Ordered Linear Logic and Applications

Jeff Polakow

August 2001

CMU-CS-01-152

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

Thesis Committee:
Frank Pfenning, Chair
Robert Harper
John Reynolds
Dana Scott
Dale Miller, Pennsylvania State University

This research was supported in part by the National Science Foundation under grants CCR-9804014 and CCR-9619584.

Keywords: ordered linear logic, linear logic, non-commututative logic, logic programming, logical frameworks

This work is dedicated to my parents.

Acknowledgments

Firstly, and foremost, I would like to thank my principal advisor, Frank Pfenning, for his patience with me, and for teaching me most of what I know about logic and type theory.

I would also like to acknowledge some useful discussions with Kevin Watkins which led me to simplify some of this work.

Finally, I would like to thank my other advisor, John Reynolds, for all his kindness and support over the last five years.

Abstract

This thesis introduces a new logical system, ordered linear logic, which combines reasoning with unrestricted, linear, and ordered hypotheses. The logic conservatively extends (intuitionistic) linear logic, which contains both unrestricted and linear hypotheses, with a notion of ordered hypotheses. Ordered hypotheses must be used exactly once, subject to the order in which they were assumed (*i.e.*, their order cannot be changed during the course of a derivation). This ordering constraint allows for logical representations of simple data structures such as stacks and queues. We construct ordered linear logic in the style of Martin-Löf from the basic notion of a hypothetical judgement. We then show normalization for the system by constructing a sequent calculus presentation and proving cut-elimination of the sequent system.

After introducing the basic logical system, we show how to extend techniques from linear logic to achieve an ordered logic programming language, Olli, and an ordered logical framework, OLF. Olli and OLF allow quite elegant encodings of situations involving simple data structures which are not possible without ordered hypotheses. Example Olli programs include a translator to and from deBruijn notation, and a breadth-first graph traversal program. The major OLF application presented in this dissertation is an analysis of some syntactic properties of the CPS transform.

Contents

1 Introduction **13**

I Ordered Linear Logic **19**

2 Ordered Linear Logic **21**

2.1 Overview 21

2.2 Judgements 22

2.3 Notation 22

2.4 Unrestricted Hypothetical Judgements 23

2.5 Linear Hypothetical Judgements 30

2.6 Ordered Hypothetical Judgements 34

2.7 Combined Reasoning 38

2.8 Modalities 40

2.9 Other Approaches to Combined Reasoning 42

2.10 Derivation Rules for Ordered Linear Logic 44

2.11 Properties of Ordered Linear Logic 45

3 Normal Deductions **49**

3.1 Normal Deductions for Ordered Linear Logic 50

3.2 Properties of Normal Deductions 52

3.3 Directed Deductions 53

4 **Sequent Calculus** **55**

4.1 Sequent Calculus for Ordered Linear Logic 56

4.2 Admissibility of Cut 59

4.3 Sequent Calculus With Cut 62

5 **Normalization for Ordered Linear Logic** **65**

5.1 Normal Deductions and Cut-Free Sequents 67

5.2 Directed Deductions and Sequents with Cut 70

5.3 Normalization 71

II Ordered Linear Logic Programming **73**

6 **Uniform Derivations** **75**

6.1 Uniform Fragment 76

6.2 Uniform Derivation System 77

7 **Residuation** **87**

7.1 Extended Uniform Derivations 88

7.2 Residuation of Mobile Hypotheses 91

8 **Lazy Context Splitting** **97**

8.1 An Ordered IO System 97

8.2 Resource Management System for OLL 102

8.3 Correctness of RMS 105

9 **Lazy Erasure** **111**

9.1 Making $\top$ Deterministic 111

9.2 Correctness of $\top$-flags System 118

10 **Ordered Linear Logic Programming** **127**

10.1 Olli 128

10.2 Simple Examples 129

10.3 Translating to deBruijn Notation 133

10.4 Mini-ML Abstract Machine 137

10.5 Mergesort 141

10.6 Breadth-First Tree Numbering 144

10.7 Breadth-First Search Graph Numbering 145

10.8 Parsing 148

11 Eager Failure **151**

11.1 Failing Earlier in Linear Logic 151

11.2 Extension to Ordered Contexts 154

11.3 Strict Derivation System 156

11.4 Correctness of Strict Derivations 161

11.5 Strictness Properties 166

11.6 Implementation Issues 168

III Ordered Logical Framework **171**

12 Ordered Types **173**

12.1 Proof Terms 174

12.2 Canonical Forms 182

13 Ordered Logical Framework **193**

13.1 Ordered Logical Framework 194

13.2 Typing Rules 196

13.3 Definitional Equality 198

13.4 Properties of Typing and Equality 202

14 Algorithmic Equality **209**

14.1 Completeness of Algorithmic Equality 215

14.2 Soundness of Algorithmic Equality 219

15 Decidability and Canonical Forms **223**

15.1 Decidability of Equality 223

15.2 Decidability of OLF Type-Checking 225

15.3 Canonical Forms 227

15.4 DeBruijn Terms in OLF 234

16 CPS Analysis in OLF **239**

16.1 CPS terms 239

16.2 Ordered Logical Framework Representation 242

16.2.1 DS Terms 242

16.2.2 CPS Terms 243

16.3 CPS Transform 246

16.4 Bare Abstract Machine 249

16.5 Stack Abstract Machine 252

16.6 Bare and Stack Equivalence 256

16.7 Conclusion 263

IV Conclusions **265**

17 Conclusions and Future Work **267**

Chapter 1

Introduction

> Now, it is the contention of the intuitionists ... that the basic mathematical notions, above all the notion of function, ought to be interpreted in such a way that the cleavage between mathematics, classical mathematics, that is, and programming that we are witnessing at present disappears.
>
> — Per Martin-Löf
> *Constructive Mathematics and Computer Programming* [34]

It has long been known that there are strong connections between intuitionistic logic and computation. The Curry-Howard isomorphism [30] between well-typed functional programs and constructive proofs gives a logical interpretation of pure functional computation. Thus each expression, of a particular type, represents a proof of that type viewed as a formula in intuitionistic logic. Having a type system with a logical intuition gives a computational intuition. For example, pursuing the Curry-Howard isomorphism a little further, we arrive at the understanding of expression evaluation as partial proof normalization. Since we know intuitionistic logic is normalizing, we immediately know any well-typed expression in a pure functional language will evaluate to a value. Even when considering real functional languages, which might contain unrestricted recursion or effects, a logically motivated type system is useful. Type inference algorithms, for instance, are based on a logical meaning of types and the proof theory behind that logic.

A further use of the logical intuition behind a type system is in the area of logical frameworks. A logical framework is essentially a formal language in which a deductive system can be defined and reasoned about. LF [23] is a particular logical framework based on dependent types, *i.e.*, the meta-language is dependent type theory. Examples of systems which have been represented in LF include compiler verification, Mini-ML with its meta-theory, and the Church-Rosser theorem. Furthermore Twelf [48], a concrete implementation of LF, has been used as a tool to help in the discovery of new proofs; most notably a new proof of cut-elimination for classical, intuitionistic and linear logic was discovered with the system's help [45, 44]. The ability to "reason" within the type system, and the insights necessary for its implementation come from the logical interpretation of the types.

A different method of connecting intuitionistic logic and computation is found in the logic programming paradigm. A logic program is simply a collection of predicates (over some term language) and formulas which specify when predicates are true. A query, for a program, is a formula constructed from the same set of predicates as the program. Execution of a logic program corresponds to proving a query, following a fixed operational semantics, using the program formulas. If a query is provable then we know that a specific relationship, represented by the logic program, holds between the terms in the query. Additionally, all existentially quantified variables in a query must be instantiated by the proof of that query. The "witnesses" for a query's existential variable will also be returned when a sucessful proof is found.

In this setting, computation is a by-product of proof search. Thus the proof search algorithm used in a logic programming language must be relatively simple– the programmer must understand how a program will execute– and must have a reasonable computational interpetation, *i.e.*, the operational semantics should allow us to easily think of a collection of formulas as a program. Although originally conceived within classical logic, the above mentioned existential property clearly places logic programming in the realm of intuitionistic logic.

While intuitionistic logic itself is useful as a logic for describing computations, there are many aspects of computation which fall outside its scope. Knowing a function has type $A \rightarrow B$ only conveys that the function expects an argument, of type A, and produces a result, of type B. It tells us nothing about how, or even if, the function's argument is used. For this reason, researchers have been examining the computational content of richer logics hoping to find logical explanations for known phenomena as well as new possibilities for language constructs. Along these lines, linear logic [21] has received considerable attention in the programming language community over the past decade.

Unlike intuitionistic logic, linear logic can be used to express some information about how a function uses its argument. A linear function, of type $A \multimap B$, must use its argument exactly once. This little bit of extra information has been used to describe a wide range of existing language phenomena from the difference between side-effecting and non-side-effecting functions [60] to the absence of snapback in Idealized Algol [39]. Additionally, linear logic has been used as the basis for a type system which guarantees polynomial runtime of well-typed programs [29]. Furthermore, logic programming systems and logical frameworks based on linear logic can encode state at the logical level. This is an improvement (at the very least in terms of elegance) over similar systems based on intuitionistic logic which must encode state at the term level.

The key difference between linear logic and intuitionistic logic is in the notion of how a hypothesis may be used. Intuitionistic logic places no constraints upon hypotheses; they may be ignored or repeatedly used. For this reason, we will refer to intuitionistic logic as unrestricted logic and, from this point on, reserve the description "intuitionistic" for a more general property as described below. In contrast, linear logic requires hypotheses to be used exactly once, hence the name linear. This restriction on hypothesis usage has far reaching consequences which make linear logic quite different from unrestricted logic. In Section 2.5, we shall treat this topic in some detail.

Linear logic was originally conceived as a classical logic. It is classical in the sense that DeMorgan laws hold and implication may be defined in terms of disjunction and negation. Thus, linear logic can be presented as a single-sided sequent system without the notion of hypotheses. There is an "intuitionistic" version of linear logic in which the notion of hypotheses entailing a goal is primitive. From this point on, all references to linear logic shall refer to the intuitionistic variant. It is this property of a logic, having a primitive notion of entailment, for which we will use the word intuitionistic.

This thesis introduces a new intuitionistic logical system, *ordered linear logic*, which can express more information about hypotheses than linear logic. Specifically, we will be able to convey some information about the order of hypotheses in context. In unrestricted and linear logic, the active hypotheses at any point in a proof have no order. Thus there is no way to directly encode, at the logical level, that one hypothesis was assumed before another. Ordered linear logic imposes an order on the active hypotheses which can be used to formulate such encodings.

Other logical systems with an inherent notion of ordering exist [61, 56, 1]. However, none of these logics are intuitionistic. While intuitionistic versions for some of these systems exist, current research seems to be focussed primarily on the classical presentations. We feel that an ordered logical system fundamentally based on the notion of hypothetical reasoning, in the same manner as other intuitionistic logics, will be of use to the programming language community. It might also take another step towards removing the cleavage lamented by the intuitionists.

This thesis shows that an intuitionistic logic with a notion of order on hypotheses is a useful tool for programming language applications. Towards this end, in addition to introducing ordered linear logic, we will also show several applications of the system to logic programming and logical frameworks. To further bolster the decision to explore an intuitionistic logic, we point out that analogous applications for classical ordered logics are complicated by the complexity of the systems and the lack of a clear computational interpretation.

This thesis is divided into three parts. The first part formally introduces ordered linear logic and demonstrates that the system makes sense as a logical system. The logic is motivated by a review and analysis of the reasoning systems for the three types of hypotheses (unrestricted, linear, ordered) previously mentioned. Ordered linear logic itself integrates the use of the different kinds of hypotheses into one logical system. The logic is developed in the style of Martin-Löf from the basic notion of a hypothetical judgement. In addition to a natural deduction system, we also develop a sequent system, through which we show normalization, and which will be the basis of our investigation of proof search in ordered linear logic.

The second part of the thesis demonstrates how we may base a logic programming language on ordered linear logic. The language we build, christened Olli, follows in the footsteps of λProlog [36] and its linear extension Lolli [27][1]. A logic programming interpreter may be intuitively thought of as a specialized theorem prover for which proof search is both efficient and amenable to a direct computational interpretation.

We first identify a fragment of the logic, the uniform fragment, with particularly strong proof theoretic properties. Basing Olli on the uniform fragment provides the computational interpretation, see [37]; additionally, the uniform fragment will be the

[1] Lolli is not quite an extension of λProlog since it lacks some higher-order features, *e.g.*, higher-order predicates.

basis of a logical framework based on ordered linear logic, which is discussed in the third part of the dissertation. Thereafter, we concentrate on extending techniques developed for efficient linear logic proof search to the ordered case. We also include a chapter with a collection of Olli example programs which show how to use ordered hypotheses as a logical data structure. Example programs include a translation between lambda terms (represented via higher-order abstract syntax) and deBruijn notation terms, a merge sort, a breadth-first graph traversal, and a small natural language parser which handles unbounded dependencies.

The third part of the thesis shows how we may use ordered linear logic to construct a LF-style logical framework. We begin by formalizing the ordered type system implicit within the logic which gives us an ordered lambda calculus. We show that canonical forms (necessary for a logical framework) exist for the fragment of the calculus corresponding to the uniform fragment of the logic. We next add dependent types to the system, in a similar fashion as LLF [11], to get an ordered logical framework. We show that type checking remains decidable and that a suitable notion of canonical forms still exist. We show how the deBruijn translation Olli program may be viewed as a formal, LF-style, representation of the (informal) translation between lambda terms and deBruijn terms. We also show how the ordered logical framework provides an elegant means of analysing some syntactic properties of the CPS transform.

Finally, this thesis concludes with a broad summary and some thoughts on future work.

Preliminary parts of this dissertation were previously published as [51, 52, 53, 49].

Part I

Ordered Linear Logic

Chapter 2

Ordered Linear Logic

2.1 Overview

Ordered linear logic is a logic which combines reasoning with unrestricted, linear and ordered hypotheses. Unrestricted hypotheses may be used arbitrarily often, or not at all regardless of the order in which they were assumed. Linear hypotheses must be used exactly once, also without regard to the order of their assumption. Ordered hypotheses must be used exactly once subject to the order in which they were assumed. All of these modes of reasoning independently lead to coherent logical systems which have been (and continue to be) studied in their own right– intuitionistic logic, purely linear logic, and Lambek calculus.

However, these modes of reasoning are not mutually exclusive. For instance, intuitionistic linear logic can be construed as a logic combining linear and unrestricted reasoning. This combination results in a coherent logical system which allows one to use both styles of reasoning side-by-side. Thus intuitionistic linear logic is a conservative extension of intuitionistic logic since intuitionistic linear logic encompasses unrestricted reasoning. In the same manner, ordered linear logic is a conservative extension of intuitionistic linear logic.

