B \vdash \overline{M'} : a\,\overline{N'}$

by Lemma 5.2.7 and Lemma 5.2.2

$\Delta \vdash M' \stackrel{\pm}{\Longrightarrow} \eta''$ by rule

$\eta_{\Delta_0} \vdash \theta_0 : \eta_0$ by Lemma 5.2.16

$\eta_{\Delta_0'}, \exists^{\pm} y{:}B, \eta_{\Delta_0''} \vdash \theta_0 : \eta_0$ by Lemma 5.2.23

$\eta_{\Delta_0'}, \exists^{\pm} y{:}B, \eta_{\Delta_0''} \vdash N \,\pm$ by Lemma 5.2.21

$\eta_{\Delta_0'}, \exists^{\pm} y{:}B, \eta_{\Delta_0''} \vdash (\theta_0, N{:}B) : \eta_0, \exists^{\pm} x{:}A$ by rule

$\eta'' \vdash \theta'' : \eta_0, \exists^{\pm} x{:}A, \eta_1$ by Lemma 5.2.32

1(b)

Case:

$$\mathcal{D} = \frac{\begin{array}{cc} \mathcal{D}_0 & \mathcal{D}_1 \\ \eta \vdash M \stackrel{\pm}{\Longrightarrow} \eta_0' & \eta \vdash \overline{M} \stackrel{\pm}{\Longrightarrow} \eta_1' \end{array}}{\eta \vdash M :: \overline{M} \stackrel{\pm}{\Longrightarrow} \eta_0' \sqcap \eta_1'}$$

$\Delta \vdash \theta : \eta$ and $\theta(M :: \overline{M}) = M' :: \overline{M'}$ given

$\theta M = M'$ and $\theta\overline{M} = \theta\overline{M'}$ by inversion on gsub app

$\Delta \vdash M' \stackrel{\pm}{\Longrightarrow} \eta_0''$ and $\eta_0'' \vdash \theta : \eta_0'$ by IH 1 on $\mathcal{D}_0$

$\Delta \vdash \overline{M'} \stackrel{\pm}{\Longrightarrow} \eta_1''$ and $\eta_1'' \vdash \theta : \eta_1'$ by IH 2 on $\mathcal{D}_1$

$\Delta \vdash M' :: \overline{M'} \stackrel{\pm}{\Longrightarrow} \eta_0'' \sqcap \eta_1''$ by rule

$\eta_0'' \sqcap \eta_1'' \vdash \theta : \eta_0' \sqcap \eta_1'$ by Lemma 5.2.24

□

Lemma 5.2.39 (Weakening for Ground Implication)

1. *If* $|\Delta| \vdash A : type$ *then:*

   (a) *if* $\mathcal{D} : (\Delta, \Delta' \vdash M \overset{\perp}{=}\!\!\Longrightarrow \eta)$ *then* $\eta = \eta_0, \eta_1$ *and* $\Delta_0, \forall x{:}A, \Delta_1 \vdash M \overset{\perp}{=}\!\!\Longrightarrow \eta_0, \forall x{:}A, \eta_1$ *and* $\Delta_0, \exists x{:}A, \Delta_1 \vdash M \overset{\perp}{=}\!\!\Longrightarrow \eta_0, \exists^{?} x{:}A, \eta_1$

   (b) *if* $\mathcal{D} : (\Delta, \Delta' \vdash \overline{M} \overset{\perp}{=}\!\!\Longrightarrow \eta)$ *then* $\eta = \eta_0, \eta_1$ *and* $\Delta_0, \forall x{:}A, \Delta_1 \vdash \overline{M} \overset{\perp}{=}\!\!\Longrightarrow \eta_0, \forall x{:}A, \eta_1$ *and* $\Delta_0, \exists x{:}A, \Delta_1 \vdash \overline{M} \overset{\perp}{=}\!\!\Longrightarrow \eta_0, \exists^{?} x{:}A, \eta_1$

2. *If* $|\eta_0| \vdash A : type$ *then:*

   (a) *if* $\mathcal{D} : (\eta_0, \eta_1 \vdash M \overset{\perp}{=}\!\!\Longrightarrow \eta')$ *then* $\eta = \eta_0', \eta_1'$ *and* $\eta_0, \forall x{:}A, \eta_1 \vdash M \overset{\perp}{=}\!\!\Longrightarrow \eta_0', \forall x{:}A, \eta_1'$ *and* $\eta_0, \exists^{?} x{:}A, \eta_1 \vdash M \overset{\perp}{=}\!\!\Longrightarrow \eta_0', \exists^{?} x{:}A, \eta_1'$ *and* $\eta_0, \exists^{\overset{\perp}{=}} x{:}A, \eta_1 \vdash M \overset{\perp}{=}\!\!\Longrightarrow \eta_0', \exists^{\overset{\perp}{=}} x{:}A, \eta_1'$

   (b) *if* $\mathcal{D} : (\eta_0, \eta_1 \vdash \overline{M} \overset{\perp}{=}\!\!\Longrightarrow \eta')$ *then* $\eta = \eta_0', \eta_1'$ *and* $\eta_0, \forall x{:}A, \eta_1 \vdash \overline{M} \overset{\perp}{=}\!\!\Longrightarrow \eta_0', \forall x{:}A, \eta_1'$ *and* $\eta_0, \exists^{?} x{:}A, \eta_1 \vdash \overline{M} \overset{\perp}{=}\!\!\Longrightarrow \eta_0', \exists^{?} x{:}A, \eta_1'$ *and* $\eta_0, \exists^{\overset{\perp}{=}} x{:}A, \eta_1 \vdash \overline{M} \overset{\perp}{=}\!\!\Longrightarrow \eta_0', \exists^{\overset{\perp}{=}} x{:}A, \eta_1'$

Proof: Each case is by straightforward induction on the structure of $\mathcal{D}$; the (b) cases use the (a) cases, and the (a) cases use Lemma 5.2.24. □

5.2.4 Size Ordering

In general, finding a solution to a query using `fillTerm` will, via the *fs-arr* case of `fillSpine`, requires the solution of some number of subgoals. In order to know that this process terminates, we need to know that every subgoal is, in some sense, smaller than the goal that spawned it. For example, any goal of the form $add\, z\, M\, N$ should be considered smaller than any subgoal of the form $add\,(s\, z)\, M'\, N'$, given that z is considered smaller than $s\, z$. Thus, given some well-founded ordering on LF-terms, it should be possible to prove the termination of proof-search based on this ordering. However, we do need to be careful. Consider proof search on the following signature, where the sole argument to p is moded positive.

$$\begin{aligned} &\ldots, p{:}nat \to type, \\ &f : \Pi x{:}nat.(p\,x) \to (p\,(s\,x)) \end{aligned}$$

It would be tempting to conclude that any well-typed query of the form $\Delta \vdash \boxed{?} : p\,M$ will terminate (unsuccessfully, as we have omitted any base case for p), since it seems as though x should always be smaller than $s\,x$. However, this sort of reasoning is invalid when querying for non-ground terms. For example, the (ill-moded) query $\cdot, \exists y{:}nat \vdash \boxed{?} : p\,y$ will not terminate in our semantics (nor would an equivalent example in Prolog), because each subgoal generated by `fillSpine` will, after unification, be essentially the same as the original goal before unification. Our mode/termination checker will rule out such bad queries by guaranteeing that input arguments are always ground before `fillTerm` is called, meaning that our notion of well-founded ordering on LF terms need only be defined on ground terms.

To this end, we assume the existence of a *semantic domain* $\mathbb{S}$ that comes equipped with a notion of well-founded ordering (*the semantic ordering*), and a *semantic mapping*, written $[\![M]\!]\ \Delta$, that maps terms M that are both well-typed and ground in Δ to elements of $\mathbb{S}$. For example, $\mathbb{S}$ could be the term-algebra corresponding to the natural numbers, where $[\![M]\!]\ \Delta$ counts the number of constants and variables in M. Although we use the word "semantic" here, it should be noted that $\mathbb{S}$, $[\![M]\!]\ \Delta$, semantic equality and the semantic ordering should be defined using the same syntactic machinery as elsewhere in this thesis, although we treat the well-foundedness of the semantic ordering as an axiomatic assumption. We list some of properties that we expect of a semantic domain and the semantic mapping below.

Definition 5.2.40 (Semantic Domain) *A syntactic category* $\mathbb{S}$ *is a* semantic domain *iff there exist the binary relation* $<$ *on the elements of* $\mathbb{S}$*, and the partial-function* $[\![-]\!]\ (-)$ *from terms and mixed-prefix contexts to elements of* $\mathbb{S}$ *that satisfy the following properties.*

1. *(Semantic Ordering)* $<_s$ *is a transitive, well-founded ordering.*
2. *(Semantic Mapping) If* $|\Delta| \vdash M : A$ *and* $\Delta \vdash M \doteq$ *then there exists some* S *in* $\mathbb{S}$ *such that* $[\![M]\!]\,\Delta = S$*.*
3. *(Substitution) If* $\Delta' \vdash \theta : \Delta$ *and* $\Delta \vdash M \doteq$ *and* $\theta M = M'$ *and* $[\![M]\!]\,\Delta = S$ *and* $[\![M']\!]\,\Delta' = S'$ *then* $S' = S$*.*
4. *(Weakening) If* $[\![M]\!]\,\Delta = S$ *and* Δ *can be obtained from* Δ' *by deleting some number of declarations, then* $[\![\Delta']\!]\,M = S$

Example 5.2.41 (Semantic Domain for LPO) *The proof terms from Chapter 4 can be represented in LF via a straightforward adequate encoding (for details, see* `http://www.twelf.org/lpo/`*). Let* $\mathbb{S}$ *be the syntactic category of finite labeled trees over the signature defined in Section 4.2.2, as ordered by* $<_{\mathsf{lpo}}$*, and let* $[\![M]\!]\ \Delta$ *be defined in the same manner as* $[\![C^F]\!]$ *where* M *is the LF-encoding of* C^F*.* $\mathbb{S}$ *is a semantic domain for this LF encoding.*

We leave a description of how this semantic domain can be generalized to arbitrary LF signatures to future work.

In the following example, we see how natural numbers can be used to justify the subterm ordering we have used throughout this dissertation.

Example 5.2.42 (Semantic Domain for Subterm Ordering) *The syntactic category of natural numbers under the usual ordering* $<$ *can be used as a semantic domain for any LF signature. The stripping function* $[\![M]\!]\,\Delta$ *and its generalization* *to spines* $[\![M]\!]\ A;\ \Delta$ *are defined judgmentally below. Note that the relation* $\equiv$ *holds whenever two type-families are mutually recursive; its use in the spine stripping function is motivated by considerations discussed in Section 2.1.*

$$\frac{[\![M]\!]\,\Delta, \forall x{:}A = n}{[\![\lambda x{:}A.M]\!]\,\Delta = n} \qquad \frac{\forall x{:}A \in \Delta \quad [\![\overline{M}]\!]\,A;\ \Delta = n}{[\![x\,\overline{M}]\!]\,\Delta = s\ n} \qquad \frac{c{:}A \in \Sigma \quad [\![\overline{M}]\!]\,A;\ \Delta = n}{[\![c\,\overline{M}]\!]\,\Delta = s\ n}$$

$$\frac{}{[\![\cdot]\!]\,a\,\overline{N};\ \Delta = z} \qquad \frac{\mathtt{hd}(A) \equiv \mathtt{hd}(B) \quad [\![M]\!]\,\Delta = n_0 \quad [\![\overline{M}]\!]\,B;\ \Delta = n_1 \quad add(n_0;\ n_1;\ n)}{[\![M :: \overline{M}]\!]\,\Pi x{:}A.B;\ \Delta = n}$$

$$\frac{\mathtt{hd}(A) \not\equiv \mathtt{hd}(B) \quad [\![\overline{M}]\!]\,B;\ \Delta = n}{[\![M :: \overline{M}]\!]\,\Pi x{:}A.B;\ \Delta = n}$$

The proof that this forms a valid semantic domain is straightforward.

Each semantic domain gives rise to an induction principle over LF terms. For the remainder of this section, we assume that the choice of semantic domain $\mathbb{S}$ is fixed, although we choose to leave it abstract for now. Later we will consider the impact that the strength of the semantic ordering has on our termination argument.

The semantic domain gives us a well-founded ordering on LF terms, but, in general, it is build up more complex well-founded orderings from simpler ones; for example lexicographic orderings have already been used several times in this document. Thus, if the Ackermann function is encoded in LF as $ack{:}nat \rightarrow nat \rightarrow nat \rightarrow type$, then it should be possible to specify that any query of the form $\Delta \vdash \boxed{?} : ack\, M_1\, M_2\, M_3$ is smaller than any query of the form $\Delta' \vdash \boxed{?} : ack\, N_1\, N_2\, N_3$ whenever either $[\![\Delta]\!]\,M_1 <_s [\![\Delta']\!]\,N_1$ or $[\![\Delta]\!]\,M_1 = [\![\Delta']\!]\,N_1$ and $[\![\Delta]\!]\,M_2 <_s [\![\Delta']\!]\,N_2$. To this end, we define *termination measures* below.

Termination Measures $O \ ::= \ M \mid lex\langle O_1; O_2\rangle \mid simul\langle O_1; O_2\rangle$

The *lex* and *simul* constructors can be thought of as providing lexicographic and simultaneous termination orderings. Although we could add a termination order for the empty measure and the commutative version of the simultaneous ordering, we omit their specifications for the sake of simplicity.

We lift the notions of typing, groundness and generalized substitutions to termination measures in the straightforward way. We list the names and descriptions of the relevant judgments in the table below, but omit their inference rules for the sake of brevity.

$\Gamma \vdash O$ `wellTyped`	every M in O satisfies $\Gamma \vdash M : A$
$\Delta \vdash O$ ⏚	every M in O satisfies $\Delta \vdash M$ ⏚
$\eta \vdash O$ ⏚	every M in O satisfies $\eta \vdash M$ ⏚
$\theta\, O = O'$	applying θ to every M in O results in O'

Definition 5.2.43 (Measure Functions) *For each family constant $a{:}K$ in Σ, the user must specify a deterministic function* $\mathtt{measure}_a\ \overline{M} = O$ *which is total whenever* $\cdot;\, K \vdash \overline{M} : type$*. Moreover, the function must satisfy the following properties.*

1. *if $M \in O$, then $M \in \mathtt{inputs}_a(\overline{M})$*
2. $\mathtt{measure}_a$ *is parametric in* $\overline{M}$

For example, $\mathtt{measure}_{ack}\ (M_0 :: M_1 :: M_2 :: \cdot) = lex\langle M_0; M_1\rangle$*, for every M_0, M_1 and M_2.*

We compare the size of termination measures O_1 and O_2, using the judgments $\langle\Delta_1; O_1\rangle <_o \langle\Delta_2; O_2\rangle$, $\langle\Delta_1; O_1\rangle \leq_o \langle\Delta_2; O_2\rangle$ and $\langle\Delta_1; O_1\rangle =_o \langle\Delta_2; O_2\rangle$, where O_i is assumed to be well-typed in Δ_i; the rules for $\langle\Delta_1; M\rangle <_o \langle\Delta_2; N\rangle$ and $\langle M; \Delta_1\rangle =_o \langle N; \Delta_2\rangle$ are defined in terms of the semantic mapping.

$$\frac{[\![M]\!]\,\Delta_1 <_s [\![M]\!]\,\Delta_2}{\langle M;\Delta_1\rangle <_o \langle N;\Delta_2\rangle} \qquad \frac{[\![M]\!]\,\Delta_1 = [\![M]\!]\,\Delta_2}{\langle M;\Delta_1\rangle =_o \langle N;\Delta_2\rangle}$$

$$\frac{\langle O_1;\Delta\rangle <_o \langle O_1';\Delta'\rangle}{\langle lex\langle O_1;O_2\rangle;\Delta\rangle <_o \langle lex\langle O_1';O_2'\rangle;\Delta'\rangle} \qquad \frac{\langle O_1;\Delta\rangle =_o \langle O_1';\Delta'\rangle \quad \langle O_2;\Delta\rangle <_o \langle O_2';\Delta'\rangle}{\langle lex\langle O_1;O_2\rangle;\Delta\rangle <_o \langle lex\langle O_1';O_2'\rangle;\Delta'\rangle}$$

$$\frac{\langle O_1;\Delta\rangle =_o \langle O_1';\Delta'\rangle \quad \langle O_2;\Delta\rangle =_o \langle O_2';\Delta'\rangle}{\langle lex\langle O_1;O_2\rangle;\Delta\rangle =_o \langle lex\langle O_1';O_2'\rangle;\Delta'\rangle} \qquad \frac{\langle O_1;\Delta\rangle =_o \langle O_1';\Delta'\rangle \quad \langle O_2;\Delta\rangle =_o \langle O_2';\Delta'\rangle}{\langle lex\langle O_1;O_2\rangle;\Delta\rangle \leq_o \langle lex\langle O_1';O_2'\rangle;\Delta'\rangle}$$

$$\frac{\langle O_1;\Delta\rangle <_o \langle O_1';\Delta'\rangle \quad \langle O_2;\Delta\rangle \leq_o \langle O_2';\Delta'\rangle}{\langle simul\langle O_1;O_2\rangle;\Delta\rangle <_o \langle simul\langle O_1';O_2'\rangle;\Delta'\rangle} \qquad \frac{\langle O_1;\Delta\rangle \leq_o \langle O_1';\Delta'\rangle \quad \langle O_2;\Delta\rangle <_o \langle O_2';\Delta'\rangle}{\langle simul\langle O_1;O_2\rangle;\Delta\rangle <_o \langle simul\langle O_1';O_2'\rangle;\Delta'\rangle}$$

$$\frac{\langle O;\Delta\rangle <_o \langle O';\Delta'\rangle}{\langle O;\Delta\rangle \leq_o \langle O';\Delta'\rangle} \qquad \frac{\langle O;\Delta\rangle =_o \langle O';\Delta'\rangle}{\langle O;\Delta\rangle \leq_o \langle O';\Delta'\rangle}$$

The following lemma lifts some of the useful properties of LF terms to measures.

Lemma 5.2.44 (Properties of Termination Measures)

1. *If* $\Gamma \vdash a\,\overline{M} : type$ *then* $\Gamma \vdash (\texttt{measure}_a\ \overline{M})\ \texttt{wellTyped}$
2. *If* $\Delta \vdash \texttt{inputs}_a(\overline{M}) \doteqdot$ *then* $\Delta \vdash \texttt{measure}_a\ \overline{M} \doteqdot$
3. *If* $\eta \vdash \texttt{inputs}_a(\overline{M}) \doteqdot$ *then* $\eta \vdash \texttt{measure}_a\ \overline{M} \doteqdot$
4. *If* $\theta(a\,\overline{M}) = a\,\overline{N}$ *and* $\texttt{measure}_a\ \overline{M} = O$ *then* $\theta O = O'$ *and* $\texttt{measure}_a\ \overline{N} = O'$

Proof: Direct, by the definition of $\texttt{measure}_a$. $\square$

The following lemma lifts some of the useful properties of semantic domains to termination measures.