This chapter reviews unrestricted, linear, and ordered reasoning before combining them to get ordered linear logic. After (re)constructing a formal system for each style of reasoning, we shall turn to the question of how to combine them into a coherent logical system. Finally, we give a complete presentation of ordered linear logic.

2.2 Judgements

We treat the hypothetical judgement, with an associated substitution principle, as the fundamental concept from which a (intuitionistic) logic, or reasoning system, is built. Therefore we shall take as basic three kinds of hypothetical judgement for the three reasoning styles we will review.

We view logical connectives as internalizations of the informal reasoning which makes sense for the hypothetical judgement. Thus we will describe logical connectives by means of introduction and elimination rules as usual for natural deduction systems. The introduction rules explain how to construct a proof of a formula, while the elimination rules tell us how to use a formula. From another viewpoint, introduction rules describe what information is stored in a connective while elimination rules describe what information may be obtained from a connective. Thus we have a litmus test for possible connectives– can we get out as much information as we put in?

This condition can be formalized in two different ways. If we construct a proof of a formula and then deconstruct it (*i.e.*, apply one of its elimination rules) , we should get back the information we started with. This property is captured by the notion of local reduction and corresponds to β-reduction under the Curry-Howard isomorphism. We also note that we want this property to hold for all possible construction/deconstruction combinations. Additionally, a formula should contain enough information to reconstruct a proof of itself; that is to say its proof should be constructable solely from its constituent parts. This property is captured by the concept of local expansions and corresponds to η-expansion under the Curry-Howard isomorphism. We shall only consider using logical connectives which satisfy both of these properties.

2.3 Notation

This section summarizes the notation we shall use in this chapter. All syntactic classes (i.e. x, B, Ω, etc.) can also be subscripted or primed.

For the sake of simplicity, we shall restrict ourselves to presenting first order logical systems. Thus we shall syntactically distinguish formulas from terms. We allow atomic formulas to depend upon terms and only allow quantification over terms.

We use x, y, z to denote hypothesis labels and term variables.

We use t, s for terms and a, b for term parameters[1].

We use A, B, C, etc... to denote formulas.

We use $(x_1{:}A_1)\ldots(x_n{:}A_n)$ to denote a context, or list of labelled formula occurrences. If n is 0 then the context is empty. Note that we use juxtaposition, rather than an explicit constructor, to form contexts. In order to have unambiguous proofs, we require that all labels are unique in a given context. We tacitly assume this condition holds for all contexts in the remainder of this thesis.

We use $\cdot$ to explicitly denote an empty context, or list.

We also use Γ, Δ, Ω to denote contexts. We also use juxtapostion to denote context concatenation.

We use $\mathcal{D}$, $\mathcal{E}$ to denote derivations.

We shall use the notation $\Delta \bowtie \Delta'$ to denote the non-deterministic merging of two contexts. We define the $\bowtie$ relation as follows:

$$\cdot \bowtie \cdot = \cdot \qquad \Delta(y{:}A) \bowtie \Delta' = (\Delta \bowtie \Delta')(y{:}A) \qquad \Delta \bowtie \Delta'(y{:}A) = (\Delta \bowtie \Delta')(y{:}A)$$

Note that $\bowtie$ is commutative and associative.

2.4 Unrestricted Hypothetical Judgements

We begin by reviewing a logic with unrestricted hypotheses, or familiar old intuitionistic logic. There are many informal semantics, or intuitions, which people use to think about intuitionistic logic. However, for our purposes intuitionistic logic is a way to reason about the *ability* to derive rather than the actual act of derivation itself. Thus we will informally treat an intuitionistic logic derivation of A as meaning we have the ability to derive A.

We start with a hypothetical judgement:

$$(x_1{:}A_1)\ldots(x_n{:}A_n) \vdash A$$

We shall interpret the judgement as: If we have the ability to derive each of the A_i then we have the ability to derive A.

[1] The eigenvariables "generated" by $\forall_I$ and $\exists_E$ rules– see the end of section 2.4.

We shall use Γ to denote unrestricted contexts.

We want our unrestricted judgements to satisfy several structural properties[2]. Namely, we do not want the derivability of judgements to depend upon the order of the hypotheses, nor upon the presence of extra (unused) hypotheses, nor upon the presence of redundant hypotheses. We formalize these properties as follows.

Structural Properties:

Exchange: $\Gamma_L(x_1{:}A)(x_2{:}B)\Gamma_R \vdash C$ implies $\Gamma_L(x_2{:}B)(x_1{:}A)\Gamma_R \vdash C$.

Weakening: $\Gamma \vdash A$ implies $\Gamma(x{:}B) \vdash A$.

Contraction: $\Gamma(x_1{:}A)(x_2{:}A) \vdash B$ implies $\Gamma(x_1{:}A) \vdash B$.

We will keep these properties in mind when designing the inference rules we will use. After we have a full system, we will prove as a lemma that these properties hold.

Now that we have formally introduced the notation for our hypothetical judgement and stated the structural properties it should satisfy, we must describe how to reason with such judgements. The basic intuition behind a hypothetical judgement is that we may assume we can derive the given hypotheses. We may then use hypotheses instead of actual derivations while deriving the conclusion. Thus if we are given an actual derivation of one of our hypotheses, we should be able to substitute this actual derivation for the use of the hypothesis. We formalize this intuition with the following substitution principle.

Substitution principle:

$$\Gamma_L \vdash A \text{ and } \Gamma_L(x{:}A)\Gamma_R \vdash C \text{ implies } \Gamma_L\Gamma_R \vdash C$$

Note that requiring the context for A to match part of the context for C will always be achievable as long as the expected structural properties hold for the system.[3] As with the structural rules, we will keep this substitution principle in mind when designing the inference rules and then formally prove that the substitution principle holds after we have described the full system.

[2] Properties of hypotheses have historically been called structural.

[3] We structure the substitution principle in this manner in anticipation of Chapter 13 where we shall allow dependencies among the hypotheses.

The first inference rule we will want to form is the basic rule explaining how hypotheses are used. As this rule comes from the nature of hypothetical judgements and does not pertain to any logical connective, it is neither an introduction rule nor an elimination rule.

Unrestricted hypothesis rule:

$$\frac{}{\Gamma_L(x{:}A)\Gamma_R \vdash A}\,x$$

The formulation of this rule ensures that exchange and weakening will hold for the logic since neither the location of x nor the presence of unused hypotheses matters.

We now consider implication, which may be thought of as a reflection of the hypothetical judgement into the logic itself since it allows us to express hypothetical statements as formulas.

Unrestricted Implication:

$$\frac{\Gamma(x{:}A) \vdash B}{\Gamma \vdash A \rightarrow B}\rightarrow_I \qquad \frac{\Gamma \vdash A \rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B}\rightarrow_E$$

Note that we require the hypotheses of the premises to match those of the conclusion in the elimination rule. This, when mirrored by all rules with multiple premises, allows contraction to hold for the system without any explicit structural rules.

We now show that $\rightarrow$, as characterized by its introduction and elimination rules, satisfies our two criteria from Section 2.2 for logical connectives. We first show that information is neither gained nor lost by constructing and then deconstructing a derivation of $\rightarrow$. This property is formalized by the following local reduction rule:

$$\frac{\dfrac{\begin{array}{c}\mathcal{E}\\ \Gamma(x{:}A) \vdash B\end{array}}{\Gamma \vdash A \rightarrow B}\rightarrow_I \qquad \begin{array}{c}\mathcal{D}\\ \Gamma \vdash A\end{array}}{\Gamma \vdash B}\rightarrow_E \quad \Longrightarrow \quad \begin{array}{c}\mathcal{E}[\mathcal{D}/x]\\ \Gamma \vdash B\end{array}$$

The notation $\mathcal{E}[\mathcal{D}/x]$ denotes the derivation $\mathcal{E}$ (which is a tree) with all occurrences of the hypothesis rule x (which will all be leaves) replaced by the derivation $\mathcal{D}$. Note that this reduction is just an application of the substitution principle.

We now show that a derivation of $\to$ contains sufficient information to reconstruct itself. We formalize this property with the following local expansion rule:

$$\begin{array}{c}\mathcal{D}\\ \Gamma \vdash A \to B\end{array} \quad\Longrightarrow\quad \dfrac{\dfrac{\begin{array}{c}\mathcal{D}'\\ \Gamma(x{:}A) \vdash A \to B\end{array} \qquad \dfrac{}{\Gamma(x{:}A) \vdash A}\,x}{\Gamma(x{:}A) \vdash B}\,{\to_E}}{\Gamma \vdash A \to B}\,{\to_I}$$

where $\mathcal{D}'$ is obtained from $\mathcal{D}$ by weakening.

We now turn to conjunction. We should be able to express in the logic the ability to derive two conclusions. If we can derive both A and B, we shall use $A \,\&\, B$ to reflect this fact in the logic. Furthermore, given that we know $A \,\&\, B$, we should be able to retrieve derivations of both A and B. Thus we will have two elimination rules.

Conjunction:

$$\dfrac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}\,\&_I \qquad \dfrac{\Gamma \vdash A \,\&\, B}{\Gamma \vdash A}\,\&_{E1} \qquad \dfrac{\Gamma \vdash A \,\&\, B}{\Gamma \vdash B}\,\&_{E2}$$

Again note the contraction of hypotheses in the introduction rule.

Since there are two eliminations, there are two local reductions for & depending on which elimination was used.

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}_1\\ \Gamma \vdash A\end{array} \qquad \begin{array}{c}\mathcal{D}_2\\ \Gamma \vdash B\end{array}}{\Gamma \vdash A \,\&\, B}\,\&_I}{\Gamma \vdash A}\,\&_{E1} \quad\Longrightarrow\quad \begin{array}{c}\mathcal{D}_1\\ \Gamma \vdash A\end{array}$$

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}_1\\ \Gamma \vdash A\end{array} \qquad \begin{array}{c}\mathcal{D}_2\\ \Gamma \vdash B\end{array}}{\Gamma \vdash A \,\&\, B}\,\&_I}{\Gamma \vdash B}\,\&_{E2} \quad\Longrightarrow\quad \begin{array}{c}\mathcal{D}_2\\ \Gamma \vdash B\end{array}$$

Here is the local expansion:

$$\begin{array}{c}\mathcal{D}\\ \Gamma \vdash A \,\&\, B\end{array} \quad \Longrightarrow \quad \dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma \vdash A \,\&\, B\end{array}}{\Gamma \vdash A}\,\&_{E_1} \qquad \dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma \vdash A \,\&\, B\end{array}}{\Gamma \vdash B}\,\&_{E_2}}{\Gamma \vdash A \,\&\, B}\,\&_I$$

We now consider truth, $\top$, which can be thought of as the unit of &– we have $A \,\&\, \top \equiv A \equiv \top \,\&\, A$ where $C \equiv D$ denotes that both $C \vdash D$ and $D \vdash C$ are derivable. Truth is always derivable and thus carries no information. Consequently, there will be no elimination rule since there is nothing to get out of a derivation of truth.

Truth:

$$\dfrac{}{\Gamma \vdash \top}\,\top_I \qquad \text{no elimination rule for } \top.$$

Although there is no reduction (since there is no elimination rule) we do have a notion of expansion. Any derivation of $\top$ can be "expanded" to an immediate derivation of truth from the same hypotheses. While this is not actually an expansion of the given proof, it is an expansion in the sense of reconstructing a proof of $\top$ from its constituent parts (*i.e.*, none).

$$\begin{array}{c}\mathcal{D}\\ \Gamma \vdash \top\end{array} \quad \Longrightarrow \quad \dfrac{}{\Gamma \vdash \top}\,\top_I$$

We now consider disjunction, the dual concept to conjunction. If we can derive A or we can derive B, we will express this in the logic as $A \oplus B$.[4] Furthermore, in order to make use of a derivation of $A \oplus B$, we must specify what to do in each case since we do not know which of the disjuncts we actually have the ability to derive.

Note that there will be two different introduction rules for disjunction depending on which conclusion we can actually derive. While these introduction rules are syntactically dual to the conjunction elimination rules, our disjunction elimination rule will have to break this syntactic duality.

[4] We use the notation $\oplus$ to be compatible with notation we will introduce later.

Disjunction:

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B}\oplus_{I1} \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B}\oplus_{I2} \qquad \frac{\Gamma \vdash A \oplus B \qquad \Gamma(x_1{:}A) \vdash C \quad \Gamma(x_2{:}B) \vdash C}{\Gamma \vdash C}\oplus_E$$

Since there are two introductions, there will be two corresponding reductions which rely upon the substitution principle.

$$\frac{\begin{array}{c}\mathcal{D}\\ \dfrac{\Gamma \vdash A}{\Gamma \vdash A \oplus B}\oplus_{I1}\end{array} \qquad \begin{array}{c}\mathcal{E}_1\\ \Gamma(x_1{:}A) \vdash C\end{array} \qquad \begin{array}{c}\mathcal{E}_2\\ \Gamma(x_2{:}B) \vdash C\end{array}}{\Gamma \vdash C}\oplus_E \quad \Longrightarrow \quad \begin{array}{c}\mathcal{E}_1[\mathcal{D}/x_1]\\ \Gamma \vdash C\end{array}$$

$$\frac{\begin{array}{c}\mathcal{D}\\ \dfrac{\Gamma \vdash B}{\Gamma \vdash A \oplus B}\oplus_{I2}\end{array} \qquad \begin{array}{c}\mathcal{E}_1\\ \Gamma(x_1{:}A) \vdash C\end{array} \qquad \begin{array}{c}\mathcal{E}_2\\ \Gamma(x_2{:}B) \vdash C\end{array}}{\Gamma \vdash C}\oplus_E \quad \Longrightarrow \quad \begin{array}{c}\mathcal{E}_2[\mathcal{D}/x_2]\\ \Gamma \vdash C\end{array}$$

Here is the local expansion:

$$\begin{array}{c}\mathcal{D}\\ \Gamma \vdash A \oplus B\end{array} \quad \Longrightarrow \quad \frac{\begin{array}{c}\mathcal{D}\\ \Gamma \vdash A \oplus B\end{array} \qquad \dfrac{\overline{\Gamma(x_1{:}A) \vdash A}^{\,x_1}}{\Gamma(x_1{:}A) \vdash A \oplus B}\oplus_{I1} \qquad \dfrac{\overline{\Gamma(x_2{:}B) \vdash B}^{\,x_2}}{\Gamma(x_2{:}B) \vdash A \oplus B}\oplus_{I2}}{\Gamma \vdash A \oplus B}\oplus_E$$

Finally we come to falsehood, $\mathbf{0}$,[5] which may be thought of as the unit of $\oplus$. Just as disjunction and conjunction are dual, $\mathbf{0}$ and $\top$ are dual. Since we want our logic to be sound, there will be no introduction rule for $\mathbf{0}$. However, we will have an elimination rule since there is no reason to prevent someone from assuming a derivation of $\mathbf{0}$.

Falsehood:

no introduction rule for $\mathbf{0}$. $\qquad \dfrac{\Gamma \vdash \mathbf{0}}{\Gamma \vdash C}\mathbf{0}_E$

Note that C is unconstrained; we can derive anything from a derivation of $\mathbf{0}$.

[5] Again we choose this notation to be forward compatible.

Since there is no introduction rule, there is no local reduction. However, there is an expansion:

$$\begin{array}{c}\mathcal{D}\\ \Gamma \vdash \mathbf{0}\end{array} \quad\Longrightarrow\quad \dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma \vdash \mathbf{0}\end{array}}{\Gamma \vdash \mathbf{0}}\,\mathbf{0}_E$$

For the sake of expressivity and completeness, we now consider quantification. We will assume the standard two quantifiers, $\forall$ and $\exists$, and show that they are locally sound and complete.

Universal quantification:

$$\dfrac{\Gamma \vdash A[a/x]}{\Gamma \vdash \forall x.\, A}\,\forall_I^a \qquad \dfrac{\Gamma \vdash \forall x.\, A}{\Gamma \vdash A[t/x]}\,\forall_E$$

Note that a must not appear free in the conclusion of the introduction rule. This can also be construed as another kind of judgement, usually called a parametric judgement where a is the parameter. The key idea behind a parametric judgement is that the parameter may be substituted by any term without affecting derivability of the judgement.

We have the following reduction:

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma \vdash A[a/x]\end{array}}{\Gamma \vdash \forall x.\, A}\,\forall_I^a}{\Gamma \vdash A[t/x]}\,\forall_E \quad\Longrightarrow\quad \begin{array}{c}\mathcal{D}[t/a]\\ \Gamma \vdash A[t/x]\end{array}$$

And the following expansion:

$$\begin{array}{c}\mathcal{D}\\ \Gamma \vdash \forall x.\, A\end{array} \quad\Longrightarrow\quad \dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma \vdash \forall x.\, A\end{array}}{\Gamma \vdash A[a/x]}\,\forall_E}{\Gamma \vdash \forall x.\, A}\,\forall_I^a$$

Existential quantification:

$$\dfrac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x.\, A}\,\exists_I \qquad \dfrac{\Gamma \vdash \exists x.\, A \qquad \Gamma(u{:}A[a/x]) \vdash C}{\Gamma \vdash C}\,\exists_E^a$$

Note that a must not appear free in A nor in the conclusion of the elimination rule.

We have the following reduction:

$$\dfrac{\dfrac{\overset{\mathcal{D}}{\Gamma \vdash A[t/x]}}{\Gamma \vdash \exists x.\, A}\exists_I \qquad \overset{\mathcal{E}}{\Gamma(u{:}A[a/x]) \vdash C}}{\Gamma \vdash C}\exists_E^a \quad\Longrightarrow\quad \overset{\mathcal{E}[t/a][\mathcal{D}/u]}{\Gamma \vdash C}$$

And the following expansion:

$$\overset{\mathcal{D}}{\Gamma \vdash \exists x.\, A} \quad\Longrightarrow\quad \dfrac{\overset{\mathcal{D}}{\Gamma \vdash \exists x.\, A} \qquad \dfrac{\dfrac{}{\Gamma(u{:}A[a/x]) \vdash A[a/x]}u}{\Gamma(u{:}A[a/x]) \vdash \exists x.\, A}\exists_I}{\Gamma \vdash \exists x.\, A}\exists_E^a$$

2.5 Linear Hypothetical Judgements

We now examine a logic based on linear hypotheses. Our basic hypothetical judgement will be:

$$(y_1{:}A_1)\dots(y_n{:}A_n) \vdash A$$

We use hypothesis tags y_i to emphasize that these hypotheses are linear rather than unrestricted. We shall interpret linear hypotheses as resources which must be used exactly once. We can then interpret the judgement as: If we have all the resources A_i (in any order) then we can derive A.

We use Δ to denote linear contexts.

Since these hypotheses are linear, we do not want linear contexts to satisfy contraction or weakening. Therefore the only structural property which we want to build into linear contexts is exchange.

Structural Properties:

Exchange: $\Delta_L(x_1{:}A)(x_2{:}B)\Delta_R \vdash C$ implies $\Delta_L(x_2{:}B)(x_1{:}A)\Delta_R \vdash C$.

We now state the appropriate substitution principle for linear hypothetical judgements which relates a derivation of A to a hypothesis A.

Substitution principle:

$$\Delta_A \vdash A \;\text{ and }\; \Delta_L(y{:}A)\Delta_R \vdash C \;\text{ implies }\; (\Delta_L \bowtie \Delta_A)\Delta_R \vdash C$$

Note that we re-emphasize, in this substitution principle, the irrelevance of hypothesis ordering by combining the context needed for A and the context in which hypothesis y appears with the non-deterministic merge, $\bowtie$.

The hypothesis rule will look quite different from the previous one since weakening should not hold for linear hypotheses. Actually, the absence of weakening forces the rule to only allow one hypothesis– the one being used.

Linear hypothesis rule:

$$\frac{}{y{:}A \vdash A}\,y$$

Because there can only be one hypothesis in this rule, we cannot concisely capture the exchange property as we did in the previous system. Therefore we will have to structure the other rules to allow for exchange. This will be accomplished through the use of the non-deterministic merge operator, $\bowtie$.

We come next to linear implication.

Linear Implication:

$$\frac{\Delta(y{:}A) \vdash B}{\Delta \vdash A \multimap B}\multimap_I \qquad \frac{\Delta \vdash A \multimap B \qquad \Delta_A \vdash A}{\Delta \bowtie \Delta_A \vdash B}\multimap_E$$

Note the absence of contraction in the elimination rule and instead the (non-deterministic) merging of the two contexts used in the premises. By merging contexts, we insure that each hypothesis is used only once (we restrict contraction). By allowing the contexts to be non-deterministically merged, we ensure that the order of the hypotheses is irrelevant (we allow exchange). Logical connectives whose inference rules require merging together, rather than contracting, premise contexts to form the conclusion context are called *multiplicative*.

The reduction for linear implication is, like its unrestricted counterpart, just an application of the substitution principle.

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{E}\\ \Delta(y{:}A) \vdash B\end{array}}{\Delta \vdash A \multimap B}{\multimap_I} \qquad \begin{array}{c}\mathcal{D}\\ \Delta_A \vdash A\end{array}}{\Delta \bowtie \Delta_A \vdash B}{\multimap_E} \quad\Longrightarrow\quad \begin{array}{c}\mathcal{E}[\mathcal{D}/y]\\ \Delta \bowtie \Delta_A \vdash B\end{array}$$

Here is the local expansion:

$$\begin{array}{c}\mathcal{D}\\ \Delta \vdash A \multimap B\end{array} \quad\Longrightarrow\quad \dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Delta \vdash A \multimap B\end{array} \qquad \dfrac{}{y{:}A \vdash A}{y}}{\Delta(y{:}A) \vdash B}{\multimap_E}}{\Delta \vdash A \multimap B}{\multimap_I}$$

All of the previously shown unrestricted connectives, &, $\top$, $\oplus$, $\mathbf{0}$, and the quantifiers, also make sense for linear hypothetical judgements and correspond to reasoning about the ability to derive with given resources. Thus we can interpret a derivation of the judgement $\Delta \vdash A \mathbin{\&} B$ as: Given the resources in Δ, we have the ability to derive A; and given the resources in Δ, we have the ability to derive B. Note that this differs from saying we can derive both A and B at the same time. Similarly a derivation of $\Delta \vdash A \oplus B$ can be interpreted as being able to derive either A or B. These unrestricted connectives, which require that the premises and conclusion all depend upon the same linear hypotheses, are called *additive*.

Another notion of conjunction exists for linear hypotheses which can be interpreted as actually deriving both conjuncts, at the same time. If we have derived A using the hypotheses in Δ_A and we have derived B using the hypotheses in Δ_B, then we reflect this fact in the logic as $A \otimes B$. Of course the introduction rule for $\otimes$ must have in its conclusion a context created by combining Δ_A and Δ_B since both are needed.

In order to use a derivation of $A \otimes B$, we cannot directly recover a derivation of A nor of B since we don't know which hypotheses were used by each derivation. An attempt to form such an elimination rule (along the lines of those in the previous section) would result in $\otimes$ not having both a local reduction and a local expansion. Instead, we can only use a derivation of $A \otimes B$ when we know that hypotheses $y_1{:}A$ and $y_2{:}B$ are needed in another derivtion. Then, we can replace the two hypotheses $y_1{:}A$ and $y_2{:}B$ with the hypotheses needed to derive A and B.

Multiplicative Conjunction:

$$\dfrac{\Delta_A \vdash A \qquad \Delta_B \vdash B}{\Delta_A \bowtie \Delta_B \vdash A \otimes B}\otimes_I \qquad \dfrac{\Delta \vdash A \otimes B \qquad \Delta'(y_1{:}A)(y_2{:}B) \vdash C}{\Delta' \bowtie \Delta \vdash C}\otimes_E$$

We use $\Delta_A \bowtie \Delta_B$ to combine contexts since we do not care about the order of the hypotheses; *i.e.*, we want the exchange property to hold.

We have the following reduction:

$$\dfrac{\dfrac{\begin{matrix}\mathcal{D}_1\\ \Delta_A \vdash A\end{matrix} \qquad \begin{matrix}\mathcal{D}_2\\ \Delta_B \vdash B\end{matrix}}{\Delta_A \bowtie \Delta_B \vdash A \otimes B}\otimes_I \qquad \begin{matrix}\mathcal{E}\\ \Delta'(y_1{:}A)(y_2{:}B) \vdash C\end{matrix}}{\Delta' \bowtie \Delta_A \bowtie \Delta_B \vdash C}\otimes_E \quad \Longrightarrow \quad \begin{matrix}\mathcal{E}[\mathcal{D}_1/y_1][\mathcal{D}_2/y_2]\\ \Delta' \bowtie \Delta_A \bowtie \Delta_B \vdash C\end{matrix}$$

We have the following expansion:

$$\begin{matrix}\mathcal{D}\\ \Delta \vdash A \otimes B\end{matrix} \quad \Longrightarrow \quad \dfrac{\begin{matrix}\mathcal{D}\\ \Delta \vdash A \otimes B\end{matrix} \qquad \dfrac{\dfrac{}{y_1{:}A \vdash A}y_1 \qquad \dfrac{}{y_2{:}B \vdash B}y_2}{(y_1{:}A)(y_2{:}B) \vdash A \otimes B}\otimes_I}{\Delta \vdash A \otimes B}\otimes_E$$

Just as in the unrestricted case, we can consider the unit of this conjunction. The multiplicative unit, **1**, must in essence be an empty derivation– a derivation requiring no hypotheses and thus a derivation of nothing. However, unlike the additive unit, $\top$, we can formulate both an introduction and elimination rule for **1**.

Multiplicative unit:

$$\dfrac{}{\cdot \vdash \mathbf{1}}\mathbf{1}_I \qquad \dfrac{\Delta' \vdash \mathbf{1} \qquad \Delta \vdash A}{\Delta' \bowtie \Delta \vdash A}\mathbf{1}_E$$

Note that these rules are the 0-ary versions of the rules for $\otimes$.

We have the following reduction:

$$\dfrac{\dfrac{}{\cdot \vdash \mathbf{1}}\mathbf{1}_I \qquad \begin{matrix}\mathcal{E}\\ \Delta \vdash C\end{matrix}}{\Delta \vdash C}\mathbf{1}_E \quad \Longrightarrow \quad \begin{matrix}\mathcal{E}\\ \Delta \vdash C\end{matrix}$$

We have the following expansion:

$$\begin{array}{c}\mathcal{D}\\ \Delta \vdash \mathbf{1}\end{array} \quad\Longrightarrow\quad \dfrac{\begin{array}{c}\mathcal{D}\\ \Delta \vdash \mathbf{1}\end{array} \qquad \dfrac{}{\cdot \vdash \mathbf{1}}\,\mathbf{1}_I}{\Delta \vdash \mathbf{1}}\,\mathbf{1}_E$$

While researchers have considered a multiplicative disjunction for intuitionistic linear logic [7], there seems to be no way to fit such a construction into the judgemental framework we have been developing. The notion of multiplicative disjunction seems to require allowing more than one conclusion for each judgement and thus falls outside the scope of a Martin-Löf-style natural deduction system.

2.6 Ordered Hypothetical Judgements

We finally consider a logic of ordered hypotheses. This logic is essentially the Lambek calculus [32], reconstructed from a judgemental point of view. Once again we will start with a hypothetical judgement:

$$(z_1{:}A_1)\ldots(z_n{:}A_n) \vdash A$$

where we use hypothesis tags z_i to distinguish ordered hypotheses from linear and unrestricted ones. Our interpretation of this basic judgement shall be: We can derive A when given all the resources A_i in order.

The ordered context, Ω, shall not enjoy any of the three structural properties.

We have the following substitution principle for ordered hypothetical judgements.

Ordered substitution:

$$\Omega \vdash A \;\text{ and }\; \Omega_L(z{:}A)\Omega_R \vdash C \;\text{ implies }\; \Omega_L\Omega\Omega_R \vdash C$$

Note that the hypotheses needed for A must be placed, in order, in the same location which the hypothesis A occupied in the context for C.

Since all the ordered hypotheses must be used, our ordered hypothesis rule is forced and is identical to the linear version.

Ordered hypothesis rule:

$$\dfrac{}{z{:}A \vdash A}\,z$$

The treatment of ordered implication (*i.e.*, how to reflect ordered hypothetical reasoning into the logic) is more complicated than for linear implication. Since ordered contexts will not admit the exchange rule, we have to think about where to put the new hypothesis in the ordered context. As it turns out, there are only two feasible choices for placing a new hypothesis into the ordered context.

The first possibility is to place the new hypothesis on the right of the all the other hypotheses.

Right ordered implication:

$$\dfrac{\Omega(z{:}A) \vdash B}{\Omega \vdash A \twoheadrightarrow B}\twoheadrightarrow_I \qquad \dfrac{\Omega \vdash A \twoheadrightarrow B \qquad \Omega_A \vdash A}{\Omega\Omega_A \vdash B}\twoheadrightarrow_E$$

Note that the combination of hypotheses $\Omega\Omega_A$, the concatenation of Ω and Ω_A, in the elimination rule is forced in order for there to be a local reduction rule.