Lemma 5.2.45 (Orderings on Measures)

1. $<_o$ *is a transitive, well-founded ordering*

2. *$=_o$ is a transitive, symmetric and if $|\Delta| \vdash O$ `wellTyped` and $\Delta \vdash O \doteq$ then $\langle \Delta; O\rangle =_o \langle \Delta; O\rangle$*

3. *If $\langle \Delta; O\rangle =_o \langle \Delta; O\rangle$ and Δ can be obtained from Δ' by deleting some number of declarations then $\langle \Delta; O\rangle =_o \langle \Delta'; O\rangle$*

4. *If $\langle O_1; \Delta_1\rangle =_o \langle O_1'; \Delta_1'\rangle$ and $\langle O_2; \Delta_2\rangle =_o \langle O_2'; \Delta_2'\rangle$ and $\langle O_1; \Delta_1\rangle <_o \langle O_2; \Delta_2\rangle$ then $\langle O_1'; \Delta_1'\rangle <_o \langle O_2'; \Delta_2'\rangle$*

5. *If $|\Delta| \vdash O$ `wellTyped` and $\Delta \vdash O \doteq$ and $\Delta' \vdash \theta : \Delta$ and $\theta O = O'$ then $\langle \Delta; O\rangle =_o \langle \Delta'; O'\rangle$*

6. *If $\langle \Delta; O\rangle <_o \langle \Delta'; O'\rangle$ or $\langle \Delta; O\rangle =_o \langle \Delta'; O'\rangle$ or $\langle \Delta; O\rangle \leq_o \langle \Delta'; O'\rangle$ then $\langle \Delta; O\rangle =_o \langle \Delta; O\rangle$ and $\langle \Delta; O\rangle =_o \langle \Delta; O\rangle$*

Proof: Each property is proved by a straightforward induction, using the properties of semantic domains. Note that we treat the well-foundedness of the lexicographic and simultaneous composition of well-founded orderings as being *a priori* justified (as we have seen in Chapter 3, this corresponds the multiplication of ordinals, and to a weak form of the natural sum of ordinals, respectively). □

Termination measures will be useful in defining a notion of well-founded ordering on queries, but they are not enough. For example, we can safely assume that queries whose goals are of the form $mult\,(s\,M_0)\,M_1\,M_2$ are bigger than queries whose goals of the form $add\,N_0\,N_1\,N_2$ because add never has subgoals of the form $mult$. In general, a goal of the form $a\,\overline{M}$ should be bigger than a goal of the form $b\,\overline{N}$ whenever $b \sqsubset a$. If a and b are mutually recursive (i.e. if $a \equiv b$), then $a\,\overline{M}$ should be bigger than $b\,\overline{N}$ whenever $\langle \texttt{measure}_a\ \overline{M}; \Delta\rangle <_o \langle \texttt{measure}_b\ \overline{N}; \Delta'\rangle$. If a and b are mutually recursive but distinct, and $\langle \texttt{measure}_a\ \overline{M}; \Delta\rangle =_o \langle \texttt{measure}_b\ \overline{N}; \Delta'\rangle$, then we can arbitrarily assume that $a\,\overline{M}$ is bigger than $b\,\overline{N}$, provided we make such assumptions consistently; we keep track of this by assigning a tie-breaking natural number $\mathtt{tb}_a$ to each a in Σ.

$$\textbf{Termination Vectors} \quad V \quad ::= \quad \langle a; \langle O; \Delta\rangle; m\rangle$$

Termination vectors are computed from goals using the (deterministic) function $\mathtt{size}_\Delta(A)$. As usual, we view the defining equations of $\mathtt{size}_\Delta(A)$ as shorthand for a judgmental whose totality is witnessed by an inductive proof, in this case on the structure of A.

$$\begin{array}{rcll} \mathtt{size}_\Delta(a\,\overline{M}) & = & \langle a; \langle \mathtt{measure}_a\ \overline{M}; \Delta\rangle; \mathtt{tb}_a\rangle & \\ \mathtt{size}_\Delta(\Pi x{:}A.B) & = & V & \text{if } \mathtt{size}_{\Delta,\forall x:A}(B) = V \end{array}$$

Termination vectors are strictly ordered (lexicographically) by the relation $<_v$ and compared for equality using the relation $=_v$, each of which is defined by the rules below.

$$\frac{a \sqsubset b}{\langle a; \langle O; \Delta\rangle; m\rangle <_v \langle b; \langle O'; \Delta'\rangle; m'\rangle} \qquad \frac{a \equiv b \quad \langle O; \Delta\rangle <_o \langle O'; \Delta'\rangle}{\langle a; \langle O; \Delta\rangle; m\rangle <_v \langle b; \langle O'; \Delta'\rangle; m'\rangle}$$

$$\frac{a \equiv b \quad \langle O; \Delta\rangle =_o \langle O'; \Delta'\rangle \quad m < m'}{\langle a; \langle O; \Delta\rangle; m\rangle <_v \langle b; \langle O'; \Delta'\rangle; m'\rangle} \qquad \frac{a \equiv b \quad \langle M; \Delta\rangle =_o \langle M'; \Delta'\rangle \quad m = m'}{\langle a; \langle O; \Delta\rangle; m\rangle =_v \langle b; \langle O'; \Delta'\rangle; m'\rangle}$$

Termination vectors have the following properties.

Lemma 5.2.46 (Termination Vectors)

1. *$<_v$ is a transitive, well-founded ordering*

2. *$=_v$ is transitive and symmetric*

3. *If $V_1 =_v V_1'$ and $V_2 = V_2'$ and $V_1 <_v V_2$ then $V_1' <_v V_2'$*

Proof: Direct. We treat the well-foundedness of $<$ on natural numbers and the well-foundedness of the lexicographic composition of well-founded orderings as being *a priori* justified (the latter corresponds to ordinal multiplication). □

Termination vectors give us a well-founded ordering on queries whose input arguments are ground, which will play an important role in proving that the execution of a mode/termination checked logic program terminates. However, the mode/termination checker reasons about logic programs statically, and thus must be able to reach conclusions of the form "although O_1 and O_2 may not be ground, given any θ such that θO_1 and θO_2 are both ground, it must be the case that θO_1 is smaller than θO_2." To this end, we define a notion of *ordering formulas.* Although we reuse the letters F and G in defining ordering formulas, they are not to be confused with the formulas from the assertion logics from Chapter 2 and Chapter 4.

Ordering Formulas $F, G \ ::= \ \top \mid O_1 \prec O_2 \mid O_1 \preceq O_2 \mid O_1 \approx O_2 \mid \nabla x{:}A.F \mid F \wedge G$

Ordering Contexts $\Phi \ ::= \ \cdot \mid \Phi, F$

We use a sequent calculus provability judgment of the form $\Delta; \Phi \vdash F$ to mean that the formula F is a logical consequence of the formulas in Φ, and we expect sequents such as $\exists x{:}nat; \cdot \vdash x \prec s\,x$ to be provable. This is because, for any ground M of type nat, the inequality $\langle \cdot; M\rangle <_o \langle \cdot; s\,M\rangle$ should hold, and we expect the sequent calculus to treat existentially-quantified variables as place-holders for arbitrary ground terms. However, if it happens to be the case that occurrences of $\prec$, $\preceq$ and $\approx$ in F are compatible with the orderings $<_o$, $\leq_o$ and $=_o$ then we say that F is `valid`, which we define (and generalize to ordering contexts) below.

$$\frac{}{\Delta \vdash \top\ \texttt{valid}} \qquad \frac{\langle O_1; \Delta\rangle <_o \langle O_2; \Delta\rangle}{\Delta \vdash O_1 \prec O_2\ \texttt{valid}} \qquad \frac{\langle O_1; \Delta\rangle \leq_o \langle O_2; \Delta\rangle}{\Delta \vdash O_1 \preceq O_2\ \texttt{valid}}$$

$$\frac{\langle O_1; \Delta\rangle =_o \langle O_2; \Delta\rangle}{\Delta \vdash O_1 \approx O_2\ \texttt{valid}} \qquad \frac{\Delta, \forall x{:}A \vdash F\ \texttt{valid}}{\Delta \vdash \nabla x{:}A.F} \qquad \frac{\Delta \vdash F\ \texttt{valid} \quad \Delta \vdash G\ \texttt{valid}}{\Delta \vdash F \wedge G\ \texttt{valid}}$$

$$\frac{}{\Delta \vdash \cdot\ \texttt{valid}} \qquad \frac{\Delta \vdash F\ \texttt{valid} \quad \Delta \vdash \Phi\ \texttt{valid}}{\Delta \vdash \Phi, F\ \texttt{valid}}$$

Because the definition of `valid` depends on the choice of semantic domain, and because we expect ground formulas to be `valid` whenever they are provable, the sequent calculus must be sound with respect to the semantic domain; we make this precise in Definition 5.2.47. As with the semantic domain, we leave the exact formulation of the rules for the sequent calculus abstract, although an example of one which is compatible with the semantic domain of Example 5.2.42 can be found in [Pie05]; we leave the formal specification and implementation of a sequent calculus which is compatible with the semantic domain of Example 5.2.41 to future work, although a prototype of an implementation of the Twelf termination checker based on one candidate is available from `http://www.twelf.org/lpo/`.

We lift the notion of `wellTyped`, $\overset{\shortmid}{=}$ and generalized substitution application to ordering formulas and ordering contexts, the rules of which are straightforward, but made precise below.

$$\frac{}{\Gamma \vdash \top\ \texttt{wellTyped}} \qquad \frac{\Gamma \vdash O_1\ \texttt{wellTyped} \quad \Gamma \vdash O_2\ \texttt{wellTyped}}{\Gamma \vdash (O_1 \prec O_2)\ \texttt{wellTyped}}$$

$$\frac{\Gamma \vdash O_1\ \texttt{wellTyped} \quad \Gamma \vdash O_2\ \texttt{wellTyped}}{\Gamma \vdash (O_1 \preceq O_2)\ \texttt{wellTyped}} \qquad \frac{\Gamma \vdash O_1\ \texttt{wellTyped} \quad \Gamma \vdash O_2\ \texttt{wellTyped}}{\Gamma \vdash (O_1 \approx O_2)\ \texttt{wellTyped}}$$

$$\frac{\Gamma, x{:}A \vdash F\ \texttt{wellTyped}}{\Gamma \vdash (\nabla x{:}A.F)\ \texttt{wellTyped}} \qquad \frac{\Gamma \vdash F\ \texttt{wellTyped} \quad \Gamma \vdash G\ \texttt{wellTyped}}{\Gamma \vdash F \wedge G\ \texttt{wellTyped}}$$

$$\frac{}{\Gamma \vdash (\cdot)\ \texttt{wellTyped}} \qquad \frac{\Gamma \vdash \Phi\ \texttt{wellTyped} \quad \Gamma \vdash F\ \texttt{wellTyped}}{\Gamma \vdash (\Phi, F)\ \texttt{wellTyped}}$$

$$\frac{}{\Delta \vdash \top \overset{\shortmid}{=}} \qquad \frac{\Delta \vdash O_1 \overset{\shortmid}{=} \quad \Delta \vdash O_2 \overset{\shortmid}{=}}{\Delta \vdash (O_1 \prec O_2) \overset{\shortmid}{=}} \qquad \frac{\Delta \vdash O_1 \overset{\shortmid}{=} \quad \Delta \vdash O_2 \overset{\shortmid}{=}}{\Delta \vdash (O_1 \preceq O_2) \overset{\shortmid}{=}}$$

$$\frac{\Delta \vdash O_1 \overset{\shortmid}{=} \quad \Delta \vdash O_2 \overset{\shortmid}{=}}{\Delta \vdash (O_1 \approx O_2) \overset{\shortmid}{=}} \qquad \frac{\Delta, \forall x{:}A \vdash F \overset{\shortmid}{=}}{\Delta \vdash (\nabla x{:}A.F) \overset{\shortmid}{=}} \qquad \frac{\Delta \vdash F \overset{\shortmid}{=} \quad \Delta \vdash G \overset{\shortmid}{=}}{\Delta \vdash F \wedge G \overset{\shortmid}{=}}$$

$$\frac{}{\Delta \vdash (\cdot) \overset{\shortmid}{=}} \qquad \frac{\Delta \vdash \Phi \overset{\shortmid}{=} \quad \Delta \vdash F \overset{\shortmid}{=}}{\Delta \vdash (\Phi, F) \overset{\shortmid}{=}}$$

$$\frac{}{\eta \vdash \top \doteqdot} \qquad \frac{\eta \vdash O_1 \doteqdot \quad \eta \vdash O_2 \doteqdot}{\eta \vdash (O_1 \prec O_2) \doteqdot} \qquad \frac{\eta \vdash O_1 \doteqdot \quad \eta \vdash O_2 \doteqdot}{\eta \vdash (O_1 \preceq O_2) \doteqdot}$$

$$\frac{\eta \vdash O_1 \doteqdot \quad \eta \vdash O_2 \doteqdot}{\eta \vdash (O_1 \approx O_2) \doteqdot} \qquad \frac{\eta, \forall x{:}A \vdash F \doteqdot}{\eta \vdash (\nabla x{:}A.F) \doteqdot} \qquad \frac{\eta \vdash F \doteqdot \quad \eta \vdash G \doteqdot}{\eta \vdash F \wedge G \doteqdot}$$

$$\frac{}{\eta \vdash (\cdot) \doteqdot} \qquad \frac{\eta \vdash \Phi \doteqdot \quad \eta \vdash F \doteqdot}{\eta \vdash (\Phi, F) \doteqdot}$$

$$\begin{array}{rcll}
\theta\top & = & \top & \\
\theta\,(O_1 \prec O_2) & = & O_1' \prec O_2' & \text{if } \theta\,O_1 = O_1' \text{ and } \theta O_2 = O_2' \\
\theta\,(O_1 \preceq O_2) & = & O_1' \preceq O_2' & \text{if } \theta\,O_1 = O_1' \text{ and } \theta O_2 = O_2' \\
\theta\,(O_1 \approx O_2) & = & O_1' \approx O_2' & \text{if } \theta\,O_1 = O_1' \text{ and } \theta O_2 = O_2' \\
\theta\,(\nabla x{:}A.F) & = & \nabla y{:}B.G & \text{if } \theta A = B \text{ and } (\theta, y/x)F = G \\
\theta(F \wedge G) & = & F' \wedge G' & \text{if } \theta F = F' \text{ and } \theta G = G' \\
\theta\,(\cdot) & = & \cdot & \\
\theta\,(\Phi, F) & = & \Phi', G & \text{if } \theta\Phi = \Phi' \text{ and } \theta F = G
\end{array}$$

Definition 5.2.47 (Soundness of Denotations and Proofs) *Given the notion of* `valid` *induced by a given semantic domain, we say that the judgment* $\Delta; \Phi \vdash F$ *is* sound *iff the following propositions hold.*

1. *If* $\Delta; \Phi \vdash F$ *and* Δ *can be obtained from* Δ' *by deleting some number of declarations then* $\Delta'; \Phi \vdash F$

2. *If* $\Delta; \Phi \vdash F$ *and* $\Delta \vdash \Phi$ `wellTyped` *and* $\Delta \vdash F$ `wellTyped` *and* $\Delta' \vdash \theta : \Delta$ *and* $\theta F = F'$ *and* $\theta\Phi = \Phi'$ *then* $\Delta'; \Phi' \vdash F'$

3. *If* $\Delta; \Phi \vdash F$ *and* $\Delta \vdash \Phi$ `wellTyped` *and* $\Delta \vdash \Phi \doteqdot$ *and* $\Delta \vdash \Phi$ `valid` *and* $\Delta \vdash F$ `wellTyped` *and* $\Delta \vdash F \doteqdot$ *then* $\Delta \vdash F$ `valid`

Sometimes it is useful to know that solved goals always satisfy certain invariants. For example, once the goal *add* $M_0\ M_1\ M_2$ is solved, we should know that the invariant

$M_1 \preceq M_2$ is valid. In general, for each $a{:}K$ in Σ, the user must define the parametric function $\mathtt{redInv}_a$ that maps spines of the appropriate length to formulas (i.e. if $K = \Pi x_1{:}A_1.\ldots\ \Pi x_n{:}A_n.\mathit{type}$ then $\mathtt{redInv}_a\ (M_1 :: \ldots :: M_n :: \cdot)$ is built up from $M_1 \ldots M_n$ uniformly without regards to the structure of each M_i). Reduction invariants satisfy the following properties.

Lemma 5.2.48 (Reduction Invariants)

1. *If* $\Gamma \vdash a\,\overline{M} : \mathit{type}$ *then* $\Gamma \vdash \mathtt{redInv}_a\ \overline{M}\ \mathtt{wellTyped}$
2. *If* $\theta(a\,\overline{M}) = a\,\overline{M'}$ *and* $\mathtt{redInv}_a\ \overline{M} = F$ *then* $\theta F = F'$ *and* $\mathtt{redInv}_a\ \overline{M'} = F'$
3. *If* $\Delta \vdash a\,\overline{M} \doteqdot$ *then* $\Delta \vdash \mathtt{redInv}_a\ \overline{M} \doteqdot$
4. *If* $\eta \vdash a\,\overline{M} \doteqdot$ *then* $\eta \vdash \mathtt{redInv}_a\ \overline{M} \doteqdot$

Proof: Direct, by the definition of $\mathtt{redInv}_a$. □

The following lemmas lift some of the useful properties of LF terms and spines to measures, ordering formulas and contexts.