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{E}\\ \Omega(z{:}A) \vdash B\end{array}}{\Omega \vdash A \twoheadrightarrow B}\twoheadrightarrow_I \qquad \begin{array}{c}\mathcal{D}\\ \Omega_A \vdash A\end{array}}{\Omega\Omega_A \vdash B}\twoheadrightarrow_E \quad\Longrightarrow\quad \begin{array}{c}\mathcal{E}[\mathcal{D}/z]\\ \Omega\Omega_A \vdash B\end{array}$$

Here is the local expansion:

$$\begin{array}{c}\mathcal{D}\\ \Omega \vdash A \twoheadrightarrow B\end{array} \quad\Longrightarrow\quad \dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Omega \vdash A \twoheadrightarrow B\end{array} \qquad \dfrac{}{z{:}A \vdash A}\,z}{\Omega(z{:}A) \vdash B}\twoheadrightarrow_E}{\Omega \vdash A \twoheadrightarrow B}\twoheadrightarrow_I$$

The other possibility for an ordered implication is to place the hypothesis on the left of the ordered context.

Left ordered implication:

$$\dfrac{(z{:}A)\Omega \vdash B}{\Omega \vdash A \rightarrowtail B}\rightarrowtail_I \qquad \dfrac{\Omega \vdash A \rightarrowtail B \qquad \Omega_A \vdash A}{\Omega_A\Omega \vdash B}\rightarrowtail_E$$

Note the reverse combination, $\Omega_A\Omega$, from the $\twoheadrightarrow_E$ rule is needed in this elimination rule.

We have the following local reduction:

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{E}\\ (z{:}A)\Omega \vdash B\end{array}}{\Omega \vdash A \rightarrowtail B}\rightarrowtail_I \qquad \begin{array}{c}\mathcal{D}\\ \Omega_A \vdash A\end{array}}{\Omega_A\Omega \vdash B}\rightarrowtail_E \qquad \Longrightarrow \qquad \begin{array}{c}\mathcal{E}[\mathcal{D}/y]\\ \Omega_A\Omega \vdash B\end{array}$$

We have the following expansion:

$$\begin{array}{c}\mathcal{D}\\ \Omega \vdash A \rightarrowtail B\end{array} \qquad \Longrightarrow \qquad \dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Omega \vdash A \rightarrowtail B\end{array} \qquad \dfrac{}{z{:}A \vdash A}z}{(z{:}A)\Omega \vdash B}\rightarrowtail_E}{\Omega \vdash A \rightarrowtail B}\rightarrowtail_I$$

Note that our choice of notation for the ordered implications has nothing to do with monics or epics from category theory and that Lambek used a slightly different notation for the two ordered implications. Specifically, $A \twoheadrightarrow B$ was written as B/A, and $A \rightarrowtail B$ was written as $A\backslash B$. We feel that the arrow notation is easier to read; furthermore, it allows us to use the familiar backwards arrow notation for logic programming in Chapter 10.

We do not have any other ordered implications because there is no way to formulate complementary introduction and elimination rules when hypotheses are added to any other location in the ordered context. Suppose we tried to place the new hypothesis somewhere in the middle of the ordered context. In order to have a general rule, we could not require an exact number of other hypotheses to already be in the ordered context, *e.g.*, placing a hypothesis two formulas from the right is not a good rule. The only other choice is to non-deterministically place the hypothesis somewhere in the ordered context. When we try to do this, we do not have enough information, in the elimination rule, to ensure that we get back a derivation of the same judgement we started with after applying a local reduction.

At this point we give several examples of derivable and non-derivable ordered hypothetical judgements which give a feel for the ordered implications. We assume implications are left-associative. First, the judgement $\cdot \vdash A \twoheadrightarrow (A \twoheadrightarrow B) \twoheadrightarrow B$ is not derivable because the A will be on the wrong side of the $A \twoheadrightarrow B$ in the ordered context. However, $\cdot \vdash (A \twoheadrightarrow B) \twoheadrightarrow A \twoheadrightarrow B$ is derivable. Similarly, $\cdot \vdash A \twoheadrightarrow (A \twoheadrightarrow B) \rightarrowtail B$ is derivable.

Like the linear hypothetical judgement, all of the additive connectives, and quantifiers, make sense for the ordered hypothetical judgement. However, the multiplicative connectives must be modified to ensure that exchange will not hold. Similarly to the implications, there will be two multiplicative conjunctions since the order in which the conjuncts were derived matters.

Right multiplicative conjunction:

$$\frac{\Omega_A \vdash A \qquad \Omega_B \vdash B}{\Omega_A\Omega_B \vdash A \bullet B}\bullet_I \qquad \frac{\Omega \vdash A \bullet B \qquad \Omega_L(z_1{:}A)(z_2{:}B)\Omega_R \vdash C}{\Omega_L\Omega\Omega_R \vdash C}\bullet_E$$

The reduction rule for this conjunction is similar to the rule for linear multiplicative conjunction.

$$\frac{\dfrac{\begin{array}{c}\mathcal{D}_1\\ \Omega_A \vdash A\end{array} \qquad \begin{array}{c}\mathcal{D}_2\\ \Omega_B \vdash B\end{array}}{\Omega_A\Omega_B \vdash A \bullet B}\bullet_I \qquad \begin{array}{c}\mathcal{E}\\ \Omega_L(z_1{:}A)(z_2{:}B)\Omega_R \vdash C\end{array}}{\Omega_L\Omega_A\Omega_B\Omega_R \vdash C}\bullet_E \quad \Longrightarrow \quad \begin{array}{c}\mathcal{E}[\mathcal{D}_1/z_1][\mathcal{D}_2/z_2]\\ \Omega_L\Omega_A\Omega_B\Omega_R \vdash C\end{array}$$

We have the following expansion:

$$\begin{array}{c}\mathcal{D}\\ \Omega \vdash A \bullet B\end{array} \quad \Longrightarrow \quad \frac{\begin{array}{c}\mathcal{D}\\ \Omega \vdash A \bullet B\end{array} \qquad \dfrac{\dfrac{}{z_1{:}A \vdash A}z_1 \qquad \dfrac{}{z_2{:}B \vdash B}z_2}{(z_1{:}A)(z_2{:}B) \vdash A \bullet B}\bullet_I}{\Omega \vdash A \bullet B}\bullet_E$$

Note that $(A \bullet B) \twoheadrightarrow C \equiv A \twoheadrightarrow B \twoheadrightarrow C$ which effectively states that $\bullet$ is left-adjunct to $\twoheadrightarrow$ as might be expected. We also point out the equivalence breaks if we replace $\twoheadrightarrow$ by $\rightarrowtail$; however if we switch the order of A and B we find that $(B \bullet A) \rightarrowtail C \equiv A \rightarrowtail B \rightarrowtail C$ does hold.

Here is the other multiplicative conjunction which can be thought of as reversing the orders of the conjuncts when used.

Left multiplicative conjunction:

$$\frac{\Omega_A \vdash A \qquad \Omega_B \vdash B}{\Omega_A\Omega_B \vdash B \circ A}\circ_I \qquad \frac{\Omega \vdash B \circ A \qquad \Omega_L(z_1{:}A)(z_2{:}B)\Omega_R \vdash C}{\Omega_L\Omega\Omega_R \vdash C}\circ_E$$

Note the reversal of A and B between the premises in the elimination rule.

Now we can formulate the following reduction:

$$\dfrac{\dfrac{\begin{matrix}\mathcal{D}_1\\ \Omega_A \vdash A\end{matrix} \qquad \begin{matrix}\mathcal{D}_2\\ \Omega_B \vdash B\end{matrix}}{\Omega_A\Omega_B \vdash B \circ A}\circ_I \qquad \begin{matrix}\mathcal{E}\\ \Omega_L(z_1{:}A)(z_2{:}B)\Omega_R \vdash C\end{matrix}}{\Omega_L\Omega_A\Omega_B\Omega_R \vdash C}\circ_E \quad\Longrightarrow\quad \begin{matrix}\mathcal{E}[\mathcal{D}_1/z_1][\mathcal{D}_2/z_2]\\ \Omega_L\Omega_A\Omega_B\Omega_R \vdash C\end{matrix}$$

We have the following expansion:

$$\begin{matrix}\mathcal{D}\\ \Omega \vdash B \circ A\end{matrix} \quad\Longrightarrow\quad \dfrac{\begin{matrix}\mathcal{D}\\ \Omega \vdash B \circ A\end{matrix} \qquad \dfrac{\dfrac{}{z_1{:}A \vdash A}z_1 \qquad \dfrac{}{z_2{:}B \vdash B}z_2}{(z_1{:}A)(z_2{:}B) \vdash B \circ A}\circ_I}{\Omega \vdash B \circ A}\circ_E$$

Note that $(A \circ B) \rightarrowtail C \equiv A \rightarrowtail B \rightarrowtail C$ which effectively states that $\circ$ is left-adjunct to $\rightarrowtail$ as might also be expected. Furthermore, $A \circ B$ is equivalent to $B \bullet A$.

Since it involves no resources, the multiplicative unit is the same as in the linear case.

2.7 Combined Reasoning

We would like to combine the three modes of reasoning into one coherent logical system which allows us to use unrestricted, linear, and ordered hypotheses side-by-side. For this purpose we shall begin with a compound hypothetical judgement containing three contexts, one for each kind of hypothesis.

$$\Gamma; \Delta; \Omega \vdash A$$

We will now need to formulate three substitution principles, one explaining how to use each kind of hypothesis. First consider how to relate ordered hypotheses to derivations. Suppose we know

$$\begin{matrix}\mathcal{E}\\ \Gamma; \Delta; \Omega_L(z{:}A)\Omega_R \vdash C\end{matrix} \qquad \text{and} \qquad \begin{matrix}\mathcal{D}\\ \Gamma; \Delta_A; \Omega_A \vdash A\end{matrix}$$

We would like to be able to substitute occurrences of z in $\mathcal{E}$ with $\mathcal{D}$ to get a new derivation of C which doesn't depend upon z. However, what about the linear and ordered hypotheses used in $\mathcal{D}$? Since z is an ordered hypthesis, it only occurs once in $\mathcal{E}$ and this occurrence respects z's place in the ordered context. Therefore, we may just throw in Δ_A with Δ and replace z with Ω_A without anything going wrong (see lemma 2). Then we get the following derivation:

$$\begin{array}{c}\mathcal{E}[\mathcal{D}/z]\\ \Gamma;\Delta \bowtie \Delta_A;\Omega_L\Omega\Omega_R \vdash C\end{array}$$

What happens when we try to substitute derivations for linear hypotheses? Suppose we know

$$\begin{array}{c}\mathcal{E}\\ \Gamma;\Delta(y{:}A);\Omega \vdash C\end{array} \qquad \text{and} \qquad \begin{array}{c}\mathcal{D}\\ \Gamma;\Delta_A;\Omega_A \vdash A\end{array}$$

We can proceed as before except for the placement of Ω_A. Since y is not an ordered variable, its usage in $\mathcal{E}$ is not constrained by the order of the variables in Ω. This leaves us with no safe place to put Ω_A since we have no way of knowing which parts of Ω will have been used at the time y is used. Consider the following example:

$$\begin{array}{c}\mathcal{E}\\ \cdot;y{:}A;z{:}B \vdash C\end{array} \qquad \dfrac{}{\cdot;\cdot;z'{:}A \vdash A}\,z' \;= \mathcal{D}$$

Note that C may be either $A \bullet B$ or $B \bullet A$.

There are only two possible ways to substitute $\mathcal{D}$ for y:

$$\begin{array}{c}\mathcal{E}[\mathcal{D}/y]\\ \cdot;\cdot;(z'{:}A)(z{:}B) \vdash C\end{array} \qquad \text{or} \qquad \begin{array}{c}\mathcal{E}[\mathcal{D}/y]\\ \cdot;\cdot;(z{:}B)(z'{:}A) \vdash C\end{array}$$

However neither resulting judgement is derivable for both of the previously mentioned choices for C.

The only way we can safely substitute a derivation $\mathcal{D}$ for y is if $\mathcal{D}$ does not require ordered hypotheses. Similar reasoning shows us that we may only substitute a derivation for an unrestricted hypothesis if that derivation doesn't depend upon ordered or linear hypotheses.

We now state the substitution principles we have arrived at:

Substitution principles:

$\Gamma_A(x{:}A)\Gamma;\Delta;\Omega \vdash C$ and $\Gamma_A;\cdot;\cdot \vdash A$ implies $\Gamma_A\Gamma;\Delta;\Omega \vdash C$.

$\Gamma;\Delta_L(y{:}A)\Delta_R;\Omega \vdash C$ and $\Gamma;\Delta_A;\cdot \vdash A$ implies $\Gamma;(\Delta_L \bowtie \Delta_A)\Delta_R;\Omega \vdash C$.

$\Gamma;\Delta;\Omega_L(z{:}A)\Omega_R \vdash C$ and $\Gamma;\Delta_A;\Omega_A \vdash A$ implies $\Gamma;\Delta \bowtie \Delta_A;\Omega_L\Omega_A\Omega_R \vdash C$.

The same considerations used to formulate the substitution principles must also guide the formulation of the inference rules. Thus we have the following rules for unrestricted implication:

$$\frac{\Gamma A;\Delta;\Omega \vdash B}{\Gamma;\Delta;\Omega \vdash A \to B}\to_I \qquad \frac{\Gamma;\Delta;\Omega \vdash A \to B \qquad \Gamma;\cdot;\cdot \vdash A}{\Gamma;\Delta;\Omega \vdash B}\to_E$$

where the linear and ordered contexts in the minor premise of the elimination rule must be empty.

For a presentation, using the three context hypothetical judgement, of the all the connectives previously considered see Section 2.10.

2.8 Modalities

While motivating the substitution principles, we concluded that we can only substitute linear hypotheses with derivations which require empty ordered contexts; and likewise we can only substitute unrestricted hypotheses with derivations which require empty linear and empty ordered contexts. These properties can be reflected into the logic with the use of unary connectives, which we sometimes call modalities. Thus we will introduce two modalities, ¡ and ![6], to respectively express independence from ordered hypotheses, and from both ordered and linear hypotheses. Their introduction rules are as follows:

$$\frac{\Gamma;\Delta;\cdot \vdash A}{\Gamma;\Delta;\cdot \vdash \text{¡}A}\text{¡}_I \qquad \frac{\Gamma;\cdot;\cdot \vdash A}{\Gamma;\cdot;\cdot \vdash\, !A}!_I$$

The elimination rules for these modalities will reflect our intuition that they capture exactly what is needed to be substitutable for linear and unrestricted hypotheses.

$$\frac{\Gamma;\Delta;\Omega \vdash \text{¡}A \qquad \Gamma;\Delta_L(y{:}A)\Delta_R;\Omega_L\Omega_R \vdash C}{\Gamma;(\Delta_L \bowtie \Delta)\Delta_R;\Omega_L\Omega\Omega_R \vdash C}\text{¡}_E \qquad \frac{\Gamma;\Delta;\Omega \vdash\, !A \qquad \Gamma(x{:}A)\Gamma';\Delta_C;\Omega_L\Omega_R \vdash C}{\Gamma\Gamma';\Delta_C \bowtie \Delta;\Omega_L\Omega\Omega_R \vdash C}!_E$$

[6] This is the same ! as in linear logic.

These elimination rules are simply the substitution principles for linear and unrestricted hypotheses without constraints on the contexts. These constraints have been captured by the modalities.

Both these connectives also satisfy our local soundness and completeness criteria as shown by the following local reductions and expansions:

Reduction for ¡:

$$\dfrac{\dfrac{\begin{matrix}\mathcal{D}\\ \Gamma;\Delta_A;\cdot\vdash A\end{matrix}}{\Gamma;\Delta_A;\cdot\vdash \text{¡}A}\,\text{¡}_I \qquad \begin{matrix}\mathcal{E}\\ \Gamma;\Delta_L(y{:}A)\Delta_R;\Omega\vdash C\end{matrix}}{\Gamma;(\Delta_L\bowtie\Delta_A)\Delta_R;\Omega\vdash C}\,\text{¡}_E \qquad\Longrightarrow\qquad \begin{matrix}\mathcal{E}[\mathcal{D}/y]\\ \Gamma;(\Delta_L\bowtie\Delta_A)\Delta_R;\Omega\vdash C\end{matrix}$$

Expansion for ¡:

$$\begin{matrix}\mathcal{D}\\ \Gamma;\Delta;\Omega\vdash \text{¡}A\end{matrix} \qquad\Longrightarrow\qquad \dfrac{\begin{matrix}\mathcal{D}\\ \Gamma;\Delta;\Omega\vdash \text{¡}A\end{matrix} \qquad \dfrac{\dfrac{}{\Gamma;y{:}A;\cdot\vdash A}\,y}{\Gamma;y{:}A;\cdot\vdash \text{¡}A}\,\text{¡}_I}{\Gamma;\Delta;\Omega\vdash \text{¡}A}\,\text{¡}_E$$

Reduction for !:

$$\dfrac{\dfrac{\begin{matrix}\mathcal{D}\\ \Gamma;\cdot;\cdot\vdash A\end{matrix}}{\Gamma;\cdot;\cdot\vdash {!}A}\,!_I \qquad \begin{matrix}\mathcal{E}\\ \Gamma(x{:}A)\Gamma';\Delta;\Omega\vdash C\end{matrix}}{\Gamma;\Delta;\Omega\vdash C}\,!_E \qquad\Longrightarrow\qquad \begin{matrix}\mathcal{E}[\mathcal{D}/x]\\ \Gamma\Gamma';\Delta;\Omega\vdash C\end{matrix}$$

Expansion for !:

$$\begin{matrix}\mathcal{D}\\ \Gamma;\Delta;\Omega\vdash {!}A\end{matrix} \qquad\Longrightarrow\qquad \dfrac{\begin{matrix}\mathcal{D}\\ \Gamma;\Delta;\Omega\vdash {!}A\end{matrix} \qquad \dfrac{\dfrac{}{\Gamma(x{:}A);\cdot;\cdot\vdash A}\,x}{\Gamma(x{:}A);\cdot;\cdot\vdash {!}A}\,!_I}{\Gamma;\Delta;\Omega\vdash {!}A}\,!_E$$

Note that both of these modalities are idempotent ($!!A \equiv {!}A$ and $\text{¡¡}A \equiv \text{¡}A$) and that ! subsumes ¡ ($!\text{¡}A \equiv {!}A$ and $\text{¡}!A \equiv {!}A$)[7]. Therefore, we do not get new modalities

[7] We now use $C \equiv D$ to denote that both $\cdot;\cdot;C\vdash D$ and $\cdot;\cdot;D\vdash C$ are derivable.

from combinations of these two. We also point out that the unrestricted and linear implications are definable using these modalities: $A \rightarrow B \equiv (!A) \twoheadrightarrow B \equiv (!A) \rightarrowtail B$ and $A \multimap B \equiv ({¡}A) \twoheadrightarrow B \equiv ({¡}A) \rightarrowtail B$.

The equivalences for $\multimap$ are established from the following substitution:

$$\mathcal{D} = \cfrac{\cfrac{}{\Gamma; y{:}A; \cdot \vdash A}\,y}{\Gamma; y{:}A; \cdot \vdash {¡}A}\,{¡}_I \qquad \text{and} \qquad \begin{array}{c}\mathcal{E}\\ \Gamma; \Delta; \Omega_L(z{:}{¡}A)\Omega_R \vdash C\end{array} \qquad \Longrightarrow \qquad \begin{array}{c}\mathcal{E}[\mathcal{D}/z]\\ \Gamma; \Delta \bowtie (y{:}A); \Omega_L\Omega_R \vdash C\end{array}$$

and the following derivation:

$$\cfrac{\cfrac{}{\Gamma; \cdot; z{:}{¡}A \vdash {¡}A}\,z \qquad \begin{array}{c}\mathcal{D}\\ \Gamma; \Delta(y{:}A); \Omega_L\Omega_R \vdash C\end{array}}{\Gamma; \Delta; \Omega_L(z{:}{¡}A)\Omega_R \vdash C}\,{¡}_E$$

which show that "mobilized" ordered hypotheses, *i.e.*, ordered hypotheses of the form ${¡}A$, are essentially (when considering derivability) the same as linear hypotheses. Similar results hold for "banged" ordered hypotheses ($!A$) and unrestricted hypotheses.

The modalities, $!$ and ${¡}$, only partially distribute across the binary connectives. In particular the following are all derivable judgements

$$\cdot;\cdot;!(A \twoheadrightarrow B) \vdash !A \twoheadrightarrow !B \quad \cdot;\cdot;!A \bullet !B \vdash !(A \bullet B) \quad \cdot;\cdot;!(A \& B) \vdash !A \& !B \quad \cdot;\cdot;!A \oplus !B \vdash !(A \oplus B)$$

while none of their converses are derivable. Furthermore, replacing $!$ with ${¡}$, $\bullet$ with $\circ$ or $\twoheadrightarrow$ with $\rightarrowtail$ does not affect derivability.

At this point our development of ordered linear logic is complete. Note that the unordered fragment of this logic is identical to intuitionistic linear logic[8]. Before giving a full description of the system, we would like to point out that there are other possibilities for combining these various modes of reasoning.

2.9 Other Approaches to Combined Reasoning

Our system, following the tradition of linear logic, combines unrestricted, linear and ordered reasoning about hypotheses; it is the hypotheses which have constraints placed upon their usage. Another possibility is to place constraints at the context

[8] $A \otimes B$, in linear logic, is equivalent to ${¡}A \bullet {¡}B$ in ordered linear logic.

level and have "mixed" contexts. This is the approach espoused by the logic of bunched implications (BI) [40] which combines unrestricted and linear reasoning (but not ordered reasoning).

In BI, contexts are not flat lists but trees. It is the tree nodes which are either unrestricted or linear. This system has two implications, $\rightarrow$ for unrestricted reasoning and $-\!\!*$ for linear reasoning. The two implications correspond to two different context constructors, one of which allows either of its arguments to be copied and erased. Thus the judgement $\cdot \vdash A -\!\!* B \rightarrow B$ is derivable (where $\cdot$ denotes either empty context) while $\cdot \vdash B \rightarrow A -\!\!* B$ is not derivable. It seems likely that one could extend BI with an ordered implication and an ordered context constructor and thus have an entirely different system from ordered linear logic which combines unrestricted, linear, and ordered reasoning.

A further possibility is to combine the formula level approach with the context level approach. That is to have a system like BI which also has modalities. This is essentially how Non-commutative Logic (NL) [1, 57] was constructed. NL combines all three types of reasoning, however unrestricted reasoning is limited to the formula level, by the use of a modality, while linear and ordered reasoning take place at the context level. Thus NL's context is a tree-like hierarchical structure (more specifically a generalization of strict partial orders called order varieties) where all nodes are linear and some nodes are ordered. This forces the mobility of formulas to be scoped.

NL is presented as a classical logic. It is thus presented as a single-sided sequent system. Furthermore its context is circular in order to have a single negation. However, there is an intuitionistic version of NL [58, 16] which does not have a circular context.

Finally, we note that ordered linear logic can be thought of as an intuitionistic version of cyclic linear logic [61], which was an early attempt to combine ordered reasoning with linear logic. However, ordered linear logic was developed independently of cyclic linear logic and has a solid basis in the tradition Martin-Löf style logic. Because of this intuitionistic grounding, ordered linear logic gives rise to a term calculus and is a suitable basis for an LF style logical framework [23].

2.10 Derivation Rules for Ordered Linear Logic

In this section we summarize the derivation rules for ordered linear logic.

$$\frac{}{\Gamma_L(x{:}A)\Gamma_R; \cdot; \cdot \vdash A}\,x \qquad \frac{}{\Gamma; y{:}A; \cdot \vdash A}\,y \qquad \frac{}{\Gamma; \cdot; z{:}A \vdash A}\,z$$

$$\frac{\Gamma(x{:}A); \Delta; \Omega \vdash B}{\Gamma; \Delta; \Omega \vdash A \rightarrow B}\rightarrow_I \qquad \frac{\Gamma; \Delta; \Omega \vdash A \rightarrow B \qquad \Gamma; \cdot; \cdot \vdash A}{\Gamma; \Delta; \Omega \vdash B}\rightarrow_E$$

$$\frac{\Gamma; \Delta(y{:}A); \Omega \vdash B}{\Gamma; \Delta; \Omega \vdash A \multimap B}\multimap_I \qquad \frac{\Gamma; \Delta; \Omega \vdash A \multimap B \qquad \Gamma; \Delta_A; \cdot \vdash A}{\Gamma; \Delta \bowtie \Delta_A; \Omega \vdash B}\multimap_E$$

$$\frac{\Gamma; \Delta; \Omega(z{:}A) \vdash B}{\Gamma; \Delta; \Omega \vdash A \twoheadrightarrow B}\twoheadrightarrow_I \qquad \frac{\Gamma; \Delta; \Omega \vdash A \twoheadrightarrow B \qquad \Gamma; \Delta_A; \Omega_A \vdash A}{\Gamma; \Delta \bowtie \Delta_A; \Omega\Omega_A \vdash B}\twoheadrightarrow_E$$

$$\frac{\Gamma; \Delta; (z{:}A)\Omega \vdash B}{\Gamma; \Delta; \Omega \vdash A \rightarrowtail B}\rightarrowtail_I \qquad \frac{\Gamma; \Delta; \Omega \vdash A \rightarrowtail B \qquad \Gamma; \Delta_A; \Omega_A \vdash A}{\Gamma; \Delta \bowtie \Delta_A; \Omega_A\Omega \vdash B}\rightarrowtail_E$$

$$\frac{\Gamma; \Delta_A; \Omega_L \vdash A \qquad \Gamma; \Delta_B; \Omega_R \vdash B}{\Gamma; \Delta_A \bowtie \Delta_B; \Omega_L\Omega_R \vdash A \bullet B}\bullet_I \qquad \frac{\Gamma; \Delta; \Omega \vdash A \bullet B \qquad \Gamma; \Delta_C; \Omega_L(z_1{:}A)(z_2{:}B)\Omega_R \vdash C}{\Gamma; \Delta \bowtie \Delta_C; \Omega_L\Omega\Omega_R \vdash C}\bullet_E$$

$$\frac{\Gamma; \Delta_A; \Omega_R \vdash A \qquad \Gamma; \Delta_B; \Omega_L \vdash B}{\Gamma; \Delta_A \bowtie \Delta_B; \Omega_L\Omega_R \vdash B \circ A}\circ_I \qquad \frac{\Gamma; \Delta; \Omega \vdash B \circ A \qquad \Gamma; \Delta_C; \Omega_L(z_1{:}A)(z_2{:}B)\Omega_R \vdash C}{\Gamma; \Delta \bowtie \Delta_C; \Omega_L\Omega\Omega_R \vdash C}\circ_E$$

$$\frac{}{\Gamma; \cdot; \cdot \vdash \mathbf{1}}\mathbf{1}_I \qquad \frac{\Gamma; \Delta; \Omega \vdash \mathbf{1} \qquad \Gamma; \Delta_C; \Omega_L\Omega_R \vdash C}{\Gamma; \Delta \bowtie \Delta_C; \Omega_L\Omega\Omega_R \vdash C}\mathbf{1}_E$$

$$\frac{\Gamma; \Delta; \Omega \vdash A \qquad \Gamma; \Delta; \Omega \vdash B}{\Gamma; \Delta; \Omega \vdash A \mathbin{\&} B}\&_I \qquad \frac{\Gamma; \Delta; \Omega \vdash A \mathbin{\&} B}{\Gamma; \Delta; \Omega \vdash A}\&_{E1} \qquad \frac{\Gamma; \Delta; \Omega \vdash A \mathbin{\&} B}{\Gamma; \Delta; \Omega \vdash B}\&_{E2}$$

$$\frac{\Gamma; \Delta; \Omega \vdash A}{\Gamma; \Delta; \Omega \vdash A \oplus B}\oplus_{I1} \qquad \frac{\Gamma; \Delta; \Omega \vdash B}{\Gamma; \Delta; \Omega \vdash A \oplus B}\oplus_{I2}$$

$$\frac{\Gamma; \Delta; \Omega \vdash A \oplus B \qquad \Gamma; \Delta_C; \Omega_L(z_1{:}A)\Omega_R \vdash C \qquad \Gamma; \Delta_C; \Omega_L(z_2{:}B)\Omega_R \vdash C}{\Gamma; \Delta \bowtie \Delta_C; \Omega_L\Omega\Omega_R \vdash C}\oplus_E$$

$$\frac{}{\Gamma; \Delta; \Omega \vdash \top}\top_I \qquad \frac{\Gamma; \Delta; \Omega \vdash \mathbf{0}}{\Gamma; \Delta \bowtie \Delta_C; \Omega_L \Omega \Omega_R \vdash C}\mathbf{0}_E$$

$$\frac{\Gamma; \Delta; \Omega \vdash [a/x]A}{\Gamma; \Delta; \Omega \vdash \forall x.\ A}\forall_I^a \qquad \frac{\Gamma; \Delta; \Omega \vdash \forall x.\ A}{\Gamma; \Delta; \Omega \vdash [t/x]A}\forall_E$$

$$\frac{\Gamma; \Delta; \Omega \vdash [t/x]A}{\Gamma; \Delta; \Omega \vdash \exists x.\ A}\exists_I \qquad \frac{\Gamma; \Delta; \Omega \vdash \exists x.\ A \qquad \Gamma; \Delta_C; \Omega_L(z{:}A[a/x])\Omega_R \vdash C}{\Gamma; \Delta \bowtie \Delta_C; \Omega_L \Omega \Omega_R \vdash C}\exists_E^a$$

$$\frac{\Gamma; \Delta; \cdot \vdash A}{\Gamma; \Delta; \cdot \vdash {¡}A}{¡}_I \qquad \frac{\Gamma; \Delta; \Omega \vdash {¡}A \qquad \Gamma; \Delta_L(y{:}A)\Delta_R; \Omega_L \Omega_R \vdash C}{\Gamma; (\Delta_L \bowtie \Delta)\Delta_R; \Omega_L \Omega \Omega_R \vdash C}{¡}_E$$

$$\frac{\Gamma; \cdot; \cdot \vdash A}{\Gamma; \cdot; \cdot \vdash !A}!_I \qquad \frac{\Gamma; \Delta; \Omega \vdash !A \qquad \Gamma(x{:}A); \Delta_C; \Omega_L \Omega_R \vdash C}{\Gamma; \Delta \bowtie \Delta_C; \Omega_L \Omega \Omega_R \vdash C}!_E$$

where a must not appear free in the conclusion the $\forall_I^a$ rule; similarly, a must not appear free in the first premise nor the conclusion of the $\exists_E^a$ rule.