Lemma 5.2.49 (Gen. Substitutions on Measures, Formulas, Contexts)

1. (a) *If* $\theta O = O'$ *and* $\theta O = O''$ *then* $O' = O''$
   (b) *If* $\theta F = F'$ *and* $\theta F = F''$ *then* $F' = F''$
   (c) *If* $\theta \Phi = \Phi'$ *and* $\theta \Phi = \Phi''$ *then* $\Phi' = \Phi''$
2. *If* $\Delta' \vdash \theta : \Delta$ *then:*
   (a) *If* $|\Delta| \vdash O\ \mathtt{wellTyped}$ *then* $\theta O = O'$ *and* $|\Delta'| \vdash O'\ \mathtt{wellTyped}$
   (b) *If* $|\Delta| \vdash F\ \mathtt{wellTyped}$ *then* $\theta F = F'$ *and* $|\Delta'| \vdash F'\ \mathtt{wellTyped}$
   (c) *If* $|\Delta| \vdash \Phi\ \mathtt{wellTyped}$ *then* $\theta \Phi = \Phi'$ *and* $|\Delta'| \vdash \Phi'\ \mathtt{wellTyped}$

Proof: Each case in 1 is by straightforward induction on the derivation of either generalized substitution application. 1(c) uses 1(b), 1(b) uses 1(a), 1(a) uses Lemma 5.2.2. Each case in 2 is by straightforward induction on the given typing derivation; 2(c) uses 2(b), 2(b) uses 2(a) and 2(a) uses Lemma 5.2.7. □

Lemma 5.2.50 (Weakening for Measures, Formulas, Contexts)
If $\Gamma \vdash A : type$ *and* $x\#A$ *then:*

1. *if* $\Gamma, \Gamma' \vdash O$ `wellTyped` *then* $\Gamma, x{:}A, \Gamma' \vdash O$ `wellTyped`

2. *if* $\Gamma, \Gamma' \vdash F$ `wellTyped` *then* $\Gamma, x{:}A, \Gamma' \vdash F$ `wellTyped`

3. *if* $\Gamma, \Gamma' \vdash \Phi$ `wellTyped` *then* $\Gamma, x{:}A, \Gamma' \vdash \Phi$ `wellTyped`

Proof: By induction on the structure of the given typing derivation; 3 uses 2, 2 uses 1, and 1 uses Lemma 5.1.6. □

Lemma 5.2.51 (Weakening for Ground On Measures, Formulas, Contexts)

1. *If* Δ *can be obtained from* Δ' *by deleting some number of declarations then:*

   (a) *If* $\Delta \vdash O \doteq$ *then* $\Delta' \vdash O \doteq$

   (b) *If* $\Delta \vdash F \doteq$ *then* $\Delta' \vdash F \doteq$

   (c) *If* $\Delta \vdash \Phi \doteq$ *then* $\Delta' \vdash \Phi \doteq$

2. *If* η *can be obtained from* η' *by deleting some number of declarations then:*

   (a) *If* $\eta \vdash O \doteq$ *then* $\eta' \vdash O \doteq$

   (b) *If* $\eta \vdash F \doteq$ *then* $\eta' \vdash F \doteq$

   (c) *If* $\eta \vdash \Phi \doteq$ *then* $\eta' \vdash \Phi \doteq$

Proof: By straightforward induction on the structure of the given groundness derivations. Each (c) case uses the corresponding (b) case, each (b) case uses the corresponding (a) case, and each (a) case uses Lemma 5.2.18. □

Lemma 5.2.52 (Preservation of Ground on Measures, Formulas, Contexts)

1. If $\Delta' \vdash \theta : \Delta$ then:

(a) If $\Delta \vdash O \,\underline{\top}$ and $\theta O = O'$ then $\Delta' \vdash O' \,\underline{\top}$

(b) If $\Delta \vdash F \,\underline{\top}$ and $\theta F = F'$ then $\Delta' \vdash F' \,\underline{\top}$

(c) If $\Delta \vdash \Phi \,\underline{\top}$ and $\theta \Phi = \Phi'$ then $\Delta' \vdash \Phi' \,\underline{\top}$

2. If $\Delta \vdash \theta : \eta$ then:

(a) $\eta \vdash O \,\underline{\top}$ and $\theta O = O'$ then $\Delta \vdash O' \,\underline{\top}$

(b) $\eta \vdash F \,\underline{\top}$ and $\theta F = F'$ then $\Delta \vdash F' \,\underline{\top}$

(c) $\eta \vdash \Phi \,\underline{\top}$ and $\theta \Phi = \Phi'$ then $\Delta \vdash \Phi' \,\underline{\top}$

3. If $\eta' \vdash \theta : \eta$ then:

(a) $\eta \vdash O \,\underline{\top}$ and $\theta O = O'$ then $\eta' \vdash O' \,\underline{\top}$

(b) $\eta \vdash F \,\underline{\top}$ and $\theta F = F'$ then $\eta' \vdash F' \,\underline{\top}$

(c) $\eta \vdash \Phi \,\underline{\top}$ and $\theta \Phi = \Phi'$ then $\eta' \vdash \Phi' \,\underline{\top}$

Proof: Each case is by a straightforward induction on the given derivation of groundness. Each (c) case uses the corresponding (b) case, each (b) case uses the corresponding (a) case, and each (a) case uses Lemma 5.2.20. □

Lemma 5.2.53 (Substitution for Validity) *If $\Delta' \vdash \theta : \Delta$ then:*

1. *if* $|\Delta| \vdash F$ `wellTyped` *and* $\Delta \vdash F \doteq$ *and* $\Delta \vdash F$ `valid` *and* $\theta F = F'$ *then* $\Delta' \vdash F'$ `valid`

2. *if* $|\Delta| \vdash \Phi$ `wellTyped` *and* $\Delta \vdash \Phi \doteq$ *and* $\Delta \vdash \Phi$ `valid` *and* $\theta\Phi = \Phi'$ *then* $\Delta' \vdash \Phi'$ `valid`

Proof: By straightforward induction on the derivation of `valid`; 1 uses Lemma 5.2.45 and 2 uses 1. □

Lemma 5.2.54 (Weakening for Validity) *If Δ can be obtained from Δ' by deleting some number of declarations, then:*

1. *if* $\Delta \vdash F$ `valid` *then* $\Delta' \vdash F$ `valid`

2. *if* $\Delta \vdash \Phi$ `valid` *then* $\Delta' \vdash \Phi$ `valid`

Proof: By straightforward induction on the structure of `valid`; 1 uses Lemma 5.2.45 and 2 uses 1. □

5.2.5 The Mode and Termination Checker

We are now ready to define the mode/termination checker. We will find it useful to make lists of LF types; we abuse notation slightly by overloading the notation used for spines.

$$\text{Type Lists} \quad \overline{A}, \overline{B} \quad ::= \quad \cdot \mid A :: \overline{A}$$

We say that $\Gamma \vdash \overline{A}$ `wellTyped` if every A in $\overline{A}$ is well typed.

$$\frac{}{\Gamma \vdash \cdot\ \texttt{wellTyped}} \qquad \frac{\Gamma \vdash A : \mathit{type} \quad \Gamma \vdash \overline{A}\ \texttt{wellTyped}}{\Gamma \vdash A :: \overline{A}\ \texttt{wellTyped}}$$

We lift generalized substitution application analogously. The following lemma lifts some of the useful properties of LF types to type lists.

Lemma 5.2.55 (Type Lists)

1. *If* $\Gamma, \Gamma' \vdash \overline{A}$ `wellTyped` *and* $\Gamma \vdash B : type$ *then* $\Gamma, x{:}A, \Gamma' \vdash \overline{A}$ `wellTyped`

2. *If* $\theta\overline{A} = \overline{A'}$ *and* $\theta\overline{A} = \overline{A''}$ *then* $\overline{A'} = \overline{A''}$

3. *If* $x\#\overline{A}$ *and* $(\theta, y/x)\overline{A} = \overline{A'}$ *or* $(\theta, M{:}B/x)\overline{A} = \overline{A'}$ *then* $\theta\overline{A} = \overline{A'}$

4. *If* $\Delta' \vdash \theta : \Delta$ *and* $|\Delta| \vdash \overline{A}$ `wellTyped` *then* $\theta\overline{A} = \overline{A'}$ *and* $|\Delta'| \vdash \overline{A}$ `wellTyped`

Below we define the mode/termination checker for Twelf. We assume that we are given a semantic domain $\mathbb{S}$ and a sequent calculus $\Delta; \Phi \vdash F$ which is sound with respect to $\mathbb{S}$. The mode/termination checker is specified judgmentally below, and is inspired by the mode and termination checking algorithms sketched in [RP96, Roh96] and the reduces checking algorithm specified in [Pie05]. In other words, it is essentially a syntactic specification for the combination of the mode, termination and reduces checkers implemented in Twelf, which has been used to formally verify the correctness of all the proofs in Chapter 2 (where $\mathbb{SS}$ is defined in Example 5.2.42 and $\Delta; \Psi \vdash i$s the sequent calculus specified in [Pie05]) and in Chapter 4 (where $\mathbb{SS}$ is a generalization of the one defined in Example 5.2.41 and the formal specification of $\Delta; F \vdash i$s left to future work); see `http://www.twelf.org/slr` and `http://www.twelf.org/lpo` for the source files. We do not require that the notion of provability for $\Delta; \Phi \vdash F$ be decidable, but if it is, then so is mode/termination checking.

The judgment $\vdash \Sigma'$ `tc` a should be read as saying "a is a mode/termination-checked logic program in the signature Σ'" (where Σ' should be thought of as a subset of the given LF signature Σ). Similarly, $\Delta \vdash \Delta'$ `tc` a should be read as "a is a mode/termination-checked logic program in the context Δ' (which should be thought of as a subset of Δ)"; $\eta \vdash \eta'$ `tc` a is the lifting of $\Delta \vdash \Delta'$ `tc` a to abstract substitutions.

All these judgments can be interpreted functionally by induction/recursion on the structure of Σ', Δ' and η', respectively. In general, a logic-program a might call another logic-program b. If a and b are mutually recursive, then $b \equiv a$ must hold, in which case the aforementioned judgments will mode/termination-check b explicitly. If a can call b, but not vice-versa, then $b \sqsubset a$ must hold, in which case b will only be mode/termination-checked when we consider the well-behavedness of the program clauses of a that call b.

$$\frac{}{\vdash \cdot \ \texttt{tc}\ a} \qquad \frac{\vdash \Sigma' \ \texttt{tc}\ a}{\vdash \Sigma', b{:}K \ \texttt{tc}\ a} \qquad \frac{\texttt{hd}(A) \not\equiv a \quad \vdash \Sigma' \ \texttt{tc}\ a}{\vdash \Sigma', c{:}A \ \texttt{tc}\ a}$$

$$\frac{\texttt{hd}(A) \equiv a \quad \vdash \Sigma' \ \texttt{tc}\ a \quad \cdot \vdash A \ \texttt{okD}\ \cdot}{\vdash \Sigma', c{:}A \ \texttt{tc}\ a}$$

$$\frac{}{\Delta \vdash \cdot \ \texttt{tc}\ a} \qquad \frac{\Delta \vdash \Delta' \ \texttt{tc}\ a}{\Delta \vdash \Delta', \exists x{:}A \ \texttt{tc}\ a} \qquad \frac{\texttt{hd}(A) \not\equiv a \quad \Delta \vdash \Delta' \ \texttt{tc}\ a}{\Delta \vdash \Delta', \forall x{:}A \ \texttt{tc}\ a}$$

$$\frac{\texttt{hd}(A) \equiv a \quad \Delta \vdash \Delta' \ \texttt{tc}\ a \quad \Delta \vdash A \ \texttt{okD}\ \cdot}{\Delta \vdash \Delta', \forall x{:}A \ \texttt{tc}\ a}$$

$$\frac{}{\eta \vdash \cdot \ \texttt{tc}\ a} \qquad \frac{\eta \vdash \eta' \ \texttt{tc}\ a}{\eta \vdash \eta', \exists^{?} x{:}A \ \texttt{tc}\ a} \qquad \frac{\eta \vdash \eta' \ \texttt{tc}\ a}{\eta \vdash \eta', \exists^{\pm} x{:}A \ \texttt{tc}\ a} \qquad \frac{\texttt{hd}(A) \not\equiv a \quad \eta \vdash \eta' \ \texttt{tc}\ a}{\eta \vdash \eta', \forall x{:}A \ \texttt{tc}\ a}$$

$$\frac{\texttt{hd}(A) \equiv a \quad \eta \vdash \eta' \ \texttt{tc}\ a \quad \eta \vdash A \ \texttt{okD}\ \cdot}{\eta \vdash \eta', \forall x{:}A \ \texttt{tc}\ a}$$

The judgment $\Delta \vdash A$ `okG` can be thought of as saying "performing search on the mode-compatible query $\Delta \vdash \boxed{?} : A$ is guaranteed to terminate." Similarly, the judgment $\eta \vdash A$ `okG` can be thought of as saying that, for every θ approximated by η, querying the goal formula θA will terminate. Both judgments can be interpreted functionally by induction/recursion on A.

$$\frac{|\Delta| \vdash A : \mathit{type} \quad \Delta, \forall x{:}A \vdash B \ \texttt{okG}}{\Delta \vdash \Pi x{:}A.B \ \texttt{okG}}$$

$$\frac{|\Delta| \vdash a\,\overline{M} : \mathit{type} \quad \vdash \Sigma \ \texttt{tc} \ a \quad \Delta \vdash \Delta \ \texttt{tc} \ a \quad \Delta \vdash \texttt{inputs}_a(\overline{M}) \doteqdot}{\Delta \vdash a\,\overline{M} \ \texttt{okG}}$$

$$\frac{|\eta| \vdash A : \mathit{type} \quad \eta, \forall x{:}A \vdash B \ \texttt{okG}}{\eta \vdash \Pi x{:}A.B \ \texttt{okG}}$$

$$\frac{|\eta| \vdash a\,\overline{M} : \mathit{type} \quad \vdash \Sigma \ \texttt{tc} \ a \quad \eta \vdash \eta \ \texttt{tc} \ a \quad \eta \vdash \texttt{inputs}_a(\overline{M}) \doteqdot}{\eta \vdash a\,\overline{M} \ \texttt{okG}}$$

The judgment $\Delta \vdash A \ \texttt{okD} \ \overline{A}$ can be thought of as saying that the program declaration A, when used to solve any goal whose head matches A, will terminate, as will solving the left-over subgoals $\overline{A}$. This judgment is lifted to abstract substitutions in the usual way (i.e. by replacing Δ with η). Both judgments can be interpreted functionally by induction/recursion on A.

$$\frac{|\Delta| \vdash A : \mathit{type} \quad \Delta, \exists x{:}A \vdash B \ \texttt{okD} \ \overline{A} \quad x \in FV(B)}{\Delta \vdash \Pi x{:}A.B \ \texttt{okD} \ \overline{A}} \qquad \frac{\Delta \vdash B \ \texttt{okD} \ A :: \overline{A}}{\Delta \vdash A \to B \ \texttt{okD} \ \overline{A}}$$

$$\frac{\Delta \vdash \texttt{inputs}_a(\overline{M}) \doteqdot\Longrightarrow \eta \quad \eta \vdash \texttt{inputs}_a(\overline{M}) \doteqdot \quad \eta; \cdot \vdash \overline{A} \ \texttt{okSGs} \ a\,\overline{M}}{\Delta \vdash a\,\overline{M} \ \texttt{okD} \ \overline{A}}$$

$$\frac{|\eta| \vdash A : \mathit{type} \quad \eta, \exists^? x{:}A \vdash B \ \texttt{okD} \ \overline{A} \quad x \in FV(B)}{\eta \vdash \Pi x{:}A.B \ \texttt{okD} \ \overline{A}} \qquad \frac{\eta \vdash B \ \texttt{okD} \ A :: \overline{A}}{\eta \vdash A \to B \ \texttt{okD} \ \overline{A}}$$

$$\frac{\eta \vdash \texttt{inputs}_a(\overline{M}) \doteqdot\Longrightarrow \eta' \quad \eta' \vdash \texttt{inputs}_a(\overline{M}) \doteqdot \quad \eta'; \cdot \vdash \overline{A} \ \texttt{okSGs} \ a\,\overline{M}}{\eta \vdash a\,\overline{M} \ \texttt{okD} \ \overline{A}}$$

The judgment $\Delta; \Phi \vdash \overline{A} \ \texttt{okSGs} \ a\,\overline{M}$ means that, if the reduction invariants Φ are valid in Δ, then solving all of the subgoals of $a\,\overline{M}$ in $\overline{A}$ one-at-a-time, in order, will terminate. This is also lifted to generalized substitutions in the usual way. These judgments can be interpreted functionally by induction/recursion on $\overline{A}$.

$$\frac{|\Delta| \vdash a\,\overline{M} : \mathit{type} \quad \Delta \vdash \mathtt{outputs}_a(\overline{M}) \mathrel{\underline{\underline{\downarrow}}} \quad \Delta; \Phi \vdash \mathtt{redInv}_a\ \overline{M}}{\eta; \Phi \vdash \cdot\ \mathtt{okSGs}\ a\,\overline{M}}$$

$$\frac{|\Delta| \vdash A : \mathit{type} \quad \Delta; \Phi \vdash A\ \mathtt{okSG}\ a\,\overline{M} \Longrightarrow \langle \eta; F\rangle \quad \eta; \Phi, F \vdash \overline{A}\ \mathtt{okSGs}\ a\,\overline{M}}{\Delta; \Phi \vdash A :: \overline{A}\ \mathtt{okSGs}\ a\,\overline{M}}$$

$$\frac{|\eta| \vdash a\,\overline{M} : \mathit{type} \quad \eta \vdash \mathtt{outputs}_a(\overline{M}) \mathrel{\underline{\underline{\downarrow}}} \quad \Delta_\eta; \Phi \vdash \mathtt{redInv}_a\ \overline{M}}{\eta; \Phi \vdash \cdot\ \mathtt{okSGs}\ a\,\overline{M}}$$

$$\frac{|\Delta| \vdash A : \mathit{type} \quad \eta; \Phi \vdash A\ \mathtt{okSG}\ a\,\overline{M} \Longrightarrow \langle \eta'; F\rangle \quad \eta'; \Phi, F \vdash \overline{A}\ \mathtt{okSGs}\ a\,\overline{M}}{\eta; \Phi \vdash A :: \overline{A}\ \mathtt{okSGs}\ a\,\overline{M}}$$

The judgment $\Delta; \Phi \vdash A\ \mathtt{okSG}\ a\,\overline{M} \Longrightarrow \langle \Phi; F\rangle$ means that, if Φ is assumed to be valid in Δ, then A can be shown to be an "ok" goal which is smaller than $a\,\overline{M}$, and moreover, when A has been successfully solved, the output-substitution will be approximated by η, and the invariant F will be valid in the output-context. This judgment is lifted to η in usual way. Both judgments can be interpreted functionally by induction/recursion on A.

$$\frac{\begin{array}{c}\mathtt{hd}(A) \equiv a \quad |\Delta| \vdash A : \mathit{type} \quad \Delta, \forall x{:}A \vdash A \ \mathtt{okD} \ \cdot \\ \Delta, \forall x{:}A; \Phi \vdash B \ \mathtt{okSG} \ a\,\overline{M} \Longrightarrow \langle \eta, \forall x{:}A; F \rangle\end{array}}{\Delta; \Phi \vdash \Pi x{:}A.B \ \mathtt{okSG} \ a\,\overline{M} \Longrightarrow \langle \eta; \nabla x{:}A.F \rangle}$$

$$\frac{\mathtt{hd}(A) \not\equiv a \quad |\Delta| \vdash A : \mathit{type} \quad \Delta, \forall x{:}A; \Phi \vdash B \ \mathtt{okSG} \ a\,\overline{M} \Longrightarrow \langle \eta, \forall x{:}A; F \rangle}{\Delta; \Phi \vdash \Pi x{:}A.B \ \mathtt{okSG} \ a\,\overline{M} \Longrightarrow \langle \eta; \nabla x{:}A.F \rangle}$$

$$\frac{\begin{array}{c}b \equiv a \quad \mathtt{tb}_b \geq \mathtt{tb}_a \quad \Delta \vdash \mathtt{inputs}_b(\overline{N}) \doteqdot \\ \Delta; \Phi \vdash \mathtt{measure}_b \ \overline{N} \prec \mathtt{measure}_a \ \overline{M} \quad \Delta \vdash \mathtt{outputs}_b(\overline{N}) \doteqdot\Longrightarrow \eta\end{array}}{\Delta; \Phi \vdash b\,\overline{N} \ \mathtt{okSG} \ a\,\overline{M} \Longrightarrow \langle \eta; \mathtt{redInv}_b \ \overline{N} \rangle}$$

$$\frac{\begin{array}{c}b \equiv a \quad \mathtt{tb}_b < \mathtt{tb}_a \quad \Delta \vdash \mathtt{inputs}_b(\overline{N}) \doteqdot \\ \Delta; \Phi \vdash \mathtt{measure}_b \ \overline{N} \preceq \mathtt{measure}_a \ \overline{M} \quad \Delta \vdash \mathtt{outputs}_b(\overline{N}) \doteqdot\Longrightarrow \eta\end{array}}{\Delta; \Phi \vdash b\,\overline{N} \ \mathtt{okSG} \ a\,\overline{M} \Longrightarrow \langle \eta; \mathtt{redInv}_b \ \overline{N} \rangle}$$

$$\frac{b \sqsubset a \quad \Delta \vdash \mathtt{inputs}_b(\overline{N}) \doteqdot \quad \vdash \Sigma \ \mathtt{tc} \ b \quad \Delta \vdash \Delta \ \mathtt{tc} \ b \quad \Delta \vdash \mathtt{outputs}_b(\overline{N}) \doteqdot\Longrightarrow \eta}{\Delta; \Phi \vdash b\,\overline{N} \ \mathtt{okSG} \ a\,\overline{M} \Longrightarrow \langle \eta; \mathtt{redInv}_b \ \overline{N} \rangle}$$

$$\frac{\begin{array}{c}\mathtt{hd}(A) \equiv a \quad |\eta| \vdash A : \mathit{type} \quad \eta, \forall x{:}A \vdash A \ \mathtt{okD} \ \cdot \\ \eta, \forall x{:}A; \Phi \vdash B \ \mathtt{okSG} \ a\,\overline{M} \Longrightarrow \langle \eta', \forall x{:}A; F\rangle\end{array}}{\eta; \Phi \vdash \Pi x{:}A.B \ \mathtt{okSG} \ a\,\overline{M} \Longrightarrow \langle \eta'; \nabla x{:}A.F\rangle}$$

$$\frac{\mathtt{hd}(A) \not\equiv a \quad |\eta| \vdash A : \mathit{type} \quad \eta, \forall x{:}A; \Phi \vdash B \ \mathtt{okSG} \ a\,\overline{M} \Longrightarrow \langle \eta', \forall x{:}A; F\rangle}{\eta; \Phi \vdash \Pi x{:}A.B \ \mathtt{okSG} \ a\,\overline{M} \Longrightarrow \langle \eta'; \nabla x{:}A.F\rangle}$$

$$\frac{\begin{array}{c}b \equiv a \quad \mathtt{tb}_b \geq \mathtt{tb}_a \quad \eta \vdash \mathtt{inputs}_b(\overline{N}) \doteqdot \quad \Delta_\eta; \Phi \vdash \mathtt{measure}_b \ \overline{N} \prec \mathtt{measure}_a \ \overline{M} \\ \eta \vdash \mathtt{outputs}_b(\overline{M}) \doteqdot\!\!\Longrightarrow \eta' \quad \eta' \vdash \mathtt{redInv}_b \ \overline{N} \doteqdot\end{array}}{\eta; \Phi \vdash b\,\overline{N} \ \mathtt{okSG} \ a\,\overline{M} \Longrightarrow \langle \eta'; \mathtt{redInv}_b \ \overline{N}\rangle}$$

$$\frac{\begin{array}{c}b \equiv a \quad \mathtt{tb}_b < \mathtt{tb}_a \quad \eta \vdash \mathtt{inputs}_b(\overline{N}) \doteqdot \quad \Delta_\eta; \Phi \vdash \mathtt{measure}_b \ \overline{N} \preceq \mathtt{measure}_a \ \overline{M} \\ \eta \vdash \mathtt{outputs}_b(\overline{M}) \doteqdot\!\!\Longrightarrow \eta' \quad \eta' \vdash \mathtt{redInv}_b \ \overline{N} \doteqdot\end{array}}{\eta; \Phi \vdash b\,\overline{N} \ \mathtt{okSG} \ a\,\overline{M} \Longrightarrow \langle \eta'; \mathtt{redInv}_b \ \overline{N}\rangle}$$

$$\frac{\begin{array}{c}b \sqsubset a \quad \eta \vdash \mathtt{inputs}_b(\overline{N}) \doteqdot \quad \vdash \Sigma \ \mathtt{tc} \ b \quad \eta \vdash \eta \ \mathtt{tc} \ b \\ \eta \vdash \mathtt{outputs}_b(\overline{M}) \doteqdot\!\!\Longrightarrow \eta' \quad \eta' \vdash \mathtt{redInv}_b \ \overline{N} \doteqdot\end{array}}{\eta; \Phi \vdash b\,\overline{N} \ \mathtt{okSG} \ a\,\overline{M} \Longrightarrow \langle \eta'; \mathtt{redInv}_b \ \overline{N}\rangle}$$

Finally, we use the judgment $\Delta \vdash A \ \mathtt{solved}$ to say that the output of solving a query was successful.

$$\frac{\Delta, \forall x{:}A \vdash B \ \mathtt{solved}}{\Delta \vdash \Pi x{:}A.B \ \mathtt{solved}} \qquad \frac{|\Delta| \vdash a\,\overline{M} : \mathit{type} \quad \Delta \vdash \overline{M} \doteqdot \quad \Delta \vdash \mathtt{redInv}_a \ \overline{M} \ \mathtt{valid}}{\Delta \vdash a\,\overline{M} \ \mathtt{solved}}$$

Lemma 5.2.56 (OK Goals are Well Typed) *If* $\Delta \vdash A \ \mathtt{okG}$ *then* $|\Delta| \vdash A : \mathit{type}$

Proof: By a straightforward induction over the given derivation. □

Lemma 5.2.57 (Substitutions Preserve Well-Modedness)

1. *If* $\mathcal{D} : (\Delta_0 \vdash a \ \mathtt{tc} \ \Delta_0, \Delta_1)$ *and* $\Delta_0' \vdash \theta_0 : \Delta_0$ *and* $\Delta_0', \Delta_1' \vdash \theta_0, \theta_1 : \Delta_0, \Delta_1$ *then* $\Delta_0' \vdash a \ \mathtt{tc} \ \Delta_0', \Delta_1'$

2. *If* $\mathcal{D} : (\Delta \vdash A \; \texttt{okD} \; \overline{B})$ *and* $\Delta' \vdash \theta : \Delta$ *and* $\theta A = A'$ *and* $\theta\overline{B} = \overline{B'}$ *then* $\Delta' \vdash A' \; \texttt{okD} \; \overline{B'}$

3. *If* $\mathcal{D} : (\eta; \Phi \vdash \overline{A} \; \texttt{okSGs} \; a\,\overline{M})$ *and* $\eta' \vdash \theta : \eta$ *and* $|\eta| \vdash \Phi \; \texttt{wellTyped}$ *and* $\theta\Phi = \Phi'$ *and* $\theta\overline{A} = \overline{A'}$ *and* $\theta(a\,\overline{M}) = a\,\overline{M'}$ *then* $\eta'; \Phi' \vdash \overline{A'} \; \texttt{okSGs} \; a\,\overline{M'}$

4. *If* $\mathcal{D} : (\eta_0; \Phi \vdash A \; \texttt{okSG} \; a\,\overline{M} \Longrightarrow \langle \eta_1; F \rangle)$ *and* $\eta_0' \vdash \theta : \eta_0$ *and* $|\eta_0| \vdash \Phi \; \texttt{wellTyped}$ *and* $\theta\Phi = \Phi'$ *and* $\theta A = A'$ *and* $\theta(a\,\overline{M}) = a\,\overline{M'}$ *and* $\theta P = P'$ *then there exists* η_1' *s.t.* $\eta_0'; \Phi' \vdash A' \; \texttt{okSG} \; a\,\overline{M'} \Longrightarrow \langle \eta_1'; P' \rangle$ *and* $\eta_1' \vdash \theta : \eta_1$

5. *If* $\mathcal{D} : (\eta \vdash A \; \texttt{okD} \; \overline{B})$ *and* $\eta' \vdash \theta : \eta$ *and* $\theta A = A'$ *and* $\theta\overline{B} = B'$ *then* $\eta' \vdash A' \; \texttt{okD} \; \overline{B'}$

6. *If* $\mathcal{D} : (\eta_0 \vdash a \; \texttt{tc} \; \eta_0, \eta_1)$ *and* $\eta_0', \eta_1' \vdash \theta_0, \theta_1 : \eta_0, \eta_1$ *and* $\eta_0' \vdash \theta_0 : \eta_0$ *then* $\eta_0' \vdash a \; \texttt{tc} \; \eta_0', \eta_1'$

7. *If* $\mathcal{D} : (\eta_0; \Phi \vdash A \; \texttt{okSG} \; a\,\overline{M} \Longrightarrow \langle \eta_1; F \rangle)$ *and* $\Delta \vdash \theta : \eta_0$ *and* $|\eta_0| \vdash \Phi \; \texttt{wellTyped}$ *and* $\theta\Phi = \Phi'$ *and* $\theta A = A'$ *and* $\theta(a\,\overline{M}) = a\,\overline{M'}$ *and* $\theta F = F'$ *then there exists* η_1' *s.t.* $\Delta; \Phi' \vdash A' \; \texttt{okSG} \; a\,\overline{M'} \Longrightarrow \langle \eta_1'; P' \rangle$ *and* $\eta_1' \vdash \theta : \eta_1$

8. *If* $\mathcal{D} : (\eta; \Phi \vdash \overline{A} \; \texttt{okSGs} \; a\,\overline{M})$ *and* $\Delta \vdash \theta : \eta$ *and* $|\eta| \vdash \Phi \; \texttt{wellTyped}$ *and* $\theta\Phi = \Phi'$ *and* $\theta\overline{A} = \overline{A'}$ *and* $\theta(a\,\overline{M}) = a\,\overline{M'}$ *then* $\Delta; \Phi' \vdash \overline{A'} \; \texttt{okSGs} \; a\,\overline{M'}$

9. *If* $\mathcal{D} : (\eta_0 \vdash a \; \texttt{tc} \; \eta_0, \eta_1)$ *and* $\Delta_0, \Delta_1 \vdash \theta_0, \theta_1 : \eta_0, \eta_1$ *and* $\Delta_0 \vdash \theta_0 : \eta_0$ *then* $\Delta_0 \vdash a \; \texttt{tc} \; \Delta_0, \Delta_1$

Proof: By mutual induction on $\mathcal{D}$; most of the work is done by Lemma 5.2.38, Lemma 5.2.20 and Definition 5.2.47. $\square$

Lemma 5.2.58 (Weakening for Mode Checking)

1. *If* $\Delta \vdash \Delta' \; \texttt{tc} \; a$ *and* $|\Delta| \vdash B : type$ *then:*

(a) $\Delta, \exists x{:}B \vdash \Delta'$ $\texttt{tc}$ a

(b) *if* $\texttt{hd}(B) \equiv a$ *then* $\Delta, \forall x{:}B \vdash \Delta'$ $\texttt{tc}$ a

2. *If* $\Delta_0, \Delta_1 \vdash A$ $\texttt{okD}$ $\overline{A}$ *and* $|\Delta_0| \vdash B : type$ *then:*

   (a) $\Delta_0, \exists x{:}B, \Delta_1 \vdash A$ $\texttt{okD}$ $\overline{A}$

   (b) *if* $\texttt{hd}(B) \equiv \texttt{hd}(A)$ *then* $\Delta_0, \forall x{:}B, \Delta_1 \vdash A$ $\texttt{okD}$ $\overline{A}$

3. *If* $\eta_0, \eta_1; \Phi \vdash \overline{A}$ $\texttt{okSGs}$ $a\,\overline{M}$ *and* $|\eta_0| \vdash B : type$ *then:*

   (a) $\eta_0, \exists^? x{:}B, \eta_1; \Phi \vdash \overline{A}$ $\texttt{okSGs}$ $a\,\overline{M}$

   (b) *if* $\texttt{hd}(B) \equiv \texttt{hd}(A)$ *then* $\eta_0, \forall x{:}B, \eta_1; \Phi \vdash \overline{A}$ $\texttt{okSGs}$ $a\,\overline{M}$

4. *If* $\eta_0, \eta_1; \Phi \vdash A$ $\texttt{okSG}$ $a\,\overline{M} \Longrightarrow \langle \eta'; F \rangle$ *and* $|\eta_0| \vdash B : type$ *then:*

   (a) $\eta' = \eta'_0, \eta'_1$ *and* $\eta_0, \exists^? x{:}B, \eta_1; \Phi \vdash A$ $\texttt{okSG}$ $a\,\overline{N} \Longrightarrow \langle \eta'_0, \forall x{:}B, \eta'_1; F \rangle$

   (b) *if* $\texttt{hd}(B) \equiv \texttt{hd}(A)$ *then* $\eta' = \eta'_0, \eta'_1$ *and* $\eta_0, \forall x{:}B, \eta_1; \Phi \vdash A$ $\texttt{okSG}$ $a\,\overline{N} \Longrightarrow$ $\langle \eta'_0, \forall x{:}B, \eta'_1; F \rangle$

5. *If* $\eta_0, \eta_1 \vdash A$ $\texttt{okD}$ $\overline{A}$ *and* $|\eta_0| \vdash B : type$ *then:*

   (a) $\eta_0, \exists^? x{:}B, \eta_1 \vdash A$ $\texttt{okD}$ $\overline{A}$

   (b) *if* $\texttt{hd}(B) \equiv \texttt{hd}(A)$ *then* $\eta_0, \forall x{:}B, \eta_1 \vdash A$ $\texttt{okD}$ $\overline{A}$

6. *If* $\eta_0, \eta_1 \vdash \eta'$ $\texttt{tc}$ a *and* $|\eta_0| \vdash B : type$ *then:*

   (a) $\eta_0, \exists^? x{:}B, \eta_1 \vdash \eta'$ $\texttt{tc}$ a

   (b) *if* $\texttt{hd}(B) \equiv a$ *then* $\eta_0, \forall x{:}B, \eta_1 \vdash \eta'$ $\texttt{tc}$ a

Proof: By simultaneous induction, using most of the weakening lemmas proven thus far and the definition of a sound sequent calculus. □

Lemma 5.2.59 (Termination Checking Mutually Recursive Types)

If $a \equiv b$ then:

1. *if* $\vdash \Sigma$ `tc` a *then* $\vdash \Sigma$ `tc` b
2. *if* $\Delta \vdash \Delta'$ `tc` a *then* $\Delta \vdash \Delta'$ `tc` a
3. *if* $\eta \vdash \eta'$ `tc` a *then* $\eta \vdash \eta'$ `tc` a

Proof: Each case is by straightforward induction on the given derivation. $\square$

We are almost ready to begin the proof of the soundness of the termination/mode checker. The statement of the theorem will along the lines of "for all termination checked queries $\Delta \vdash \boxed{?} : A$, either there exists θ, Δ', M and A' such that `fillTerm`$(\Delta \vdash \boxed{?} : A) \overset{\theta}{\Longrightarrow} \langle \Delta'; M; A' \rangle$ or no such θ, Δ', M and A' exist." Although this statement is nontrivial when interpreted intuitionistically (as has been our convention throughout this dissertation), interpreted classically it is a direct consequence of the law of excluded middle. However, we find it advantageous for the theorem to be phrased in such a way that it is nontrivial when viewed either way. To this end, we define the judgment `failTerm` (along with the helper-judgments `failHeads` and `failSpine`) that characterize when a query is known to fail.

$$\frac{\texttt{failHeads}_{\Sigma;\Delta}\langle \Delta \vdash \boxed{?} : a\,\overline{M}\rangle}{\texttt{failTerm}(\Delta \vdash \boxed{?} : a\,\overline{M})} \qquad \frac{\texttt{failTerm}(\Delta, \forall x{:}A \vdash \boxed{?} : B)}{\texttt{failTerm}(\Delta \vdash \boxed{?} : \Pi x{:}A.B)}$$

$$\frac{\texttt{failHeads}_{\Sigma';\Delta'}\langle \Delta \vdash \boxed{?} : a\,\overline{M}\rangle}{\texttt{failHeads}_{\Sigma';\Delta',\exists x:A}\langle \Delta \vdash \boxed{?} : a\,\overline{M}\rangle} \qquad \frac{\texttt{hd}(A) \neq a \quad \texttt{failHeads}_{\Sigma';\Delta'}\langle \Delta \vdash \boxed{?} : a\,\overline{N}\rangle}{\texttt{failHeads}_{\Sigma';\Delta',\forall x:A}\langle \Delta \vdash \boxed{?} : a\,\overline{N}\rangle}$$

$$\frac{\texttt{hd}(A) = a \quad \texttt{failSpine}(\Delta; A \vdash \overline{\boxed{?}} : a\,\overline{N}) \quad \texttt{failHeads}_{\Sigma';\Delta'}\langle \Delta \vdash \boxed{?} : a\,\overline{N}\rangle}{\texttt{failHeads}_{\Sigma';\Delta',\forall x:A}\langle \Delta \vdash \boxed{?} : a\,\overline{N}\rangle}$$

$$\frac{\begin{array}{c}\texttt{hd}(A) = a \quad \texttt{fillSpine}(\Delta; A \vdash \overline{\boxed{?}} : a\,\overline{N}) \overset{\theta}{\Longrightarrow} \langle \Delta''; A'; \overline{M}; a\,\overline{N'}\rangle \quad \theta(a\,\overline{N}) = a\,\overline{N''} \\ \texttt{unify}(\Delta'' \vdash \texttt{outputs}_a(\overline{N'}) \stackrel{\bullet}{=} \texttt{outputs}_a(\overline{N''})) \Longrightarrow \texttt{fail} \quad \texttt{failHeads}_{\Sigma';\Delta'}\langle \Delta \vdash \boxed{?} : a\,\overline{N}\rangle\end{array}}{\texttt{failHeads}_{\Sigma';\Delta',\forall x:A}\langle \Delta \vdash \boxed{?} : a\,\overline{N}\rangle}$$

$$\frac{\texttt{failHeads}_{\Sigma';\Delta'}\langle \Delta \vdash \boxed{?} : a\,\overline{M}\rangle}{\texttt{failHeads}_{\Sigma',a:K;\cdot}\langle \Delta \vdash \boxed{?} : a\,\overline{M}\rangle} \qquad \frac{\texttt{hd}(A) \neq a \quad \texttt{failHeads}_{\Sigma';\cdot}\langle \Delta \vdash \boxed{?} : a\,\overline{N}\rangle}{\texttt{failHeads}_{\Sigma',c:A;\cdot}\langle \Delta \vdash \boxed{?} : a\,\overline{N}\rangle}$$

$$\frac{\texttt{hd}(A) = a \quad \texttt{failSpine}(\Delta; A \vdash \overline{\boxed{?}} : a\,\overline{N}) \quad \texttt{failHeads}_{\Sigma';\cdot}\langle \Delta \vdash \boxed{?} : a\,\overline{N}\rangle}{\texttt{failHeads}_{\Sigma',c:A;\cdot}\langle \Delta \vdash \boxed{?} : a\,\overline{N}\rangle}$$

$$\frac{\begin{array}{c}\texttt{hd}(A) = a \quad \texttt{fillSpine}(\Delta; A \vdash \overline{\boxed{?}} : a\,\overline{N}) \overset{\theta}{\Longrightarrow} \langle \Delta''; A'; \overline{M}; a\,\overline{N'}\rangle \quad \theta(a\,\overline{N}) = a\,\overline{N''} \\ \texttt{unify}(\Delta'' \vdash \texttt{outputs}_a(\overline{N'}) \stackrel{\bullet}{=} \texttt{outputs}_a(\overline{N''})) \Longrightarrow \texttt{fail} \quad \texttt{failHeads}_{\Sigma';\cdot}\langle \Delta \vdash \boxed{?} : a\,\overline{N}\rangle\end{array}}{\texttt{failHeads}_{\Sigma',c:A;\cdot}\langle \Delta \vdash \boxed{?} : a\,\overline{N}\rangle}$$

$$\frac{}{\texttt{failHeads}_{\cdot;\cdot}\langle \Delta \vdash \boxed{?} : a\,\overline{N}\rangle} \qquad \frac{\texttt{unify}(\Delta \vdash \texttt{inputs}_a(\overline{M}) \stackrel{\bullet}{=} \texttt{inputs}_a(\overline{N})) \Longrightarrow \texttt{fail}}{\texttt{failSpine}(\Delta; a\,\overline{M} \vdash \overline{\boxed{?}} : a\,\overline{N})}$$

$$\frac{x \in FV(B) \quad \texttt{failSpine}(\Delta, \exists x{:}A; B \vdash \overline{\boxed{?}} : a\,\overline{N})}{\texttt{failSpine}(\Delta; \Pi x{:}A.B \vdash \overline{\boxed{?}} : a\,\overline{N})}$$

$$\frac{\texttt{fillSpine}(\Delta; B \vdash \overline{\boxed{?}} : a\,\overline{N}) \overset{\theta}{\Longrightarrow} \langle \Delta'; B; \overline{M}; a\,\overline{N'}\rangle \quad \theta A = A' \quad \texttt{failSpine}(\Delta'; A' \vdash \overline{\boxed{?}} : a\,\overline{N'})}{\texttt{failSpine}(\Delta; A \to B \vdash \overline{\boxed{?}} : a\,\overline{N})}$$

In the following lemma, note that $false$ is the judgment with no inference rules.