2.11 Properties of Ordered Linear Logic

We now formally state and prove that the expected structural properties and substitution principles hold for ordered linear logic.

Lemma 1 (Structural Properties) *The following all hold:*

1. $\Gamma_L(x_1{:}A)(x_2{:}A)\Gamma_R; \Delta; \Omega \vdash C$ *implies* $\Gamma_L(x{:}A)\Gamma_R; \Delta; \Omega \vdash C$.

2. $\Gamma_L \Gamma_R; \Delta; \Omega \vdash C$ *implies* $\Gamma_L(x{:}A)\Gamma_R; \Delta; \Omega \vdash C$.

3. $\Gamma_L(x_1{:}A)(x_2{:}B)\Gamma_R; \Delta; \Omega \vdash C$ *implies* $\Gamma_L(x_2{:}B)(x_1{:}A)\Gamma_R; \Delta; \Omega \vdash C$.

4. $\Gamma; \Delta_L(y_1{:}A)(y_2{:}B)\Delta_R; \Omega \vdash C$ *implies* $\Gamma; \Delta_L(y_2{:}B)(y_1{:}A)\Delta_R; \Omega \vdash C$.

Proof: By structural induction on the given derivation. Note that this induction defines a structure preserving translation—only the hypotheses change for each judgment in a derivation. □

Lemma 2 (Substitution) *The following hold:*

1. $\Gamma_A(x{:}A)\Gamma; \Delta; \Omega \vdash C$ *and* $\Gamma_A; \cdot; \cdot \vdash A$ *implies* $\Gamma_A\Gamma; \Delta; \Omega \vdash C$.
2. $\Gamma; \Delta_L(y{:}A)\Delta_R; \Omega \vdash C$ *and* $\Gamma; \Delta_A; \cdot \vdash A$ *implies* $\Gamma; (\Delta_L \bowtie \Delta_A)\Delta_R; \Omega \vdash C$.
3. $\Gamma; \Delta; \Omega_L(z{:}A)\Omega_R \vdash C$ *and* $\Gamma; \Delta_A; \Omega_A \vdash A$ *implies* $\Gamma; \Delta \bowtie \Delta_A; \Omega_L\Omega_A\Omega_R \vdash C$.

Proof: By structural induction over the given derivation for C. Again we have a structure preserving translation where every use of A in the deduction of C is replaced by the given deduction of A. □

We have already shown that each connective in the logic is locally sound and complete for a single context logic. Given the introduction and elimination rules of Section 2.10, it is a simple task to extend the previous local reductions and expansions to ordered linear logic.

Now that we have a complete logical system we would like to show that it is a "good" logic. The standard test for a natural deduction system is normalization. The local reductions for each connective provide a local normalization result, but this is not enough to guarantee a global result. Thus, we will prove a normalization result for ordered linear logic.

Although it is possible, using a Kripke-style logical-relations argument, we will not directly prove a normalization result on the natural deduction system presented in this chapter. Instead, we will follow Gentzen's path and prove a cut-elimination result for a sequent calculus which is equivalent (in terms of derivability) to the natural deduction system. In addition to helping us prove normalization of ordered linear logic, the sequent calculus system is of interest in its own right.

Specifically, we will prove normalization as follows:

1. Introduce a normal natural deduction system for ordered linear logic.
    (a) This system will only allow normal derivations.

    (b) Normal derivations are trivially converted to arbitrary derivations.

2. Introduce a cut-free sequent calculus for ordered linear logic.

3. Show admissibility of cut for sequent system.

4. Show equivalence of arbitrary natural deductions and a sequent system with cut.

5. Show equivalence of normal natural deduction system and cut-free sequent calculus.

The subsequent three chapters will be occupied with executing this plan.

Chapter 3

Normal Deductions

A normal deduction is one in which no unnecesary paths are taken when proceeding from hypotheses to conclusions– in other words, a derivation in which no formula is derived and then later eliminated. The intuition behind this property is the same which underlies the local reductions (and expansions) examined in the previous chapter: the conservation of information by the introduction/elimination rule pairs. However, the local reductions for each connective only cover introductions *immediately* followed by eliminations. They are not directly applicable to situations where a formula, A, is introduced and other formulas are eliminated (possibly modifying contexts) before eliminating A.

As noted by Prawitz [55], normal deductions correspond very well to natural human reasoning. Specifically, a normal derivation can be separated into two parts. One part consists of breaking down the given hypotheses, using the elimination rules, until all the parts needed to assemble the desired conclusion are obtained. The other part consists of producing the conclusion using the introduction rules. A further nice fact about normal derivations is that they enjoy the subformula property– every formula in a derivation is a subformula of the conclusion, or of some hypothesis in the last judgement. This property is absolutely necessary if there is to be any hope of automating proof search, which we wish to do in order to have an ordered linear logic programming language.

In this chapter, we present a refined system of natural deduction for ordered linear logic which only admits normal deductions. This system is based on separating *normal* deductions, characterized by bottom-up reasoning with introduction rules, from *atomic* deductions, characterized by top-down reasoning with elimination rules. These two can meet at any point with a coercion which allows us to view any atomic deduction as normal.

3.1 Normal Deductions for Ordered Linear Logic

We characterize normal deductions with two judgements:

$$\begin{array}{ll} \Gamma; \Delta; \Omega \vdash A \uparrow & A \text{ has a normal derivation} \\ \Gamma; \Delta; \Omega \vdash A \downarrow & A \text{ has an atomic derivation} \end{array}$$

The arrow indicates the direction of reasoning allowed.

Below are the judgements for the normal natural deduction system. To improve readability, we leave hypothesis labels implicit and use generic names for the hypothesis rules.

$$\frac{}{\Gamma_L A \Gamma_R; \cdot; \cdot \vdash A \downarrow}\,\textbf{ihyp} \qquad \frac{}{\Gamma; A; \cdot \vdash A \downarrow}\,\textbf{lhyp} \qquad \frac{}{\Gamma; \cdot; A \vdash A \downarrow}\,\textbf{ohyp} \qquad \frac{\Gamma; \Delta; \Omega \vdash A \downarrow}{\Gamma; \Delta; \Omega \vdash A \uparrow}\,\textbf{coercion}$$

$$\frac{\Gamma A; \Delta; \Omega \vdash B \uparrow}{\Gamma; \Delta; \Omega \vdash A \rightarrow B \uparrow}\rightarrow_I \qquad \frac{\Gamma; \Delta; \Omega \vdash A \rightarrow B \downarrow \qquad \Gamma; \cdot; \cdot \vdash A \uparrow}{\Gamma; \Delta; \Omega \vdash B \downarrow}\rightarrow_E$$

$$\frac{\Gamma; \Delta A; \Omega \vdash B \uparrow}{\Gamma; \Delta; \Omega \vdash A \multimap B \uparrow}\multimap_I \qquad \frac{\Gamma; \Delta; \Omega \vdash A \multimap B \downarrow \qquad \Gamma; \Delta_A; \cdot \vdash A \uparrow}{\Gamma; \Delta \bowtie \Delta_A; \Omega \vdash B \downarrow}\multimap_E$$

$$\frac{\Gamma; \Delta; A\Omega \vdash B \uparrow}{\Gamma; \Delta; \Omega \vdash A \rightarrowtail B \uparrow}\rightarrowtail_I \qquad \frac{\Gamma; \Delta; \Omega \vdash A \rightarrowtail B \downarrow \qquad \Gamma; \Delta_A; \Omega_A \vdash A \uparrow}{\Gamma; \Delta \bowtie \Delta_A; \Omega_A\Omega \vdash B \downarrow}\rightarrowtail_E$$

$$\frac{\Gamma; \Delta; \Omega A \vdash B \uparrow}{\Gamma; \Delta; \Omega \vdash A \twoheadrightarrow B \uparrow}\twoheadrightarrow_I \qquad \frac{\Gamma; \Delta; \Omega \vdash A \twoheadrightarrow B \downarrow \qquad \Gamma; \Delta_A; \Omega_A \vdash A \uparrow}{\Gamma; \Delta \bowtie \Delta_A; \Omega\Omega_A \vdash B \downarrow}\twoheadrightarrow_E$$

$$\frac{\Gamma; \Delta_A; \Omega_L \vdash A \uparrow \qquad \Gamma; \Delta_B; \Omega_R \vdash B \uparrow}{\Gamma; \Delta_A \bowtie \Delta_B; \Omega_L\Omega_R \vdash A \bullet B \uparrow}\bullet_I \qquad \frac{\Gamma; \Delta; \Omega \vdash A \bullet B \downarrow \qquad \Gamma; \Delta_C; \Omega_L AB \Omega_R \vdash C \uparrow}{\Gamma; \Delta \bowtie \Delta_C; \Omega_L\Omega\Omega_R \vdash C \uparrow}\bullet_E$$

$$\frac{\Gamma; \Delta_A; \Omega_R \vdash A \uparrow \qquad \Gamma; \Delta_B; \Omega_L \vdash B \uparrow}{\Gamma; \Delta_A \bowtie \Delta_B; \Omega_L\Omega_R \vdash A \circ B \uparrow}\circ_I \qquad \frac{\Gamma; \Delta; \Omega \vdash A \circ B \downarrow \qquad \Gamma; \Delta_C; \Omega_L BA \Omega_R \vdash C \uparrow}{\Gamma; \Delta \bowtie \Delta_C; \Omega_L\Omega\Omega_R \vdash C \uparrow}\circ_E$$

$$\frac{\Gamma;\Delta;\Omega\vdash A\uparrow \qquad \Gamma;\Delta;\Omega\vdash B\uparrow}{\Gamma;\Delta;\Omega\vdash A\mathbin{\&}B\uparrow}\&_I \qquad \frac{\Gamma;\Delta;\Omega\vdash A\mathbin{\&}B\downarrow}{\Gamma;\Delta;\Omega\vdash A\downarrow}\&_{E1} \qquad \frac{\Gamma;\Delta;\Omega\vdash A\mathbin{\&}B\downarrow}{\Gamma;\Delta;\Omega\vdash B\downarrow}\&_{E2}$$

$$\frac{\Gamma;\Delta;\Omega\vdash A\uparrow}{\Gamma;\Delta;\Omega\vdash A\oplus B\uparrow}\oplus_{I1} \qquad \frac{\Gamma;\Delta;\Omega\vdash B\uparrow}{\Gamma;\Delta;\Omega\vdash A\oplus B\uparrow}\oplus_{I2}$$

$$\frac{\Gamma;\Delta;\Omega\vdash A\oplus B\downarrow \qquad \Gamma;\Delta_C;\Omega_L A\Omega_R\vdash C\uparrow \qquad \Gamma;\Delta_C;\Omega_L B\Omega_R\vdash C\uparrow}{\Gamma;\Delta\bowtie\Delta_C;\Omega_L\Omega\Omega_R\vdash C\uparrow}\oplus_E$$

$$\frac{}{\Gamma;\cdot;\cdot\vdash\mathbf{1}\uparrow}\mathbf{1}_I \qquad \frac{\Gamma;\Delta;\Omega\vdash\mathbf{1}\downarrow \qquad \Gamma;\Delta_C;\Omega_L\Omega_R\vdash C\uparrow}{\Gamma;\Delta\bowtie\Delta_C;\Omega_L\Omega\Omega_R\vdash C\uparrow}\mathbf{1}_E$$

$$\frac{}{\Gamma;\Delta;\Omega\vdash\top\uparrow}\top_I \qquad \frac{\Gamma;\Delta;\Omega\vdash\mathbf{0}\downarrow}{\Gamma;\Delta\bowtie\Delta_C;\Omega_L\Omega\Omega_R\vdash C\uparrow}\mathbf{0}_E$$

$$\frac{\Gamma;\Delta;\Omega\vdash[a/x]A\uparrow}{\Gamma;\Delta;\Omega\vdash\forall x.\,A\uparrow}\forall_I^a \qquad \frac{\Gamma;\Delta;\Omega\vdash\forall x.\,A\downarrow}{\Gamma;\Delta;\Omega\vdash[t/x]A\downarrow}\forall_E$$

$$\frac{\Gamma;\Delta;\Omega\vdash[t/x]A\uparrow}{\Gamma;\Delta;\Omega\vdash\exists x.\,A\uparrow}\exists_I \qquad \frac{\Gamma;\Delta;\Omega\vdash\exists x.\,A\downarrow \qquad \Gamma;\Delta_C;\Omega_L[a/x]A\Omega_R\vdash C\uparrow}{\Gamma;\Delta\bowtie\Delta_C;\Omega_L\Omega\Omega_R\vdash C\uparrow}\exists_E^a$$

$$\frac{\Gamma;\Delta;\cdot\vdash A\uparrow}{\Gamma;\Delta;\cdot\vdash\text{¡}A\uparrow}\text{¡}_I \qquad \frac{\Gamma;\Delta;\Omega\vdash\text{¡}A\downarrow \qquad \Gamma;\Delta_C A;\Omega_L\Omega_R\vdash C\uparrow}{\Gamma;\Delta\bowtie\Delta_C;\Omega_L\Omega\Omega_R\vdash C\uparrow}\text{¡}_E$$

$$\frac{\Gamma;\cdot;\cdot\vdash A\uparrow}{\Gamma;\cdot;\cdot\vdash !A\uparrow}!_I \qquad \frac{\Gamma;\Delta;\Omega\vdash !A\downarrow \qquad \Gamma A;\Delta_C;\Omega_L\Omega_R\vdash C\uparrow}{\Gamma;\Delta\bowtie\Delta_C;\Omega_L\Omega\Omega_R\vdash C\uparrow}!_E$$

It is easy to see that this system only allows normal proofs. Since there is no way to turn a normal ($\uparrow$) derivation into an atomic ($\downarrow$) one, it is indeed impossible to introduce and then eliminate a formula.