Lemma 5.2.60 (Soundness of Failure)

1. *If* $\texttt{failTerm}(\Delta \vdash \boxed{?} : A)$ *and* $\texttt{fillTerm}(\Delta \vdash \boxed{?} : A) \overset{\theta}{\Longrightarrow} \langle \Delta'; M; A'\rangle$ *then*

false.

2. *If* $\texttt{failHeads}_{\Delta_0;\Sigma_0}\langle \Delta_0, \Delta_1 \vdash \boxed{?} : a\,\overline{N}\rangle$ *and* $\Sigma = \Sigma_0, \Sigma_1$ *and* $\texttt{fillHead}(\Delta_0; \Sigma_0; a\,\overline{N}) \Longrightarrow h{:}A$ *and* $\texttt{fillTerm}(\Delta_0, \Delta_1 \vdash \boxed{?} : a\,\overline{N}) \overset{\theta}{\Longrightarrow} \langle \Delta';\, h\,\overline{M};\, a\,\overline{N'}\rangle$ *then false.*

3. *If* $\texttt{failSpine}(\Delta; A \vdash \overline{\boxed{?}} : a\,\overline{N})$ *and* $\texttt{fillSpine}(\Delta; A \vdash \overline{\boxed{?}} : a\,\overline{N}) \overset{\theta}{\Longrightarrow} \langle \Delta';\, A';\, \overline{M};\, a\,\overline{N'}\rangle$ *then false.*

Proof: By a straightforward mutual induction on the given derivation of failure. □

A query is terminating if it either succeeds (i.e. `fillTerm` is inhabited) or can be shown to exhaustively fail (i.e. `failTerm` is inhabited); this is formalized in the judgments below. Because our termination proof will rely on successful queries satisfying certain invariants, we include these invariants in the definition the judgments. The judgment $(\Delta \vdash \boxed{?} : A)$ `terminatesSatisfying` $\langle \eta; F\rangle$ means that the query $\Delta \vdash \boxed{?} : A$ terminates, and, moreover, if it terminates successfully, then F is well formed in the output context and η approximates the output substitution. The judgment $(\Delta; A \vdash \overline{\boxed{?}} : a\,\overline{M})\,;\overline{A}$ `terminates` means that the process of solving the spine-query $(\Delta; A \vdash \overline{\boxed{?}} : a\,\overline{M})$ along with the left-over subgoals in $\overline{A}$ terminates. The judgment $(\Delta \vdash \boxed{?} : a\,\overline{N})$ `terminatesUsing` $\langle \Delta'; \Sigma'\rangle$ means that the query $\Delta \vdash \boxed{?} : a\,\overline{N}$ terminates when potential heads are restricted to Δ' and Σ'.

$$\frac{\texttt{fillTerm}(\Delta \vdash \boxed{?} : A) \overset{\theta}{\Longrightarrow} \langle \Delta';\, M;\, A' \rangle \quad \Delta' \vdash A'\ \texttt{solved}}{(\Delta \vdash \boxed{?} : A)\,\texttt{terminates}} \qquad \frac{\texttt{failTerm}(\Delta \vdash \boxed{?} : A)}{(\Delta \vdash \boxed{?} : A)\,\texttt{terminates}}$$

$$\frac{\texttt{failHeads}_{\Delta';\Sigma'}\langle \Delta \vdash \boxed{?} : a\,\overline{N} \rangle}{(\Delta \vdash \boxed{?} : a\,\overline{N})\ \texttt{terminatesUsing}\ \langle \Delta'; \Sigma' \rangle}$$

$$\frac{\texttt{fillTerm}(\Delta \vdash \boxed{?} : a\,\overline{N}) \overset{\theta}{\Longrightarrow} \langle \Delta'';\, M;\, a\,\overline{N'} \rangle \quad \Delta'' \vdash a\,\overline{N'}\ \texttt{solved}}{(\Delta \vdash \boxed{?} : a\,\overline{N})\ \texttt{terminatesUsing}\ \langle \Delta'; \Sigma' \rangle}$$

$$\frac{\begin{array}{c}\texttt{fillTerm}(\Delta \vdash \boxed{?} : A) \overset{\theta}{\Longrightarrow} \langle \Delta';\, M;\, A' \rangle \quad \Delta' \vdash A'\ \texttt{solved} \quad \Delta' \vdash \theta : \eta \\ \Delta_\eta = \Delta \qquad \theta F = F' \quad |\Delta'| \vdash F'\ \texttt{wellTyped} \quad \Delta' \vdash F' \doteqdot \quad \Delta' \vdash F'\ \texttt{valid}\end{array}}{(\Delta \vdash \boxed{?} : A)\ \texttt{terminatesSatisfying}\ \langle \eta; F \rangle}$$

$$\frac{\texttt{failTerm}(\Delta \vdash \boxed{?} : A)}{(\Delta \vdash \boxed{?} : A)\ \texttt{terminatesSatisfying}\ \langle \eta; F \rangle}$$

$$\frac{\begin{array}{c}\texttt{fillSpine}(\Delta; A \vdash \overline{\boxed{?}} : a\,\overline{N}) \overset{\theta}{\Longrightarrow} \langle \Delta';\, A';\, \overline{M};\, a\,\overline{N'} \rangle \quad \theta\overline{A} = \overline{A'} \\ |\Delta| \vdash \Phi\ \texttt{wellTyped} \quad \Delta \vdash \Phi \doteqdot \quad \Delta \vdash \Phi\ \texttt{valid} \quad \Delta'; \Phi \vdash \overline{A'}\ \texttt{okSGs}\ a\,\overline{N'}\end{array}}{(\Delta; A \vdash \overline{\boxed{?}} : a\,\overline{N})\,;\overline{A}\ \texttt{terminates}}$$

$$\frac{\texttt{failSpine}(\Delta; A \vdash \overline{\boxed{?}} : a\,\overline{N})}{(\Delta; A \vdash \overline{\boxed{?}} : a\,\overline{N})\,;\overline{A}\ \texttt{terminates}}$$

We are finally ready to begin our proof of termination. Most of the reasoning will be by induction on the structure of various derivations, as usual. However, we will also induct on termination vectors $V = \texttt{size}_\Delta(A)$ under the ordering $<_v$ in instances where the judgment $\Delta \vdash A\ \texttt{okG}$ is known to be inhabited. Below, we show that V will always be well defined in such situations.

Lemma 5.2.61 (Size Metric on Ok Goals)

1. *If* $\Delta \vdash A\ \texttt{okG}$ *then there exists a* V *such that* $\texttt{size}_\Delta(A) = V$ *and* $V =_v V$.

2. *If* $\texttt{size}_\Delta(A) = V$ *and* $\texttt{size}_\Delta(A) = V'$ *then* $V = V'$

3. *If* $\Delta \vdash A$ `okG` *and* $\texttt{size}_{\Delta}(A) = V$ *then* $V =_v V$.

Proof: Note the distinction between syntactic equality, $=$, and the judgmentally defined notion of equality $=_v$. 1 is by a straightforward induction on the given derivation, using Lemma 5.2.45. 2 is by straightforward induction on A. 3 is a direct corollary to 1 and 2. □

We are now ready to prove the soundness of the mode/termination checker. If this proof were to be realized by a function, the function could be rightly viewed as an interpretor for logic programs.

Theorem 5.2.62 (Soundness of The Mode/Termination Checker)

1. *If* $\mathcal{D} : (\Delta \vdash A\ \texttt{okG})$ *and* $\texttt{size}_{\Delta}(A) = V$ *and* $V =_v V$ *then* $(\Delta \vdash \boxed{?} : A)$ `terminates`

2. *If* $\mathcal{D} : (\Delta_0, \Delta_1 \vdash \Delta_0\ \texttt{tc}\ a)$ *and* $\mathcal{E} : (\vdash \Sigma_0\ \texttt{tc}\ a)$ *and* $\Sigma = \Sigma_0, \Sigma_1$ *and* $\mathcal{F} : (\Delta_0, \Delta_1 \vdash a\,\overline{N}\ \texttt{okG})$ *and* $V = \texttt{size}_{\Delta_0,\Delta_1}(a\,\overline{N})$ *and* $V =_v V$ *then* $(\Delta_0, \Delta_1 \vdash \boxed{?} : a\,\overline{N})$ `terminatesUsing` $\langle \Delta_0; \Sigma_0 \rangle$

3. *If* $\mathcal{D} : (\Delta \vdash A\ \texttt{okD}\ \overline{A})$ *and* $|\Delta| \vdash A : type$ *and* $|\Delta| \vdash \overline{A}$ `wellTyped` *and* $\mathcal{E} : (\Delta \vdash a\,\overline{N}\ \texttt{okG})$ *and* $\texttt{hd}(A) = a$ *and* $\texttt{size}_{\Delta}(a\,\overline{N}) = V$ *and* $V =_v V$ *then* $(\Delta; A \vdash \overline{\boxed{?}} : a\,\overline{N})\,;\overline{A}$ `terminates`

4. *If* $\mathcal{D} : (\Delta; \Phi \vdash A\ \texttt{okSG}\ a\,\overline{M} \Longrightarrow \langle \eta; F \rangle)$ *and* $|\Delta| \vdash A : type$ *and* $|\Delta| \vdash \Phi$ `wellTyped` *and* $\Delta \vdash \Phi \doteq$ *and* $\Delta \vdash \Phi$ `valid` *and* $\mathcal{E} : \Delta \vdash a\,\overline{M}\ \texttt{okG}$ *and* $\texttt{size}_{\Delta}(a\,\overline{N}) = V$ *and* $V =_v V$ *then* $(\Delta \vdash \boxed{?} : A)$ `terminatesSatisfying` $\langle \eta; F \rangle$

Proof: By mutual induction on a lexicographic ordering built up from three underlying orderings: termination vector ordered by $<_v$, a natural number no greater than

$s\,s\,s\,z$ ordered structurally[3], and a simultaneous ordering of two terms or derivations, each of which is ordered structurally. 1 is ordered by V, followed by $s\,s\,s\,z$, followed by the simultaneous ordering of A and A. 2 is ordered by V, followed by $s\,s\,z$, followed by the simultaneous ordering of $\mathcal{D}$ and $\mathcal{E}$. 3 is ordered by V, followed by $s\,z$, followed by the simultaneous ordering of $\mathcal{D}$ and $\mathcal{D}$. 4 is ordered by V, followed by z followed by the simultaneous ordering of $\mathcal{D}$ and $\mathcal{D}$. We show all of the cases for 1 and 3, and some representative cases of 4. 2 is a tedious exercise in case analysis, but is otherwise mostly straightforward; it uses IH 3, Lemma 5.2.7, Definition 5.2.47, Proposition 5.2.12, Lemma 5.2.48, Lemma 5.2.20 and Lemma 5.2.53.

1.

Case:

$$\mathcal{D} = \dfrac{\overset{\mathcal{D}_0}{|\Delta| \vdash a\,\overline{M} : \mathit{type}} \qquad \overset{\mathcal{D}_1}{\vdash \Sigma\ \texttt{tc}\ a} \qquad \overset{\mathcal{D}_2}{\Delta \vdash \Delta\ \texttt{tc}\ a} \qquad \overset{\mathcal{D}_3}{\Delta \vdash \texttt{inputs}_a(\overline{M}) \doteq}}{\Delta \vdash a\,\overline{M}\ \texttt{okG}}$$

$\texttt{size}_\Delta(a\,\overline{M}) = \mathrm{V}$ and $V =_v V$ given

$(\Delta \vdash \boxed{?} : a\,\overline{M})\ \texttt{terminatesUsing}\ \langle\cdot;\cdot\rangle$ by IH 2 on V, $s\,s\,z$, $\mathcal{D}_2$ and $\mathcal{D}_1$

either $\texttt{failHeads}_{\cdot;\cdot}\langle\Delta \vdash \boxed{?} : a\,\overline{M}\rangle$

or $(\texttt{fillTerm}(\Delta \vdash \boxed{?} : a\,\overline{M}) \overset{\theta}{\Longrightarrow} \langle\Delta''; M; a\,\overline{N'}\rangle$ and $\Delta'' \vdash a\,\overline{M'}\ \texttt{solved})$

by inversion on $\texttt{terminatesUsing}$

assume $\texttt{failHeads}_{\cdot;\cdot}\langle\Delta \vdash \boxed{?} : a\,\overline{M}\rangle$ Case 1

$(\Delta \vdash \boxed{?} : a\,\overline{M})\,\texttt{terminates}$ by rule

assume $\texttt{fillTerm}(\Delta \vdash \boxed{?} : a\,\overline{M}) \overset{\theta}{\Longrightarrow} \langle\Delta''; M; a\,\overline{N'}\rangle$ and $\Delta'' \vdash a\,\overline{M'}\ \texttt{solved}$

Case 2

$(\Delta \vdash \boxed{?} : a\,\overline{M})\,\texttt{terminates}$ by rule

[3]This is equivalent to using the finite ordinal 4.

Case:

$$\mathcal{D} = \dfrac{\overset{\mathcal{D}_0}{|\Delta| \vdash A : \mathit{type}} \qquad \overset{\mathcal{D}_1}{\Delta, \forall x{:}A \vdash B\ \texttt{okG}}}{\Delta \vdash \Pi x{:}A.B\ \texttt{okG}}$$

$\texttt{size}_\Delta(\Pi x{:}A.B) = V$ and $V =_v V$ given

$V = \texttt{size}_{\Delta,\forall x:a}(B)$ by inversion on $\texttt{size}$

$\texttt{size}_{\Delta,\forall x:A}(B) <_v \texttt{size}_\Delta(\Pi x{:}A.B)$ by rule

$(\Delta, \forall x{:}A \vdash \boxed{?} : B)\,\texttt{terminates}$ by IH 1 on V, $s\,s\,s\,z$, B and B

either $\texttt{failTerm}(\Delta, \forall x{:}A \vdash \boxed{?} : B)$

or $(\texttt{fillTerm}(\Delta, \forall x{:}A \vdash \boxed{?} : B) \overset{\theta}{\Longrightarrow} \langle \Delta';\, M;\, B' \rangle$ and $\Delta \vdash \Delta'$ $\texttt{solved}$B')

by inversion on $\texttt{terminates}$

assume $\texttt{failTerm}(\Delta, \forall x{:}A \vdash \boxed{?} : B)$ Case 1

$\texttt{failTerm}(\Delta \vdash \boxed{?} : \Pi x{:}A.B)$ by rule

$(\Delta \vdash \boxed{?} : \Pi x{:}A.B)\,\texttt{terminates}$ by rule

assume $\texttt{fillTerm}(\Delta, \forall x{:}A \vdash \boxed{?} : B) \overset{\theta}{\Longrightarrow} \langle \Delta';\, M;\, B' \rangle$

and $\Delta' \vdash B'\ \texttt{solved}$ Case 2

$|\Delta|, x{:}A \vdash B : \mathit{type}$ by Lemma 5.2.56

$\Delta' \vdash \theta : \Delta, \forall x{:}A$ and $\vdash \Delta'\ \texttt{raised}$

and $\theta B = B'$ and $|\Delta'| \vdash M : B'$ by Lemma 5.2.14

$\theta = \theta', y/x$ and $\Delta' = \Delta'', \forall y{:}A', \exists\Gamma$

and $\Delta'' \vdash \theta' : \Delta$ and $\theta' A = A'$ by Lemma 5.2.1

$\Delta' = \Delta'', \forall y{:}A'$ by inversion on $\texttt{raised}$

$\texttt{fillTerm}(\Delta \vdash \boxed{?} : \Pi x{:}A.B) \overset{\theta'}{\Longrightarrow} \langle \Delta'';\, \lambda y{:}A'.M;\, \Pi y{:}A'.B' \rangle$ by rule

$\Delta'' \vdash \Pi y{:}A'.B'\ \texttt{solved}$ by rule

$(\Delta \vdash \boxed{?} : \Pi x{:}A.B)\,\texttt{terminates}$ by rule

3.

Case:

$$\mathcal{D} = \cfrac{\overset{\mathcal{D}_0}{\Delta \vdash B \ \texttt{okD} \ A :: \overline{A}}}{\Delta \vdash A \to B \ \texttt{okD} \ \overline{A}}$$

$|\Delta| \vdash A \to B : \mathit{type}$ and $|\Delta| \vdash \overline{A}$ $\texttt{wellTyped}$ and $\Delta \vdash a\,\overline{N}$ $\texttt{okG}$

and $\texttt{hd}(A \to B) = a$ and $\texttt{size}_\Delta(a\,\overline{N}) = V$ and $V =_v V$ — given

$|\Delta| \vdash A : \mathit{type}$

and $|\Delta|, x{:}A \vdash B : \mathit{type}$ where $x \# B$ — by inversion for typing

$|\Delta| \vdash B : \mathit{type}$ — by Lemma 5.1.7

$|\Delta| \vdash A :: \overline{A}$ $\texttt{wellTyped}$ — by rule

$\texttt{hd}(A \to B) = \texttt{hd}(B)$ — by inversion on $\texttt{hd}$

$(\Delta; B \vdash \boxed{?} : a\,\overline{N})\,;(A :: \overline{A})$ $\texttt{terminates}$ — by IH 2 on V, $s\,s\,z$, $\mathcal{D}_0$ and $\mathcal{D}_0$

either $\texttt{failSpine}(\Delta; B \vdash \boxed{?} : a\,\overline{N})A :: \overline{A}$

or $(\texttt{fillSpine}(\Delta; B \vdash \boxed{?} : a\,\overline{N}) \overset{\theta}{\Longrightarrow} \langle \Delta';\ B';\ \overline{M};\ a\,\overline{N'} \rangle$

  and $\theta(A :: \overline{A}) = A' :: \overline{A'}$ and $|\Delta'| \vdash \Phi$ $\texttt{wellTyped}$ and $\Delta' \vdash \Phi \,\underline{\underline{\vdash}}$

  and $\Delta' \vdash \Phi$ $\texttt{valid}$ and $\Delta'; \Phi \vdash A' :: \overline{A'}$ $\texttt{okSGs}$ $a\,\overline{N'})$

by inversion on $\texttt{terminates}$

assume $\texttt{failSpine}(\Delta; B \vdash \boxed{?} : a\,\overline{N})$ — Case 1

$\texttt{failSpine}(\Delta; A \to B \vdash \boxed{?} : a\,\overline{N})$ — by rule

$(\Delta; A \to B \vdash \boxed{?} : a\,\overline{N})\,; \overline{A}$ $\texttt{terminates}$ — by rule

assume $\texttt{fillSpine}(\Delta; B \vdash \boxed{?} : a\,\overline{N}) \overset{\theta}{\Longrightarrow} \langle \Delta';\ B';\ \overline{M};\ a\,\overline{N'} \rangle$

and $\theta(A :: \overline{A}) = A'' :: \overline{A'}$ and $|\Delta'| \vdash \Phi$ $\texttt{wellTyped}$ and $\Delta' \vdash \Phi \,\underline{\underline{\vdash}}$

and $\Delta' \vdash \Phi$ $\texttt{valid}$ and $\Delta'; \Phi \vdash A' :: \overline{A'}$ $\texttt{okSGs}$ $a\,\overline{N'}$ — Case 2

$|\Delta| \vdash a\,\overline{N} : \mathit{type}$ and $\vdash \Sigma$ $\texttt{tc}$ a

and $\Delta \vdash \Delta$ `tc` a and $\Delta \vdash \mathtt{inputs}_a(\overline{N}) \mathrel{\underline{\overline{\dot{}}}}$ by inversion on `okG`

$\Delta' \vdash \theta : \Delta$ and $\vdash \Delta'$ `raised`

and $\theta B = B'$ and $\theta(\mathtt{inputs}_a(\overline{N})) = \mathtt{inputs}_a(\overline{N'})$

and $|\Delta|;\ B' \vdash \overline{M} : a\,\overline{N'}$ by Lemma 5.2.14

$\theta A = A'$ and $\theta\overline{A} = \overline{A'}$ by inversion on gsub app

$|\Delta'| \vdash A' : \mathit{type}$ and $\Delta'; \Phi \vdash A'$ `okSG` $a\,\overline{N'} \Longrightarrow \langle\eta; F\rangle$

and $\mathcal{C} : \eta; \Phi, F \vdash \overline{A'}$ `okSGs` $a\,\overline{N'}$ by inversion on `okSGs`

$|\Delta'| \vdash a\,\overline{N'} : \mathit{type}$ by Corollary 5.1.13

$\Delta' \vdash \Delta'$ `tc` a by Lemma 5.2.57

$\Delta' \vdash (\mathtt{inputs}_a(\overline{N'})) \mathrel{\underline{\overline{\dot{}}}}$ by Lemma 5.2.52

$\Delta' \vdash a\,\overline{N'}$ `okG` by rule

$\mathtt{size}_{\Delta'}(a\,\overline{N'}) = V'$ and $V' =_v V'$ by Lemma 5.2.61

$\langle\Delta; \mathtt{inputs}_a(\overline{N})\rangle =_o \langle\Delta'; \mathtt{inputs}_a(\overline{N'})\rangle$ by Lemma 5.2.45

$V =_v V'$ by def of `size` and $=_v$

$(\Delta' \vdash \boxed{?} : A')$ `terminatesSatisfying` $\langle\eta; F\rangle$ by IH 4 on V', z, $\mathcal{C}$ and $\mathcal{C}$

either $\mathtt{failTerm}(\Delta' \vdash \boxed{?} : A')$

or $(\mathtt{fillTerm}(\Delta' \vdash \boxed{?} : A') \overset{\theta'}{\Longrightarrow} \langle\Delta'';\ M';\ A''\rangle$

and $\Delta'' \vdash A''$ `solved` and $\Delta'' \vdash \theta' : \eta$ and $\Delta_\eta = \Delta'$

and $\theta' F = F'$ and $|\Delta'| \vdash F'$ `wellTyped`

and $\Delta' \vdash F' \mathrel{\underline{\overline{\dot{}}}}$ and $\Delta' \vdash F'$ `valid`) by inversion on `terminatesSatisfying`

assume $\mathtt{failTerm}(\Delta' \vdash \boxed{?} : A')$ Case 2.1

$\mathtt{failSpine}(\Delta; A \to B \vdash \overline{\boxed{?}} : a\,\overline{N})$ by rule

$(\Delta; A \to B \vdash \overline{\boxed{?}} : a\,\overline{N})\,; \overline{A}$ `terminates` by rule

assume $\mathtt{fillTerm}(\Delta' \vdash \boxed{?} : A') \overset{\theta'}{\Longrightarrow} \langle\Delta'';\ M';\ A''\rangle$

and $\Delta'' \vdash A''$ `solved` and $\Delta'' \vdash \theta' : \eta$ and $\Delta_\eta = \Delta'$

and $\theta' F = F'$ and $|\Delta'| \vdash F'$ `wellTyped`

and $\Delta' \vdash F' \doteqdot$ and $\Delta' \vdash F'$ `valid` Case 2.2

$\Delta'' \vdash \theta' : \Delta'$ and $\vdash \Delta''$ `raised`

and $\theta' A' = A''$ and $|\Delta''| \vdash M' : A''$ by Lemma 5.2.14

$\theta(a\,\overline{N'}) = a\,\overline{N''}$ and $|\Delta''| \vdash a\,\overline{N''} : \mathit{type}$ by Lemma 5.2.7

$|\Delta'| \vdash B' : \mathit{type}$ by Lemmas 5.2.7 and 5.2.2

$\theta' B' = B''$ and $|\Delta''| \vdash B'' : \mathit{type}$ by Lemma 5.2.7

$\theta\overline{M} = \overline{M'}$ and $|\Delta''|;\, B'' \vdash \overline{M'} : a\,\overline{N''}$ by Lemma 5.2.7

$\theta' \circ \theta = \theta''$ by Lemma 5.2.9

$\mathtt{fillSpine}(\Delta; A \rightarrow B \vdash \boxed{?} : a\,\overline{N}) \overset{\theta''}{\Longrightarrow} \langle \Delta'';\, A'' \rightarrow B'';\, M' :: \overline{M'};\, a\,\overline{N''} \rangle$ by rule

$|\Delta'| \vdash \overline{A'}$ `wellTyped` by Lemma 5.2.55 (twice)

$\theta'\overline{A'} = \overline{A''}$ and $|\Delta''| \vdash \overline{A''}$ `wellTyped` by Lemma 5.2.55

$\theta''\overline{A} = \overline{A''}$ by Lemma 5.2.55

$\theta'\Phi = \Phi'$ and $|\Delta''| \vdash \Phi'$ `wellTyped` by Lemma 5.2.49

$\Delta'' \vdash \Phi' \doteqdot$ by Lemma 5.2.52

$\Delta'' \vdash \Phi'$ `valid` by Lemma 5.2.53

$\Delta''; \Phi' \vdash \overline{A''}$ `okSGs` $a\,\overline{N''}$ by Lemma 5.2.57

$(\Delta; A \rightarrow B \vdash \boxed{?} : a\,\overline{N})\,; \overline{A}$ `terminates` by rule

Case:

$$\mathcal{D} = \dfrac{\overset{\mathcal{D}_0}{|\Delta| \vdash A : \mathit{type}} \qquad \overset{\mathcal{D}_1}{\Delta, \exists x{:}A \vdash B\ \mathtt{okD}\ \overline{A}} \qquad x \in FV(B)}{\Delta \vdash \Pi x{:}A.B\ \mathtt{okD}\ \overline{A}}$$

$|\Delta| \vdash \Pi x{:}A.B : \mathit{type}$ and $|\Delta| \vdash \overline{A}$ `wellTyped`

and $\Delta \vdash a\,\overline{N}$ `okG` and $\mathtt{hd}(\Pi x{:}A.B) = a$

and $\mathtt{size}_\Delta(a\,\overline{N}) = V$ and $V =_v V$ given

$|\Delta| \vdash A : \mathit{type}$ and $|\Delta, \exists\mathrm{x{:}A}| \vdash B : \mathit{type}$ by inversion for typing

$|\Delta|, x{:}A \vdash \overline{A}$ `wellTyped` by Lemma 5.2.55

$|\Delta| \vdash a\,\overline{N} : \mathit{type}$ and $\vdash \Sigma$ `tc` a

and $\Delta \vdash \Delta$ `tc` a and $\Delta \vdash \mathtt{inputs}_a(\overline{N}) \doteq$ by inversion on `okG`

$|\Delta|, x{:}A \vdash a\,\overline{N} : \mathit{type}$ by Lemma 5.1.6

$\Delta, \exists x{:}A \vdash \Delta$ `tc` a by Lemma 5.2.58

$\Delta, \exists x{:}A \vdash \Delta, \exists x{:}A$ `tc` a by rule

$\Delta, \exists x{:}A \vdash \mathtt{inputs}_a(\overline{N}) \doteq$ by Lemma 5.2.51

$\Delta, \exists x{:}A \vdash a\,\overline{N}$ `okG` by rule

$\mathtt{hd}(\Pi x{:}A.B) = \mathtt{hd}(B)$ by inversion on `hd`

$\langle \Delta; \mathtt{inputs}_a(\overline{N})\rangle =_o \langle \Delta, \exists x{:}A; \mathtt{inputs}_a(\overline{N})\rangle$ by Lemma 5.2.45 (twice)

$\mathtt{size}_{\Delta,\exists x:A}(a\,\overline{N}) = V'$ and $V' =_v V'$ by Lemma 5.2.61

$V =_v V'$ by def of `size` and $=_v$

$(\Delta, \exists x{:}A; B \vdash \overline{\boxed{?}} : a\,\overline{N})\,;\overline{A}$ `terminates` by IH 3 on V', $s\,z$, $\mathcal{D}_1$ and $\mathcal{D}_1$

either $\mathtt{failSpine}(\Delta, \exists x{:}A; B \vdash \overline{\boxed{?}} : a\,\overline{N})$

or $(\mathtt{fillSpine}(\Delta, \exists x{:}A; B \vdash \overline{\boxed{?}} : a\,\overline{N}) \overset{\theta}{\Longrightarrow} \langle \Delta';\ B';\ \overline{M};\ a\,\overline{N'}\rangle$

and $\theta\overline{A} = \overline{A'}$ and $|\Delta'| \vdash \Phi$ `wellTyped` and $\Delta' \vdash \Phi \doteq$

and $\Delta' \vdash \Phi$ `valid` and $\Delta'; \Phi \vdash \overline{A'}$ `okSGs` $a\,\overline{N'})$

by inversion on `terminates`

assume $\mathtt{failSpine}(\Delta, \exists x{:}A; B \vdash \overline{\boxed{?}} : a\,\overline{N})$ Case 1

$\mathtt{failSpine}(\Delta; \Pi x{:}A.B \vdash \overline{\boxed{?}} : a\,\overline{N})$ by rule

$(\Delta; \Pi x{:}A.B \vdash \overline{\boxed{?}} : a\,\overline{N})\,;\overline{A}$ `terminates` by rule

assume $\mathtt{fillSpine}(\Delta, \exists x{:}A; B \vdash \overline{\boxed{?}} : a\,\overline{N}) \overset{\theta}{\Longrightarrow} \langle \Delta';\ B';\ \overline{M};\ a\,\overline{N'}\rangle$

and $\theta\overline{A} = \overline{A'}$ and $|\Delta'| \vdash \Phi$ `wellTyped` and $\Delta' \vdash \Phi \doteq$

and $\Delta' \vdash \Phi$ `valid` and $\Delta'; \Phi \vdash \overline{A'}$ `okSGs` $a\,\overline{N'})$ Case 2

$\Delta' \vdash \theta : \Delta, \exists x{:}A$ and $\vdash \Delta'$ `raised` and $\theta B = B'$

and $\theta(\texttt{inputs}_a(\overline{N})) = \texttt{inputs}_a(\overline{N'})$

and $|\Delta|;\ B' \vdash \overline{M} : a\,\overline{N'}$ by Lemma 5.2.14

$\theta = \theta', M{:}A'/x$ and $\Delta' \vdash \theta' : \Delta$

and $\theta' A = A'$ and $|\Delta'| \vdash M : A'$ by Lemma 5.2.1

$\Delta', \forall y{:}A' \vdash \theta', y/x : \Delta', \forall x{:}A$ by rule

$(\theta, y/x)B = B''$ and $|\Delta'|, x{:}A' \vdash B'' : \mathit{type}$ by Lemma 5.2.7

$\texttt{fillSpine}(\Delta; \Pi x{:}A.B \vdash \boxed{?} : a\,\overline{N}) \xRightarrow{\theta'} \langle \Delta';\ \Pi y{:}A'.B'';\ M :: \overline{M};\ a\,\overline{N'}\rangle$ by rule

$x \# \overline{A}$ by the renamability of bound variables

$\theta' \overline{A} = \overline{A'}$ by Lemma 5.2.55

$(\Delta; \Pi x{:}A.B \vdash \boxed{?} : a\,\overline{N})\,;\overline{A}\ \texttt{terminates}$ by rule

Case:

$$\mathcal{D} = \cfrac{\begin{matrix}\mathcal{D}_0 & \mathcal{D}_1 & \mathcal{D}_2 \\ \Delta \vdash \texttt{inputs}_a(\overline{M}) \doteq\!\!\Longrightarrow \eta & \eta \vdash \texttt{inputs}_a(\overline{M}) \doteq & \eta; \cdot \vdash \overline{A}\ \texttt{okSGs}\ a\,\overline{M}\end{matrix}}{\Delta \vdash a\,\overline{M}\ \texttt{okD}\ \overline{A}}$$

$|\Delta| \vdash a\,\overline{M} : \mathit{type}$ and $|\Delta| \vdash \overline{A}\ \texttt{wellTyped}$ and

$\Delta \vdash a\,\overline{N}\ \texttt{okG}$ and $\texttt{size}_\Delta(a\,\overline{N} = V)$ and $V =_v V$ given

$|\Delta| \vdash a\,\overline{M} : \mathit{type}$ and $\vdash \Sigma\ \texttt{tc}\ a$

and $\Delta \vdash \Delta\ \texttt{tc}\ a$ and $\Delta \vdash \texttt{inputs}_a(\overline{N}) \doteq$ by inversion on $\texttt{okG}$

either $\texttt{unify}(\Delta \vdash \texttt{inputs}_a(\overline{M}) \overset{\bullet}{=} \texttt{inputs}_a(\overline{N})) \Longrightarrow \texttt{fail}$

or $\texttt{unify}(\Delta \vdash \texttt{inputs}_a(\overline{M}) \overset{\bullet}{=} \texttt{inputs}_a(\overline{N})) \xRightarrow{\Delta'} \theta$ by Proposition 5.2.12

assume $\texttt{unify}(\Delta \vdash \texttt{inputs}_a(\overline{N}) \overset{\bullet}{=} \texttt{inputs}_a(\overline{M})) \Longrightarrow \texttt{fail}$ Case 1

$\texttt{failSpine}(\Delta; a\,\overline{M} \vdash \boxed{?} : a\,\overline{N})$ by rule

$(\Delta; a\,\overline{M} \vdash \boxed{?} : a\,\overline{N})\,;\overline{A}\ \texttt{terminates}$ by rule

assume $\texttt{unify}(\Delta \vdash \texttt{inputs}_a(\overline{M}) \overset{\bullet}{=} \texttt{inputs}_a(\overline{N})) \xRightarrow{\Delta'} \theta$ Case 2

$\Delta' \vdash \theta : \Delta$ and $\vdash \Delta'\ \texttt{raised}$

and θ unifies $\texttt{inputs}_a(\overline{M})$ and $\texttt{inputs}_a(\overline{N})$ by Proposition 5.2.12

$\theta(a\,\overline{M}) = a\,\overline{M'}$ and $|\Delta| \vdash a\,\overline{M'} : \mathit{type}$ by Lemma 5.2.7

$\theta\overline{M} = \overline{M'}$ by inversion on gsub app

$\texttt{fillSpine}(\Delta; a\,\overline{M} \vdash \boxed{?} : a\,\overline{N}) \overset{\theta}{\Longrightarrow} \langle \Delta';\ a\,\overline{M'};\ (\cdot);\ a\,\overline{M'} \rangle$ by rule

$\theta\overline{A} = \overline{A'}$ and $|\Delta'| \vdash \overline{A'}$ $\texttt{wellTyped}$ by Lemma 5.2.55

$\theta(a\,\overline{N}) = a\,\overline{N'}$ and $|\Delta| \vdash a\,\overline{N'} : \mathit{type}$ by Lemma 5.2.7

$\theta\overline{N} = \overline{N'}$ by inversion on gsub app

$\texttt{inputs}_a(\overline{M'}) = \texttt{inputs}_a(\overline{N'})$ by def of unifier

$\Delta' \vdash \texttt{inputs}_a(\overline{N'}) \doteq$ by Lemma 5.2.20

$\Delta' \vdash \texttt{inputs}_a(\overline{M'}) \doteq$ by equality

$\Delta' \vdash \theta : \eta$ by Lemma 5.2.36

$\theta\cdot = \cdot$ by rule

$\Delta; \cdot \vdash \overline{A'}$ $\texttt{okSGs}$ $a\,\overline{M'}$

$|\Delta'| \vdash \cdot$ $\texttt{wellTyped}$ by rule

$\Delta' \vdash \cdot$ $\texttt{valid}$ by rule

$\Delta' \vdash \cdot \doteq$ by rule

$(\Delta; a\,\overline{M} \vdash \boxed{?} : a\,\overline{N})\,; \overline{A}$ $\texttt{terminates}$ by rule

4.