3.2 Properties of Normal Deductions

We remark that this system enjoys structural properties analogous to the previous system.

Lemma 3 (Normal Structural Properties) *The following all hold:*

1. $\Gamma_L A A \Gamma_R; \Delta; \Omega \vdash C \uparrow$ *implies* $\Gamma_L A \Gamma_R; \Delta; \Omega \vdash C \uparrow$.
2. $\Gamma_L \Gamma_R; \Delta; \Omega \vdash C \uparrow$ *implies* $\Gamma_L A \Gamma_R; \Delta; \Omega \vdash C \uparrow$.
3. $\Gamma_L A B \Gamma_R; \Delta; \Omega \vdash C \uparrow$ *implies* $\Gamma_L B A \Gamma_R; \Delta; \Omega \vdash C \uparrow$.
4. $\Gamma; \Delta_L A B \Delta_R; \Omega \vdash C \uparrow$ *implies* $\Gamma; \Delta_L B A \Delta_R; \Omega \vdash C \uparrow$.

5. $\Gamma_L A A \Gamma_R; \Delta; \Omega \vdash C \downarrow$ *implies* $\Gamma_L A \Gamma_R; \Delta; \Omega \vdash C \downarrow$.
6. $\Gamma_L \Gamma_R; \Delta; \Omega \vdash C \downarrow$ *implies* $\Gamma_L A \Gamma_R; \Delta; \Omega \vdash C \downarrow$.
7. $\Gamma_L A B \Gamma_R; \Delta; \Omega \vdash C \downarrow$ *implies* $\Gamma_L B A \Gamma_R; \Delta; \Omega \vdash C \downarrow$.
8. $\Gamma; \Delta_L A B \Delta_R; \Omega \vdash C \downarrow$ *implies* $\Gamma; \Delta_L B A \Delta_R; \Omega \vdash C \downarrow$.

Proof: By structural induction over the given derivation. □

Furthermore we have the following substitution principles. Notice that only an atomic derivation may be substituted for a hypothesis since the uses of assumptions are considered atomic deductions (the **ohyp**, **lhyp**, and **ihyp** rules).

Lemma 4 (Normal Substitution) *The following all hold:*

1. $\Gamma_A A \Gamma; \Delta; \Omega \vdash C \uparrow$ *and* $\Gamma_A; \cdot; \cdot \vdash A \downarrow$ *implies* $\Gamma_A \Gamma; \Delta; \Omega \vdash C \uparrow$.
2. $\Gamma; \Delta_L A \Delta_R; \Omega \vdash C \uparrow$ *and* $\Gamma; \Delta_A; \cdot \vdash A \downarrow$ *implies* $\Gamma; (\Delta_L \bowtie \Delta_A)\Delta_R; \Omega \vdash C \uparrow$.
3. $\Gamma; \Delta; \Omega_L A \Omega_R \vdash C \uparrow$ *and* $\Gamma; \Delta_A; \Omega_A \vdash A \downarrow$ *implies* $\Gamma; \Delta \bowtie \Delta_A; \Omega_L \Omega_A \Omega_R \vdash C \uparrow$.

4. $\Gamma_A A \Gamma; \Delta; \Omega \vdash C \downarrow$ *and* $\Gamma_A; \cdot; \cdot \vdash A \downarrow$ *implies* $\Gamma_A \Gamma; \Delta; \Omega \vdash C \downarrow$.

5. $\Gamma; \Delta_L A \Delta_R; \Omega \vdash C \downarrow$ *and* $\Gamma; \Delta_A; \cdot \vdash A \downarrow$ *implies* $\Gamma; (\Delta_L \bowtie \Delta_A)\Delta_R; \Omega \vdash C \downarrow$.

6. $\Gamma; \Delta; \Omega_L A \Omega_R \vdash C \downarrow$ *and* $\Gamma; \Delta_A; \Omega_A \vdash A \downarrow$ *implies* $\Gamma; \Delta \bowtie \Delta_A; \Omega_L \Omega_A \Omega_R \vdash C \downarrow$.

Proof: By structural induction over the given derivation for C. □

Since the structure of the rules in the normal system follows that of the previous system, we can easily see that the normal system simply rules out some valid deductions of the previous system. Therefore we have the following soundness theorem.

Theorem 5 *The following hold:*

1. $\Gamma; \Delta; \Omega \vdash A \uparrow$ *implies* $\Gamma; \Delta; \Omega \vdash A$
2. $\Gamma; \Delta; \Omega \vdash A \downarrow$ *implies* $\Gamma; \Delta; \Omega \vdash A$

Proof: By simple structural induction. Coercions are simply eliminated. □

The converse, that every provable proposition has a normal deduction, does indeed hold and could be proved by a Kripke logical relations argument [19]. The proof for a fragment of the system above is a minor modification of the proof of the existence of canonical forms given in [51]. Instead, we will prove it indirectly by going through a sequent calculus presentation of ordered linear logic, taking advantage of the cut elimination theorem. This will also give further validation to our sequent system, which we are interested in for its own sake, by showing that it exactly proves the propositions which have natural deductions.

3.3 Directed Deductions

Before introducing the sequent system, we introduce a third natural deduction system for ordered linear logic which is obviously equivalent to the original. This system is based on the preceding normal system and has two judgements standing for normal and atomic derivations. However, in order to recover arbitrary deductions, it also allows an additional coercion from normal to atomic derivations. We write $\Gamma; \Delta; \Omega \vdash^{+} A \uparrow$ and $\Gamma; \Delta; \Omega \vdash^{+} A \downarrow$ which is defined by exactly the same rules as the normal and atomic judgments above, plus the rule

$$\frac{\Gamma; \Delta; \Omega \vdash^{+} A \uparrow}{\Gamma; \Delta; \Omega \vdash^{+} A \downarrow}\,\textbf{lemma}$$

Theorem 6 *The following hold:*

1. $\Gamma; \Delta; \Omega \vdash^{+} A \uparrow$ *iff* $\Gamma; \Delta; \Omega \vdash A$
2. $\Gamma; \Delta; \Omega \vdash^{+} A \downarrow$ *iff* $\Gamma; \Delta; \Omega \vdash A$

Proof: In each direction, by simple structural induction on the given derivation. In the forward direction coercions are simply eliminated by the translation. In the backwards direction they are introduced if the last inference is not of the right kind. Note that these translations do not form a bijection since redundant coercions collapse. □

We also point out that Lemma 4 can be trivially extended to the directed deduction system. In succeeding chapters, we will use Lemma 4 for both normal and directed deductions.

Chapter 4

Sequent Calculus

While natural deduction systems provide an elegant formalization of intuitionistic reasoning, they are not ideally suited for analyzing properties of proof search. The main difficulty stems from the inability to directly manipulate hypotheses in a natural deduction system. A formalism better suited for analyzing proof search is the sequent calculus.

In a sequent calculus, the logical connectives are characterized by *right rules* and *left rules* which, as we shall see, correspond to the introduction and elimination rules of natural deduction. In addition we have initial sequents which play the role of coercions in the normal natural deduction system. Furthermore, the sequent calculus is usually formulated with an explicit rule for reasoning with lemmas, called *cut*, which will correspond to the lemma rule in the directed natural deduction system (the system introduced at the end of Chapter 3).

In the sequent calculus proof search proceeds in one direction, from conclusion to hypotheses (bottom-up). This in contrast to a natural deduction system which requires two separate phases of proof search, one driven by the hypotheses (using elimination rules) and another analyzing the conclusion (using introduction rules). Proof search can proceed in one direction from beginning to end because the sequent calculus gives us, via the left rules, direct access to hypotheses. This also allows dual connectives (*e.g.*, & and $\oplus$) to have syntactically dual inference rules.

In this chapter we present a sequent calculus for ordered linear logic. This sequent calculus is a conservative extension of both associative Lambek calculus [32] and the sequent system for non-commutative intuitionistic linear logic given in [8]. We will present a sequent system without an explicit cut rule. This system will correspond exactly to normal natural deductions. We will then prove that cut is an admissible rule in the system. This lets us form a sequent system with explicit cut rules which will correspond to the directed natural deduction system described at the end of Chapter 3.

4.1 Sequent Calculus for Ordered Linear Logic

Similar to natural deduction judgements, our sequents have the form:

$$\Gamma; \Delta; \Omega \Longrightarrow A$$

where Γ, Δ, Ω are lists of formulas, and A is a proposition. Again, Γ, Δ, Ω are meant to denote an intuitionistic, linear, and ordered context respectively. We sometimes refer to the formulas on the left of the sequent arrow, the hypotheses, as the antecedent, and to those on the right of the sequent arrow as the succedent. As their names suggest, left rules will operate on formulas in the antecedent, while right rules will operate on the succedent formula.

In the sequent setting, one may logically think of the three antecedent contexts as one big context where the ordered hypotheses are in a fixed relative order while the other linear and unrestricted propositions may "float". The intuitionistic propositions may also be copied or ignored. Traditionally the basic structural properties of the hypotheses– exchange, weakening, and cotraction– are explicitly formulated as inference rules in the sequent calculus. However, in keeping with the style of our natural deduction system, we will build these structural properties into the formulation of the left, right and init rules.

We start with initial sequents, which encode that all linear and ordered hypotheses must be used, while those in Γ need not be used.

$$\frac{}{\Gamma; \cdot; A \Longrightarrow A}\,\textbf{init}$$

We have two explicit structural rules: **place**, which commits a linear hypothesis to a particular place among the ordered hypotheses; and **copy**, which duplicates and places an unrestricted hypothesis.

$$\frac{\Gamma_L A \Gamma_R; \Delta; \Omega_L A \Omega_R \Longrightarrow B}{\Gamma_L A \Gamma_R; \Delta; \Omega_L \Omega_R \Longrightarrow B}\,\textbf{copy} \qquad \frac{\Gamma; \Delta_L \Delta_R; \Omega_L A \Omega_R \Longrightarrow B}{\Gamma; \Delta_L A \Delta_R; \Omega_L \Omega_R \Longrightarrow B}\,\textbf{place}$$

The following four rules describing the intuitionistic and linear implications translate the standard sequent rules for intuitionistic linear logic into our setting. Note the restrictions on the linear and ordered contexts in the two left rules which are necessary to preserve linearity and order, respectively.

$$\frac{\Gamma A; \Delta; \Omega \Longrightarrow B}{\Gamma; \Delta; \Omega \Longrightarrow A \to B}\,{\to}_R \qquad \frac{\Gamma; \Delta; \Omega_L B \Omega_R \Longrightarrow C \qquad \Gamma; \cdot; \cdot \Longrightarrow A}{\Gamma; \Delta; \Omega_L (A \to B) \Omega_R \Longrightarrow C}\,{\to}_L$$

$$\frac{\Gamma; \Delta A; \Omega \Longrightarrow B}{\Gamma; \Delta; \Omega \Longrightarrow A \multimap B}\,{\multimap}_R \qquad \frac{\Gamma; \Delta_B; \Omega_L B \Omega_R \Longrightarrow C \qquad \Gamma; \Delta_A; \cdot \Longrightarrow A}{\Gamma; \Delta_B \bowtie \Delta_A; \Omega_L (A \multimap B) \Omega_R \Longrightarrow C}\,{\multimap}_L$$

The right rule for ordered right implication $A \twoheadrightarrow B$ adds A at the right end of the ordered context. In order to allow an admissible cut rule, the left rule must then take hypotheses immediately to the right of the right implication for deriving the antecedent A. The remaining hypotheses are joined with B (in order) to derive C. We must also be careful that each linear hypothesis comes from exactly one premise, although their order does not matter (hence the merge operation $\Delta_B \bowtie \Delta_A$).

$$\frac{\Gamma; \Delta; \Omega A \Longrightarrow B}{\Gamma; \Delta; \Omega \Longrightarrow A \twoheadrightarrow B}\,{\twoheadrightarrow}_R \qquad \frac{\Gamma; \Delta_B; \Omega_L B \Omega_R \Longrightarrow C \qquad \Gamma; \Delta_A; \Omega_A \Longrightarrow A}{\Gamma; \Delta_B \bowtie \Delta_A; \Omega_L (A \twoheadrightarrow B) \Omega_A \Omega_R \Longrightarrow C}\,{\twoheadrightarrow}_L$$

The rules for left implication are symmetric.

$$\frac{\Gamma; \Delta; A \Omega \Longrightarrow B}{\Gamma; \Delta; \Omega \Longrightarrow A \rightarrowtail B}\,{\rightarrowtail}_R \qquad \frac{\Gamma; \Delta_B; \Omega_L B \Omega_R \Longrightarrow C \qquad \Gamma; \Delta_A; \Omega_A \Longrightarrow A}{\Gamma; \Delta_B \bowtie \Delta_A; \Omega_L \Omega_A (A \rightarrowtail B) \Omega_R \Longrightarrow C}\,{\rightarrowtail}_L$$

The rules for the remaining connectives do not introduce any new ideas and are essentially the rules for a sequent presentation of intuitionistic linear logic extended to include ordered contexts.