Case:

$$\mathcal{D} = \dfrac{\mathcal{D}_0 : (\texttt{hd}(A) \equiv a) \quad \mathcal{D}_1 : (|\Delta| \vdash A : \mathit{type}) \quad \mathcal{D}_2 : (\Delta, \forall x{:}A \vdash A \ \texttt{okD}\ \cdot) \qquad \mathcal{D}_3 : (\Delta, \forall x{:}A; \Phi \vdash B \ \texttt{okSG}\ a\,\overline{M} \Longrightarrow \langle \eta, \forall x{:}A; F \rangle)}{\Delta; \Phi \vdash \Pi x{:}A.B \ \texttt{okSG}\ a\,\overline{M} \Longrightarrow \langle \eta; \nabla x{:}A.F \rangle}$$

$|\Delta| \vdash \Pi x{:}A.B : \mathit{type}$ and $|\Delta| \vdash \Phi$ $\texttt{wellTyped}$

and $\Delta \vdash \Phi \doteq$ and $\Delta \vdash \Phi$ $\texttt{valid}$

and $\Delta \vdash a\,\overline{M}$ `okG` and $\texttt{size}_{\Delta}(a\,\overline{M}) = V$ and $V =_v V$ given

$|\Delta| \vdash A : \mathit{type}$ and $|\Delta|, x{:}A \vdash B : \mathit{type}$ by inversion for typing

$|\Delta, \forall x{:}\mathrm{A}| \vdash \Phi$ `wellTyped` by Lemma 5.2.50

$\Delta, \forall x{:}A \vdash \Phi \,\underline{\perp}$ by Lemma 5.2.51

$\Delta, \forall x{:}A \vdash \Phi \,\underline{\perp}$ by Lemma 5.2.54

$|\Delta| \vdash a\,\overline{M} : \mathit{type}$ and $\vdash \Sigma$ `tc` a

and $\Delta \vdash \Delta$ `tc` a and $\Delta \vdash \texttt{inputs}_a(\overline{M}) \,\underline{\perp}$ by inversion on `okG`

$|\Delta|, x{:}A \vdash a\,\overline{M} : \mathit{type}$ by Lemma 5.1.6

$\Delta, \forall x{:}A \vdash \Delta$ `tc` a by Lemma 5.2.58

$\Delta, \forall x{:}A \vdash \Delta, \forall x{:}A$ `tc` a by rule

$\Delta, \forall x{:}A \vdash \texttt{inputs}_a(\overline{M}) \,\underline{\perp}$ by Lemma 5.2.18

$\Delta, \forall x{:}A \vdash a\,\overline{M}$ `okG` by rule

$|\Delta| \vdash \texttt{measure}_a\ \overline{M}$ `wellTyped` by Lemma 5.2.44

$\Delta \vdash \texttt{measure}_a\ \overline{M} \,\underline{\perp}$ by Lemma 5.2.44

$\langle \texttt{measure}_a\ \overline{M}; \Delta\rangle =_o \langle \texttt{measure}_a\ \overline{M}; \Delta\rangle$ by Lemma 5.2.45

$\langle \texttt{measure}_a\ \overline{M}; \Delta\rangle =_o \langle \texttt{measure}_a\ \overline{M}; \Delta, \forall x{:}A\rangle$ by Lemma 5.2.45

$\texttt{size}_{\Delta,\forall x:A}(a\,\overline{M}) = V'$ and $V' =_v V'$ by Lemma 5.2.61

$V =_v V'$ by def of `size` and def of $=_v$

$(\Delta, \forall x{:}A \vdash \boxed{?} : B)$ `terminatesSatisfying` $\langle \eta, \forall x{:}A; F\rangle$

by IH 4 on V', $z\ \mathcal{D}_3$ and $\mathcal{D}_3$

either $\texttt{failTerm}(\Delta, \forall x{:}A \vdash \boxed{?} : B)$

or $(\texttt{fillTerm}(\Delta, \forall x{:}A \vdash \boxed{?} : B) \overset{\Delta'}{\Longrightarrow} \langle \theta;\ M;\ B'\rangle$

and $\Delta' \vdash B'$ `solved` and $\Delta' \vdash \theta : \eta, \forall x{:}A$ and $\Delta_\eta = \Delta, \forall x{:}A$

and $\theta F = F'$ and $|\Delta'| \vdash F$ `wellTyped` and $\Delta' \vdash F \,\underline{\perp}$ and $\Delta' \vdash F$ `valid`)

by inversion on `terminatesSatisfying`

assume $\texttt{failTerm}(\Delta, \forall x{:}A \vdash \boxed{?} : B)$ Case 1

$\texttt{failTerm}(\Delta \vdash \boxed{?} : \Pi x{:}A.B)$ by rule

$(\Delta \vdash \boxed{?} : \Pi x{:}A.B)$ $\texttt{terminatesSatisfying}$ $\langle \eta; F\rangle$ by rule

assume $\texttt{fillTerm}(\Delta, \forall x{:}A \vdash \boxed{?} : B) \overset{\Delta'}{\Longrightarrow} \langle \theta;\ M;\ B'\rangle$

and $\Delta' \vdash B'$ $\texttt{solved}$ and $\Delta' \vdash \theta : \eta, \forall x{:}A$ and $\Delta_\eta = \Delta, \forall x{:}A$

and $\theta F = F'$ and $|\Delta'| \vdash F$ $\texttt{wellTyped}$

and $\Delta' \vdash F \doteq$ and $\Delta' \vdash F$ $\texttt{valid}$ Case 2

$\Delta' \vdash \theta : \Delta, \forall x{:}A$ and $\vdash \Delta'$ $\texttt{raised}$ and $\theta B = B'$

and $\theta(\texttt{inputs}_a(\overline{M}) = \texttt{inputs}_a(\overline{M'}))$ and $|\Delta'| \vdash M : B'$ by Lemma 5.2.14

$\Delta' = \Delta'', \forall y{:}A', \exists\Gamma$ and $\theta = \theta', y/x$ and $\theta A = A'$

and $\Delta'' \vdash \theta' : \Delta$ by Lemma 5.2.1

$\Delta' = \Delta'', \forall y{:}A'$ by inversion on $\texttt{raised}$

$\texttt{fillTerm}(\Delta \vdash \boxed{?} : \Pi x{:}A.B) \overset{\theta'}{\Longrightarrow} \langle \Delta'';\ \lambda y{:}A'.M;\ \Pi y{:}A'.B'\rangle$ by rule

$\Delta'' \vdash \Pi y{:}A'.B'$ $\texttt{solved}$ by rule

$\eta = \eta', \forall x{:}A$ and $\Delta_{\eta'} = \Delta$ by inversion on Δ_η

$\Delta'' \vdash \theta' : \eta'$ by inversion on abs sub typing

$\theta'(\nabla x{:}A.F) = \nabla y{:}A'.F'$ by rule

$|\Delta''| \vdash \nabla y{:}A'.F'$ $\texttt{wellTyped}$ by rule

$\Delta'' \vdash \nabla y{:}A'.F' \doteq$ by rule

$\Delta'' \vdash \nabla y{:}A'.F'$ $\texttt{valid}$ by rule

$(\Delta \vdash \boxed{?} : \Pi x{:}A.B)$ $\texttt{terminatesSatisfying}$ $\langle \eta; F\rangle$ by rule

Case:

$$\mathcal{D} = \frac{\begin{array}{c} b \equiv a \quad \texttt{tb}_b \geq \texttt{tb}_a \quad \Delta \vdash \texttt{inputs}_b(\overline{N}) \doteq \\ \Delta;\Phi \vdash \texttt{measure}_b\ \overline{N} \prec \texttt{measure}_a\ \overline{M} \quad \Delta \vdash \texttt{outputs}_b(\overline{N}) \doteq\!\!\Longrightarrow \eta \end{array}}{\Delta;\Phi \vdash b\,\overline{N}\ \texttt{okSG}\ a\,\overline{M} \Longrightarrow \langle \eta; \texttt{redInv}_b\ \overline{N}\rangle}$$

$|\Delta| \vdash b\,\overline{N} : \mathit{type}$ and $|\Delta| \vdash \Phi$ $\texttt{wellTyped}$ and $\Delta \vdash \Phi \,\triangleq$

and $\Delta \vdash \Phi$ $\texttt{valid}$ and $\Delta \vdash a\,\overline{M}$ $\texttt{okG}$

and $\texttt{size}_\Delta(a\,\overline{M}) = V$ and $V =_v V$ given

$|\Delta| \vdash a\,\overline{M} : \mathit{type}$ and $\vdash \Sigma$ $\texttt{tc}$ a

and $\Delta \vdash \Delta$ $\texttt{tc}$ a and $\Delta \vdash \texttt{inputs}_a(\overline{M}) \,\triangleq$ by inversion on $\texttt{okG}$

$\vdash \Sigma$ $\texttt{tc}$ b and $\Delta \vdash \Delta$ $\texttt{tc}$ b by Lemma 5.2.59

$\Delta \vdash b\,\overline{N}$ $\texttt{okG}$ by rule

$\texttt{size}_\Delta(b\,\overline{N}) = V'$ and $V' =_v V'$ by Lemma 5.2.61

$V = \langle a; \langle \texttt{inputs}_a(\overline{M}); \Delta\rangle; \texttt{tb}_a\rangle$ by inversion on $\texttt{size}$

$V' = \langle b; \langle \texttt{inputs}_b(\overline{M}); \Delta\rangle; \texttt{tb}_b\rangle$ by inversion on $\texttt{size}$

$|\Delta| \vdash (\texttt{measure}_a\ \overline{M})$ $\texttt{wellTyped}$ by Lemma 5.2.44

$\Delta \vdash (\texttt{measure}_a\ \overline{M}) \,\triangleq$ Lemma 5.2.44

$|\Delta| \vdash (\texttt{measure}_b\ \overline{N})$ $\texttt{wellTyped}$ Lemma 5.2.44

$\Delta \vdash (\texttt{measure}_b\ \overline{N}) \,\triangleq$ Lemma 5.2.44

$|\Delta| \vdash (\texttt{measure}_b\ \overline{N} \prec \texttt{measure}_a\ \overline{M})$ $\texttt{wellTyped}$ by rule

$\Delta \vdash (\texttt{measure}_b\ \overline{N} \prec \texttt{measure}_a\ \overline{M}) \,\triangleq$ by rule

$\Delta \vdash (\texttt{measure}_b\ \overline{N} \prec \texttt{measure}_a\ \overline{M})$ $\texttt{valid}$ by Definition 5.2.47

$\langle \Delta; \texttt{measure}_b\ \overline{N}\rangle <_o \langle \Delta; \texttt{measure}_a\ \overline{M}\rangle$ by inversion on $\texttt{valid}$

$V' <_v V$ by rule

$(\Delta \vdash \boxed{?} : b\,\overline{N})\,\texttt{terminates}$ by IH 1 on V', $s\,s\,s\,v$, $b\,\overline{N}$ and $b\,\overline{N}$

either $\texttt{failTerm}(\Delta \vdash \boxed{?} : b\,\overline{N})$

or $(\texttt{fillTerm}(\Delta \vdash \boxed{?} : a\,\overline{N}) \overset{\theta}{\Longrightarrow} \langle \Delta'; M; a\,\overline{N'}\rangle$ and $\Delta' \vdash a\,\overline{N'}$ $\texttt{solved})$

by inversion on $\texttt{terminates}$

assume $\texttt{failTerm}(\Delta \vdash \boxed{?} : b\,\overline{N})$ Case 1

$(\Delta \vdash \boxed{?} : b\,\overline{N})$ $\texttt{terminatesSatisfying}$ $\langle \eta; \texttt{redInv}_b\ \overline{N}\rangle$ by rule

$\texttt{fillTerm}(\Delta \vdash \boxed{?} : b\,\overline{N}) \overset{\theta}{\Longrightarrow} \langle \Delta';\, M;\, b\,\overline{N'}\rangle$ and $\Delta' \vdash b\,\overline{N'}\ \texttt{solved}$ Case 2

$\Delta \vdash \theta : \Delta'$ and $\vdash \Delta'\ \texttt{raised}$

and $\theta(b\,\overline{N}) = b\,\overline{N'}$ and $|\Delta| \vdash M : a\,\overline{N'}$ by Lemma 5.2.14

$\theta\overline{N} = \overline{N'}$ by inversion on gsub application

$|\Delta'| \vdash b\,\overline{N'} : \mathit{type}$ and $\Delta' \vdash b\,\overline{N'} \mathrel{\underline{\underline{\perp}}}$

and $\Delta' \vdash \texttt{redInv}_b\ \overline{N'}\ \texttt{valid}$ by inversion on $\texttt{solved}$

$\Delta' \vdash \texttt{outputs}_b(\overline{N'}) \mathrel{\underline{\underline{\perp}}}$ by Lemma 5.2.22

$\Delta' \vdash \theta : \eta$ by Lemma 5.2.36

$\Delta_\eta = \Delta$ by Lemma 5.2.17

$\theta(\texttt{redInv}_b\ \overline{N}) = \texttt{redInv}_b\ \overline{N'}$ by Lemma 5.2.48

$|\Delta'| \vdash (\texttt{redInv}_b\ \overline{N'})\ \texttt{wellTyped}$ by Lemma 5.2.48

$\Delta' \vdash (\texttt{redInv}_b\ \overline{N'}) \mathrel{\underline{\underline{\perp}}}$ by Lemma 5.2.48

$(\Delta \vdash \boxed{?} : b\,\overline{N})\ \texttt{terminatesSatisfying}\ \langle \eta; \texttt{redInv}_b\ \overline{N}\rangle$ by rule

□

The proof of Theorem 5.2.62 relies only on syntactically finitary methods, plus the principle of well-founded induction over the ordering $<_v$ on the syntactic category V, which is defined in terms of well-founded ordering $<_o$ over the syntactic category O, which is defined in terms of the well-founded ordering $<_s$ over the semantic domain $\mathbb{S}$. If $<_v$ has order-type α then $<_o$ has order type α^ω (it is the supremum of α^n for all n, where n is the length of O) and $<_v$ has order-type $\omega \cdot \alpha^\omega \cdot \omega$. If $\alpha = \omega$, as is the case when $\mathbb{S}$ is intended to represent the subterm ordering, then this is equivalent to the ordinal $\omega^{\omega+1}$. By the arguments in Chapter 3, this proof could be formalized in PA_3, but not PA_2, which is essentially the best that can be done.

Chapter 6

Conclusion and Related Work

The philosophical viewpoint behind the notion of syntactic finitism described in this document is well precedented. With the possible exception of the status of lexicographic orderings, the difference between finitism and syntactic finitism is more or less than the difference between the domain in which they are applied—mathematics for the former, programming languages research for the latter. The concept of finitism in mathematics goes at least as far back as Kroenecker, and in the context of programming languages research, strikingly many metatheorems have syntactically finitary proofs, including most confluence and type safety results (see [LCH07] for a syntactically finitary proof of type safety for a significant subset of Standard ML). The success of the proof assistant Twelf, which we have argued is a formalization of the notion of syntactic finitism, is itself a testament to the relevance of syntactic finitism to the metatheory of programming languages.

We do not claim to be the first to characterize a logical relation in terms of the provability of a logical predicate. Indeed the idea goes back to Tait's original paper [Tai67]. Nor are we the first to give a syntactically finitary proof of normalization of the simply-typed lambda calculus: see [Abe08] for a particularly nice proof that bears some resemblance to our proof of Theorem 2.1.11, only without using a logical relation. However, to our knowledge, the use of an assertion logic for the purpose of giving a (syntactically) finitary account of logical relations proofs is novel.

Structural logical relations are especially well suited for being formalized in the proofs as logic programs discipline, and make the proof-theoretic assumptions that a particular proof is based on explicit. Proofs by logical relations are popular in large part because they tend to scale well; structural logical relations appear to preserve this property [SS08]. Formulating conventional logical relations proofs can usually be accomplished in type theories with strong notion of inductive definition (e.g. Coq or Lego) under the proofs as (functional) programs discipline [Alt93, BW97, DX06], although these formalizations are not in the spirit of finitism. Moreover, such proofs must implicitly assume the validity of the principles they are purporting to investigate, which can, in the case of invalid reasoning principles (see [Coq86] for some plausible examples), lead to seemingly-valid "consistency" proofs for inconsistent formal systems. Often, conventional logical relations proofs make extensive use of fully impredicative, second-order quantification, which is beyond the current state of the art of ordinal analysis [Rat06].

Our application of the ideas and results from ordinal analysis (the branch of proof theory created by Gentzen's seminal work [Gen36, Gen38, Gen43]) to identify the ordinal $\omega^{\omega^{\omega}}$ with the concept of syntactic finitism is precedented by, and justified using, the classification of primitive recursive arithmetic and ordinal recursive functions in terms of fragments of Peano arithmetic. This program was initiated by [Gen43], and refined substantially by Parsons [Par66, Par70] and Mints [Min73a, Min73b]. A research program similar in spirit has been carried out for fragments of, and extensions to, Martin-Löfs, type theory [ML84], which is a formalization of mathematical reasoning that that has been justified by extensive philosophical arguments, and serves as the foundation for the proof assistant Agda [Nor07], which is used by many programming languages researchers. Overviews of this program can be found in [Set08, Kah02].

We have demonstrated that lexicographic path induction is a suitable replacement for transfinite induction in at least some settings. We are not the first to prove the consistency of a logic using an ordering from term rewriting theory: [DP98, Bit99, Urb01] show the strong normalization of cut-elimination for different formulations of first-order logic using either the multiset path ordering or lexicographic path ordering. However, because these results rely on proving termination of term rewriting systems, they cannot be scaled to arithmetic (by Gödel's second incompleteness theorem and [Buc95]).

Several other logics from the programming languages literature formulate the notion of induction defined in [ML71], which inspired the the rule *hcl*. The proofs of consistency for $FO\lambda\Delta^{\mathbb{N}}$ [MM00], Linc [MT03], and $\mathcal{G}$ [GMN08] all rely on (conventional) logical relations; the proof of consistency for LKID [Bro06] uses model theory. We are optimistic that many of these systems can be proven consistent using finitary reasoning extended with lexicographic path induction. However, the proof theoretic strength of Martin-Löf's fulls intuitionistic theory of *iterated* inductive definitions exceeds that of the small Veblen ordinal ([ML71], section 10), and thus its consistency cannot be proven by lexicographic path induction. We leave whether our technique scales to any useful logics or type theories whose proof-theoretic ordinal is greater than ε_0, but smaller than the small Veblen ordinal, to future work.

Our interpretation of syntactically finitary proofs as total logic programs is well precedented, most prominently by the proof assistant Twelf [PS99, HC05]. However, until now, a proof of the soundness of a mode/termination checker for logic programming on LF terms has only been sketched [RP96, Roh96, Pie05]. We leave a syntactically finitary characterization of a coverage checking, such as [SP03], and thus a complete account of proofs-as-logic-programs to future work. The proof assistant Abella [Gac08, GMN08] allows for reasoning very much in the spirit of syntactic finitism, using a variation of FO$\lambda\Delta^{\mathbb{N}}$ [MM00] as a logical framework. In Abella, logic programming is used to help facilitate finding metatheorems.

Our presentation of the semantics of logic programming differs sharply from the accounts in [Pfe91, RP96, Roh96], where proof-search is expressed as a small-step reduction on formulas in a *unification logic*, sometimes coupled with an explicit continuation stack. Our definition of proof search over LF terms is more reminiscent of the account given in [PW90], although our treatment of logic variables differs substantially. Our procedure is also somewhat similar to the procedure defined over first-order Hereditary Harrop formulas [MNPS91] defined in [Nad93] [Cer98], although their treatments do not deal with dependent types. As far as we are aware, our use of mode information to specify the semantics of proof search over LF terms is novel.

Syntactically finitary proofs can also be accounted for in a functional programming paradigm. The system that is perhaps most philosophically compatible with syntactic finitism is $\mathcal{M}_2^+$ of [Sch00], whose metatheoretic development focuses more on coverage analysis than on termination. A somewhat more unconventional system can be found in [DPS97], in which abstract syntax is by terms in a modal λ-calculus with a primitive-recursion operator, although this system is unable to represent dependencies.

Our work fits into the larger paradigm of programming using judgments and derivations. Other work along these lines include [Mil92a, CU04] in the logic programming paradigm, and [SPS04, Pos08, Pie08, LZH08] in the functional programming paradigm, although the system described in [LZH08] is notable for its reliance on infinitary rules.

Bibliography

[Abe08] Andreas Abel. Normalization for the simply-typed lambda-calculus in Twelf. In Carsten Schürmann, editor, *Proceedings of the Fourth International Workshop on Logical Frameworks and Meta-Languages*, pages 3 – 16. Elsevier, ENTCS 199, February 2008.

[AF98] Jeremy Avigad and Solomon Feferman. Gödel's functional ("Dialectica") interpretation. In S. Buss, editor, *The Handbook of Proof Theory*, pages 337–405. North-Holland, 1998.

[Alt93] Thorsten Altenkirch. A formalization of the strong normalization proof for System F in LEGO. In M. Bezem and J.F. Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 13–28, Utrecht, The Netherlands, March 1993. Springer-Verlag LNCS 664.

[Bit99] Elias Tahhan Bittar. Strong normalization proofs for cut-elimination in Gentzen's sequent calculi. In *Proceedings of the Symposium: Algebra and Computer Science. Helena Rasiowa in memoriam*, volume 46, pages 179–225. Banach Center Publications, 1999.

[Bro06] James Brotherston. *Sequent Calculus Proof Systems for Inductive Definitions.* PhD thesis, University of Edinburgh, November 2006.

[Buc91] Wilfried Buchholz. Notation systems for infinitary derivations. *Archive for Mathematical Logic*, 30(5–6):277–296, 1991.

[Buc95] Wilfried Buchholz. Proof-theoretic analysis of termination proofs. *Ann. Pure Appl. Logic*, 75(1-2):57–65, 1995.

[BW87] Wilfried Buchholz and Stan Wainer. Provably computable functions and the fast growing hierarchy. In S. Simpson, editor, *Logic and Combinatorics*, volume 65 of *Contemporary Mathematics*, pages 179–198. American Mathematical Society, 1987.

[BW97] B. Barras and B. Werner. Coq in Coq. Soumis, 1997.

[Cer98] Iliano Cervesato. Proof-theoretic foundation of compilation in logic programming languages. In J. Jaffar, editor, *Proceedings of the 1998 Joint International Conference and Symposium on Logic Programming (JICSLP'98)*, pages 115–129, Manchester, UK, June 1998. MIT Press.

[Coq86] Thierry Coquand. An analysis of Girard's paradox. In *Symposium on Logic Computer Science*, pages 227–236. IEEE, June 1986.

[CP03] Iliano Cervesato and Frank Pfenning. A linear spine calculus. *Journal of Logic and Computation*, 13(5):639–688, 2003.

[CR91] Walter A. Carnielli and Michael Rathjen. Hydrae and subsystems of arithmetic. Technical report, Institut fr Mathematische Logik und Grundlagenforschung Universitt Mnster, 1991.

[CU04] James Cheney and Christian Urban. Alpha-prolog: a logic programming language with names, binding and alpha-equivalence. In *ICLP*, 2004.

[DM79] Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.

[DO88] Nachum Dershowitz and Mitsuhiro Okada. Proof-theoretic techniques for term rewriting theory. In *LICS*, pages 104–111. IEEE Computer Society, 1988.

[DP98] Roy Dyckhoff and Luis Pinto. Cut-elimination and a permutation-free sequent calculus for intuitionistic logic. *Studia Logica*, 60(1):107–118, 1998.

[DPS97] Joëlle Despeyroux, Frank Pfenning, and Carsten Schürmann. Primitive recursion for higher-order abstract syntax. In R. Hindley, editor, *Proceedings of the Third International Conference on Typed Lambda Calculus and Applications (TLCA'97)*, pages 147–163, Nancy, France, April 1997. Springer-Verlag LNCS. An extended version is available as Technical Report CMU-CS-96-172, Carnegie Mellon University.

[DX06] Kevin Donnelly and Hongwei Xi. A formalization of strong normalization for simply-typed lambda calculus and System F. In Alberto Momigliano and Brigitte Pientka, editors, *Proceedings of Workshop on Logical Frameworks and Meta-Languages: Theory and Practice*, pages 109–125, Seattle, WA, August 2006. Elsevier, ENTCS 174, Issue 5.

[Dyb91] Peter Dybjer. Inductive sets and families in Martin-Löf's type theory and their set-theoretic semantics. In *Logical Frameworks*, pages 280–306. Cambridge University Press, 1991.

[Gac08] Andrew Gacek. The Abella interactive theorem prover (system description). In A. Armando, P. Baumgartner, and G. Dowek, editors, *Proceed-*

ings of IJCAR 2008, volume 5195 of *Lecture Notes in Artificial Intelligence*, pages 154–161. Springer, August 2008.

[Gal91] Jean H. Gallier. What's so special about Kruskal's theorem and the ordinal Γ_0? A survey of some results in proof theory. *Ann. Pure Appl. Logic*, 53(3):199–260, 1991.

[Gena] Gerhard Gentzen. The consistency of elementary number theory. In *The Collected Papers of Gerhard Gentzen*, pages 493–565. English translation of [Gen36].

[Genb] Gerhard Gentzen. Provability and nonprovability of restricted transfinite inductioin in elementary number theory. In *The Collected Works of Gerhard Gentzen*, pages 287–308. English translation of [Gen43].

[Gen36] Gerhard Gentzen. Die widerspruchsfreiheit der reinen zahlentheorie. *Mathematische Annalen*, pages 493–565, 1936. Translated to English as [Gena].

[Gen38] Gerhard Gentzen. Neue fassung des widerspruchsfreiheitsbeweises fur die reine zahlentheorie. *Forschung zur Logik und zur Grundlegung der exacten Wissenschaften*, 4:19–44, 1938. Translated to English as [Gen69].

[Gen43] Gerhard Gentzen. Beweisbarkeit und unbeweisbarkeit von anfangsfallen der transfiniten induktion in der reinen zahlentheorie. *Mathematische Annalen*, 119:140–161, 1943. Translated to English as [Genb].

[Gen69] Gerhard Gentzen. New version of the consistency proof for elementary number theory. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 252–286. North-Holland Publishing Co., Amsterdam, 1969. English translation of [Gen38].

[GLT89] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, 1989.

[GMN08] Andrew Gacek, Dale Miller, and Gopalan Nadathur. Combining generic judgments with recursive definitions. In F. Pfenning, editor, *Proceedings of LICS 2008*, pages 33–44. IEEE Computer Society, June 2008.

[Göd58] Kurt Gödel. Über eine bisher noch nicht benutzte erweiterung des finiten standpunktes. *Dialectica*, 12:280–287, 1958. Translated to English as [Göd94].

[Göd94] Kurt Gödel. On an extension of finitary methods which has not yet been used. In Solomon Feferman et al., editor, *Kurt Gödel: Collected Works: Volume II*, volume II, pages 271–280. Oxford University Press, 1994. English translation of [Göd58].

[HC05] Robert Harper and Karl Crary. How to believe a Twelf proof. avalaible from `http://www.cs.cmu.edu/~rwh/papers/how/believe-twelf.pdf`, May 2005.

[Her95] Hugo Herbelin. A lambda-calculus structure isomorphic to Gentzen-style sequent calculus structure. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers*, volume 933 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 1995.

[HHP87] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. In *Symposium on Logic in Computer Science*, pages 194–204. IEEE, June 1987. An extended and revised version is available as

Technical Report CMU-CS-89-173, School of Computer Science, Carnegie Mellon University.

[HK91] Bernard R. Hodgson and Clement F. Kent. A survey of ordinal interpretations of type epsilon0for termination of rewriting systems. In *Proceedings of the 2nd International CTRS Workshop on Conditional and Typed Rewriting Systems*, pages 137–142, London, UK, 1991. Springer-Verlag.

[HL07] Robert Harper and Daniel R. Licata. Mechanizing metatheory in a logical framework. *Journal of Functional Programming*, 2007.

[How70] W. A. Howard. Assignment of ordinals to terms for primitive recursive functions of finite type. In A.Kino, J. Myhill, and R. E. Vesley, editors, *Intuitionism and Proof Theory*, pages 443–458. North-Holland, 1970.

[HP05] Robert Harper and Frank Pfenning. On equivalence and canonical forms in the lf type theory. *Transactions on Computational Logic*, 6:61–101, January 2005.

[Kah02] Reinhard Kahle. Mathematical proof theory in the light of ordinal analysis. *Synthese*, 133(1–2):237–255, October 2002.

[KL80] Sam Kamin and Jean-Jacques Levy. Attemps for generalising the recursive path orderings. Unpublished lecture notes, 1980.

[Kun80] Kenneth Kunen. *Set Theory: An Introduction to Independence Proofs*, volume 102 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1980.

[LCH07] Daniel K. Lee, Karl Crary, and Robert Harper. Towards a mechanized metatheory of standard ml. In *POPL '07: Proceedings of the 34th an-*

nual ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pages 173–184, New York, NY, USA, 2007. ACM Press.

[LZH08] Daniel R. Licata, Noam Zeilberger, and Robert Harper. Focusing on binding and computation. In *LICS*, pages 241–252. IEEE Computer Society, 2008.

[Mil91] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

[Mil92a] Dale Miller. Abstract syntax and logic programming. In *Proceedings of the First and Second Russian Conferences on Logic Programming*, pages 322–337, Irkutsk and St. Petersburg, Russia, 1992. Springer-Verlag LNAI 592.

[Mil92b] Dale Miller. Unification under a mixed prefix. *Journal of Symbolic Computation*, 14:321–358, 1992.

[Min73a] G. E. Mints. Exact estimates of the provability of transfinite induction in the initial segments of arithmetic. *Journal of Mathematical Sciences*, 1(1):85–91, January 1973.

[Min73b] G.E. Mints. Quantifier-free and one-quantifier systems. *Journal of Mathematical Sciences*, 1(1):85–91, January 1973.

[ML71] Per Martin-Löf. Hauptsatz for the intuitionistic theory of iterated inductive definitions. In J.E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*. North Holland, 1971.

[ML84] Per Martin-Löf. *Intuitionistic Type Theory.* Biblioplois, Napoli, 1984. Notes of Giowanni Sambin on a series of lectues given in Padova.

[ML85] Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. Technical Report 2, Scuola di Specializzazione in Logica Matematica, Dipartimento di Matematica, Università di Siena, 1985.

[MM00] Raymond McDowell and Dale Miller. Cut-elimination for a logic with definitions and induction. *Theoretical Computer Science*, 232(1-2):91–119, 2000.

[MNPS91] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[Mos04] Ingo Lepper Georg Moser. Why ordinals are good for you. In *ESSLLI 2003 - Course Material II*, volume 6 of *Collegium Logicum*, pages 1–65. The Kurt Gödel Society, 2004.

[MT03] Alberto Momigliano and Alwen Tiu. Induction and co-induction in sequent calculus. In Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors, *TYPES*, volume 3085 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2003.

[Nad93] Gopalan Nadathur. A proof procedure for the logic of hereditary Harrop formulas. *Journal of Automated Reasoning*, 11(1):115–145, August 1993.

[Nor07] Ulf Norell. *Towards a practical programming language based on dependent type theory.* PhD thesis, Chalmers University of Technology, September 2007.

[NPP08] Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. Contextual modal type theory. *ACM Transactions on Computal Logic*, 9(3):1–49, 2008.

[Par66] Charles Parsons. Ordinal recursion in partial systems of number theory. *Notices of the American Mathematical Society*, 13:857–858, 1966.

[Par70] Charles Parsons. On a number theoretic choice schema and its relation to induction. In A. Kino, J. Myhill, and R. E. Vesley, editors, *Intuitionism and Proof Theory*, pages 459–473. North-Holland, 1970.

[Pfe91] Frank Pfenning. Logic programming in the LF logical framework. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pages 149–181. Cambridge University Press, 1991.

[Pfe92] Frank Pfenning. Computation and deduction. Unpublished lecture notes, 277 pp. Revised May 1994, April 1996, May 1992.

[Pfe95] Frank Pfenning. Structural cut elimination. In D. Kozen, editor, *Proceedings of the Tenth Annual Symposium on Logic in Computer Science*, pages 156–166, San Diego, California, June 1995. IEEE Computer Society.

[Pie05] Brigitte Pientka. Verifying termination and reduction properties about higher-order logic programs. *Journal of Automated Reasoning*, 34(2):179–207, 2005.

[Pie08] Brigitte Pientka. A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In *35th Annual ACM Symposium on Principles of Programming Languages (POPL'08)*, pages 371–382. ACM, 2008.

[Plo81] Gordon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, Aarhus, Denmark, September 1981.

[PM93] C. Paulin-Mohring. Inductive Definitions in the System Coq - Rules and Properties. In M. Bezem and J.-F. Groote, editors, *Proceedings of the conference Typed Lambda Calculi and Applications*, number 664 in Lecture Notes in Computer Science, 1993. LIP research report 92-49.

[Pos08] Adam Poswolsky. *Functional Programming with Logical Frameworks: The Delphin Project*. PhD thesis, Yale University, December 2008.

[PS98] Frank Pfenning and Carsten Schürmann. *Twelf User's Guide*, 1.2 edition, September 1998. Available as Technical Report CMU-CS-98-173, Carnegie Mellon University.

[PS99] Frank Pfenning and Carsten Schürmann. System description: Twelf — a meta-logical framework for deductive systems. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, pages 202–206, Trento, Italy, July 1999. Springer-Verlag LNAI 1632.

[PW90] David Pym and Lincoln Wallen. Investigations into proof-search in a system of first-order dependent function types. In M.E. Stickel, editor, *10th International Conference on Automated Deduction*, pages 236–250, Kaiserslautern, Germany, July 1990. Springer-Verlag LNCS 449.

[Rat06] Michael Rathjen. The art of ordinal analysis. In *Proceedings of the International Congress of Mathematicians*, volume II, pages 45–69. European Mathematical Society, 2006.

[Rat07] Michael Rathjen. Theories and ordinals: Ordinal analysis. In *Computation and Logic in the Real World*, volume 4497/2007 of *Lecture Notes in Computer Science*, pages 632–637. Springer Berlin / Heidelberg, 2007.

[Ree09] Jason Reed. Higher-order constraint simplification in dependent type theory. In *LFMTP '09: Proceedings of the Fourth International Workshop on Logical Frameworks and Meta-Languages*, pages 49–56, New York, NY, USA, 2009. ACM.

[Rob65] Joel W. Robbin. *Subrecursive Hierarchies.* PhD thesis, Princeton University, April 1965.

[Roh96] Ekkehard Rohwedder. Verifying the meta-theory of deductive systems. Unpublished draft, 1996.

[RP96] Ekkehard Rohwedder and Frank Pfenning. Mode and termination checking for higher-order logic programs. In Hanne Riis Nielson, editor, *Proceedings of the European Symposium on Programming*, pages 296–310, Linköping, Sweden, April 1996. Springer-Verlag LNCS 1058.

[Sch77] K. Schütte. *Proof Theory.* Springer-Verlag, 1977.

[Sch00] Carsten Schürmann. *Automating the Meta-Theory of Deductive Systems.* PhD thesis, Carnegie Mellon University, 2000. CMU-CS-00-146.

[Set08] Anton Setzer. Proof theory and Martin-Löf Type Theory. In P.; Bourdeau M.; Heinzmann G. Atten, M. v.; Boldini, editor, *One Hundred Years of Intuitionism (1907 – 2007)*, pages 257 – 279. Birkhäuser, 2008.

[SP03] Carsten Schürmann and Frank Pfenning. A coverage checking algorithm for LF. In David Basin and Burkhart Wolff, editors, *Proccedings of Theo-*

rem Proving in Higher Order Logics (TPHOLs '03), volume LNCS-2758, Rome, Italy, 2003. Springer Verlag.

[SPS04] Carsten Schürmann, Adam Poswolsky, and Jeffrey Sarnat. The ∇-calculus. Functional programming with higher-order encodings. Technical Report YALEU/DCS/TR-1272, Yale University, October 2004.

[SS08] Carsten Schürmann and Jeffrey Sarnat. Structural logical relations. In F. Pfenning, editor, *Proceedings of LICS 2008*, pages 69–80. IEEE Computer Society, June 2008.

[SS09] Jeffrey Sarnat and Carsten Schürmann. Lexicographic path induction. In *TLCA '09: Proceedings of the 9th International Conference on Typed Lambda Calculi and Applications*, pages 279–293, Berlin, Heidelberg, 2009. Springer-Verlag.

[Sza93] Nora Szasz. A machine checked proof that ackermann's function is not primitive recursive. In G. Plotkin G. Huet, editor, *Logical Environments*, pages 317–338. Cambridge University Press, 1993.

[Tai67] W. W. Tait. Intensional interpretation of functionals of finite type I. *Journal of Symbolic Logic*, 32:198–212, 1967.

[Tai81] W. W. Tait. Finitism. *Journal of Philosophy*, 78(9):524–546, September 1981.

[TS00] A.S. Troelstra and H. Schwichtenberg. *Basic Proof Theory.* Number 43 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, second edition, 2000.

[Urb01] Christian Urban. Strong normalisation for a Gentzen-like cut-elimination procedure. In *Proceedings of the 5th International Conference on Typed Lambda Calculi and Applications*, pages 415–42, May 2001.

[Vir99] Roberto Virga. *Higher-Order Rewriting with Dependent Types.* PhD thesis, Department of Mathematical Sciences, Carnegie Mellon University, 1999.

[Web93] H. Weber. Leopold Kronecker. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 2:5–31, 1893.

[Wei95] Andreas Weiermann. Termination proofs for term rewriting systems by lexicographic path orderings imply multiply recursive derivation lengths. *Theoretical Computer Science*, 139(1-2):355–362, 1995.