$$\frac{\Gamma; \Delta_A; \Omega_L \Longrightarrow A \qquad \Gamma; \Delta_B; \Omega_R \Longrightarrow B}{\Gamma; \Delta_A \bowtie \Delta_B; \Omega_L \Omega_R \Longrightarrow A \bullet B}\,{\bullet}_R \qquad \frac{\Gamma; \Delta; \Omega_L A B \Omega_R \Longrightarrow C}{\Gamma; \Delta; \Omega_L (A \bullet B) \Omega_R \Longrightarrow C}\,{\bullet}_L$$

$$\frac{\Gamma; \Delta_A; \Omega_R \Longrightarrow A \qquad \Gamma; \Delta_B; \Omega_L \Longrightarrow B}{\Gamma; \Delta_A \bowtie \Delta_B; \Omega_L \Omega_R \Longrightarrow A \circ B}\,{\circ}_R \qquad \frac{\Gamma; \Delta; \Omega_L B A \Omega_R \Longrightarrow C}{\Gamma; \Delta; \Omega_L (A \circ B) \Omega_R \Longrightarrow C}\,{\circ}_L$$

$$\frac{}{\Gamma; \cdot; \cdot \Longrightarrow \mathbf{1}}\,\mathbf{1}_R \qquad \frac{\Gamma; \Delta; \Omega_L \Omega_R \Longrightarrow C}{\Gamma; \Delta; \Omega_L \mathbf{1} \Omega_R \Longrightarrow C}\,\mathbf{1}_L$$

$$\frac{}{\Gamma; \Delta; \Omega \Longrightarrow \top}\,\top_R \qquad \frac{\Gamma; \Delta; \Omega \Longrightarrow A \qquad \Gamma; \Delta; \Omega \Longrightarrow B}{\Gamma; \Delta; \Omega \Longrightarrow A \mathbin{\&} B}\,\&_R$$

$$\frac{\Gamma; \Delta; \Omega_L A \Omega_R \Longrightarrow C}{\Gamma; \Delta; \Omega_L (A \mathbin{\&} B) \Omega_R \Longrightarrow C}\,\&_{L1} \qquad \frac{\Gamma; \Delta; \Omega_L B \Omega_R \Longrightarrow C}{\Gamma; \Delta; \Omega_L (A \mathbin{\&} B) \Omega_R \Longrightarrow C}\,\&_{L2}$$

$$\frac{\Gamma; \Delta; \Omega \Longrightarrow A}{\Gamma; \Delta; \Omega \Longrightarrow (A \oplus B)}\,\oplus_{R1} \qquad \frac{\Gamma; \Delta; \Omega \Longrightarrow B}{\Gamma; \Delta; \Omega \Longrightarrow (A \oplus B)}\,\oplus_{R2}$$

$$\frac{\Gamma; \Delta; \Omega_L A \Omega_R \Longrightarrow C \qquad \Gamma; \Delta; \Omega_L B \Omega_R \Longrightarrow C}{\Gamma; \Delta; \Omega_L (A \oplus B) \Omega_R \Longrightarrow C}\,\oplus_L \qquad \frac{}{\Gamma; \Delta; \Omega_L \mathbf{0} \Omega_R \Longrightarrow C}\,\mathbf{0}_L$$

$$\frac{\Gamma; \Delta; \Omega \Longrightarrow A[a/x]}{\Gamma; \Delta; \Omega \Longrightarrow \forall x.\, A}\,\forall^a_R \qquad \frac{\Gamma; \Delta; \Omega_L (A[t/x]) \Omega_R \Longrightarrow C}{\Gamma; \Delta; \Omega_L (\forall x.\, A) \Omega_R \Longrightarrow C}\,\forall_L$$

$$\frac{\Gamma; \Delta; \Omega \Longrightarrow [t/x]A}{\Gamma; \Delta; \Omega \Longrightarrow \exists x.\, A}\,\exists_R \qquad \frac{\Gamma; \Delta; \Omega_L (A[a/x]) \Omega_R \Longrightarrow C}{\Gamma; \Delta; \Omega_L (\exists x.\, A) \Omega_R \Longrightarrow C}\,\exists^a_L$$

$$\frac{\Gamma; \cdot; \cdot \Longrightarrow A}{\Gamma; \cdot; \cdot \Longrightarrow\, !A}\,!_R \qquad \frac{\Gamma A; \Delta; \Omega_L \Omega_R \Longrightarrow C}{\Gamma; \Delta; \Omega_L (!A) \Omega_R \Longrightarrow C}\,!_L$$

$$\frac{\Gamma; \Delta; \cdot \Longrightarrow A}{\Gamma; \Delta; \cdot \Longrightarrow \text{¡}A}\,\text{¡}_R \qquad \frac{\Gamma; \Delta A; \Omega_L \Omega_R \Longrightarrow C}{\Gamma; \Delta; \Omega_L (\text{¡}A) \Omega_R \Longrightarrow C}\,\text{¡}_L$$

As usual we require that a not appear free in the conclusions of the $\forall^a_R$ and $\exists^a_L$ rules.

It is clear that this system has the subformula property: only instances of subformulas of propositions present in the conclusion can appear in the derivation. Since proof search based on this form of sequent calculus proceeds bottom-up, this is a critical property. It is due, of course, to the absence of any explicit cut rule.

To give a feel for how the sequent system works, we show a sample derivation which sketches how ordered linear logic can be used for natural language parsing. Suppose $\Gamma = [(\texttt{np} \twoheadrightarrow \texttt{vp} \twoheadrightarrow \texttt{snt})\,(\texttt{tv} \twoheadrightarrow \texttt{np} \twoheadrightarrow \texttt{vp})\,(\texttt{loves} \twoheadrightarrow \texttt{tv})\,(\texttt{mary} \twoheadrightarrow \texttt{np})\,(\texttt{bob} \twoheadrightarrow \texttt{np})]$ where all the words and grammatical abbreviations[1] are atomic formulas. We may think of the formulas in Γ as a grammar for simple English sentences. For example, $\texttt{np} \twoheadrightarrow \texttt{vp} \twoheadrightarrow \texttt{snt}$ specifies that a sentence is a verb phrase to the right of a noun phrase. The phrase to be parsed with the grammar is in the ordered context. The succedent contains the grammatical pattern with which we are trying to classify the input. Thus to parse the sentence: `mary loves bob`, we would prove: $\Gamma; \cdot; \texttt{mary loves bob} \Longrightarrow \texttt{snt}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\Theta\\ \Gamma;\cdot;\texttt{np tv np}\Longrightarrow\texttt{snt}\end{array}\qquad \dfrac{}{\Gamma;\cdot;\texttt{bob}\Longrightarrow\texttt{bob}}\,\textbf{init}}{\Gamma;\cdot;\texttt{np tv}\,(\texttt{bob}\twoheadrightarrow\texttt{np})\,\texttt{bob}\Longrightarrow\texttt{snt}}\,{\twoheadrightarrow_L}\qquad \dfrac{}{\Gamma;\cdot;\texttt{loves}\Longrightarrow\texttt{loves}}\,\textbf{init}}{\Gamma;\cdot;\texttt{np}\,(\texttt{loves}\twoheadrightarrow\texttt{tv})\,\texttt{loves}\,(\texttt{bob}\twoheadrightarrow\texttt{np})\,\texttt{bob}\Longrightarrow\texttt{snt}}\,{\twoheadrightarrow_L}\qquad \dfrac{}{\Gamma;\cdot;\texttt{mary}\Longrightarrow\texttt{mary}}\,\textbf{init}}{\Gamma;\cdot;(\texttt{mary}\twoheadrightarrow\texttt{np})\,\texttt{mary}\,(\texttt{loves}\twoheadrightarrow\texttt{tv})\,\texttt{loves}\,(\texttt{bob}\twoheadrightarrow\texttt{np})\,\texttt{bob}\Longrightarrow\texttt{snt}}\,{\twoheadrightarrow_L}}{\Gamma;\cdot;\texttt{mary loves bob}\Longrightarrow\texttt{snt}}\,\textbf{copy}*\mathbf{3}$$

where $\Theta =$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma;\cdot;\texttt{snt}\Longrightarrow\texttt{snt}}\,\textbf{init}\qquad \dfrac{}{\Gamma;\cdot;\texttt{vp}\Longrightarrow\texttt{vp}}\,\textbf{init}}{\Gamma;\cdot;(\texttt{vp}\twoheadrightarrow\texttt{snt})\,\texttt{vp}\Longrightarrow\texttt{snt}}\,{\twoheadrightarrow_L}\qquad \dfrac{}{\Gamma;\cdot;\texttt{np}\Longrightarrow\texttt{np}}\,\textbf{init}}{\Gamma;\cdot;(\texttt{np}\twoheadrightarrow\texttt{vp}\twoheadrightarrow\texttt{snt})\,\texttt{np vp}\Longrightarrow\texttt{snt}}\,{\twoheadrightarrow_L}\qquad \dfrac{}{\Gamma;\cdot;\texttt{np}\Longrightarrow\texttt{np}}\,\textbf{init}}{\Gamma;\cdot;(\texttt{np}\twoheadrightarrow\texttt{vp}\twoheadrightarrow\texttt{snt})\,\texttt{np}\,(\texttt{np}\twoheadrightarrow\texttt{vp})\,\texttt{np}\Longrightarrow\texttt{snt}}\,{\twoheadrightarrow_L}\qquad \dfrac{}{\Gamma;\cdot;\texttt{tv}\Longrightarrow\texttt{tv}}\,\textbf{init}}{\Gamma;\cdot;(\texttt{np}\twoheadrightarrow\texttt{vp}\twoheadrightarrow\texttt{snt})\,\texttt{np}\,(\texttt{tv}\twoheadrightarrow\texttt{np}\twoheadrightarrow\texttt{vp})\,\texttt{tv np}\Longrightarrow\texttt{snt}}\,{\twoheadrightarrow_L}}{\Gamma;\cdot;\texttt{np tv np}\Longrightarrow\texttt{snt}}\,\textbf{copy}*\mathbf{2}$$

Note that this is not the only way to derive the end-sequent. For instance, we could have moved all instances of **copy** and **place** to the beginning of the derivation; or we could have applied $\twoheadrightarrow_L$ to the formulas in a different order.

4.2 Admissibility of Cut

We shall now validate our version of the sequent calculus by showing the admissibility of cut in the system. Towards this end, we have the following lemma.

[1] `snt` = sentence, `np` = noun phrase, `vp` = verb phrase, `tv` = transitive verb

Lemma 7 (Weakening, Contraction, and Exchange) *The following all hold:*

1. $\Gamma_L\Gamma_R;\Delta;\Omega \Longrightarrow C$ *implies* $\Gamma_L A\Gamma_R;\Delta;\Omega \Longrightarrow C$.
2. $\Gamma_L AA\Gamma_R;\Delta;\Omega \Longrightarrow C$ *implies* $\Gamma_L A\Gamma_R;\Delta;\Omega \Longrightarrow C$.
3. $\Gamma_L AB\Gamma_R;\Delta;\Omega \Longrightarrow C$ *implies* $\Gamma_L BA\Gamma_R;\Delta;\Omega \Longrightarrow C$.
4. $\Gamma;\Delta_L AB\Delta_R;\Omega \Longrightarrow C$ *implies* $\Gamma;\Delta_L BA\Delta_R;\Omega \Longrightarrow C$.

Proof: By structural induction on the given derivation. Note that the structure of the derivation remains the same. □

Our sequent system combines the ideas of a multi-zone presentation due to Andreoli [2] with implicit structural rules to permit a proof of the admissibility of cut (and thus cut elimination) by structural induction as in [44].

Theorem 8 (Admissibility of Cut)
The following three statements hold:

Cut$_\Omega$: $\Gamma;\Delta_C;\Omega_C \Longrightarrow C$ *and* $\Gamma;\Delta;\Omega_L C\Omega_R \Longrightarrow A$ *implies* $\Gamma;\Delta_C \bowtie \Delta;\Omega_L\Omega_C\Omega_R \Longrightarrow A$.

Cut$_\Delta$: $\Gamma;\Delta_C;\cdot \Longrightarrow C$ *and* $\Gamma;\Delta_L C\Delta_R;\Omega \Longrightarrow A$ *implies* $\Gamma;(\Delta_L \bowtie \Delta_C)\Delta_R;\Omega \Longrightarrow A$.

Cut$_\Gamma$: $\Gamma_L;\cdot;\cdot \Longrightarrow C$ *and* $\Gamma_L C\Gamma_R;\Delta;\Omega \Longrightarrow A$ *implies* $\Gamma_L\Gamma_R;\Delta;\Omega \Longrightarrow A$.

Proof: By induction on the structure of the cut formula C, the type of cut where Cut$_\Gamma >$ Cut$_\Delta >$ Cut$_\Omega$, and the derivations of the premises. Therefore we may apply the induction hypothesis in the following cases: 1) the cut formula gets smaller; 2) the same cut formula but we move from Cut$_\Gamma$ to Cut$_\Delta$ or Cut$_\Omega$; 3) the same cut formula but we move from Cut$_\Delta$ to Cut$_\Omega$; 4) the cut formula and type of cut stay the same but one of the derivations gets smaller.

There are 5 basic cases to consider: **init** cases where one of the premises is an **init** rule, structural cases where the cut formula is the subject of a structural rule, essential cases where the principal formula of both premises is cut, commutative cases where the cut formula is a side formula on the first or second premise. Note that these cases are not mutually exclusive.

We show some representative cases.

case 1: init cases.

case: $\dfrac{}{\Gamma;\cdot;C \Longrightarrow C}\,\textbf{init}$ and $\Gamma;\Delta;\Omega_L C \Omega_R \Longrightarrow A$ is trivial.

case: $\Gamma;\Delta;\Omega \Longrightarrow C$ and $\dfrac{}{\Gamma;\cdot;C \Longrightarrow C}\,\textbf{init}$ is trivial.

case: $\Gamma_L;\cdot;\cdot \Longrightarrow C$ and $\dfrac{}{\Gamma_L C \Gamma_R;\cdot;A \Longrightarrow A}\,\textbf{init}$ then $\dfrac{}{\Gamma_L \Gamma_R;\cdot;A \Longrightarrow A}\,\textbf{init}$

case 2: Structural cases.

case: $\Gamma_L;\cdot;\cdot \Longrightarrow C$ and $\dfrac{\Gamma_L C \Gamma_R;\Delta;\Omega_L C \Omega_R \Longrightarrow A}{\Gamma_L C \Gamma_R;\Delta;\Omega_L \Omega_R \Longrightarrow A}\,\textbf{copy}$

Then

$$\begin{array}{ll}
\Gamma_L \Gamma_R;\Delta;\Omega_L C \Omega_R \Longrightarrow A & \text{ind. hyp.(Cut}_\Gamma\text{)} \\
\Gamma_L \Gamma_R;\cdot;\cdot \Longrightarrow C & \text{weakening} \\
\Gamma_L \Gamma_R;\Delta;\Omega_L \Omega_R \Longrightarrow A & \text{ind. hyp.(Cut}_\Gamma > \text{Cut}_\Omega\text{)}
\end{array}$$

case 3: Essential cases.

case: $\dfrac{\Gamma C;\Delta;\Omega \Longrightarrow B}{\Gamma;\Delta;\Omega \Longrightarrow C \twoheadrightarrow B}\,{\twoheadrightarrow_R}$ and $\dfrac{\Gamma;\Delta_A;\Omega_L B \Omega_R \Longrightarrow A \qquad \Gamma;\cdot;\cdot \Longrightarrow C}{\Gamma;\Delta_A;\Omega_L (C \twoheadrightarrow B)\Omega_R \Longrightarrow A}\,{\twoheadrightarrow_L}$

Then

$$\begin{array}{ll}
\Gamma;\Delta;\Omega \Longrightarrow B & \text{ind. hyp.(Cut}_\Gamma\text{)} \\
\Gamma;\Delta_A \bowtie \Delta;\Omega_L \Omega \Omega_R \Longrightarrow A & \text{ind. hyp.(Cut}_\Omega\text{)}
\end{array}$$

case 4: Commutative cases where cut formula is not principal in first hypothesis (i.e. end-sequent of first given derivation can't end in a right rule).

case: $\dfrac{\Gamma;\Delta_B;\Omega_L B \Omega_R \Longrightarrow C \qquad \Gamma;\Delta_D;\Omega_D \Longrightarrow D}{\Gamma;\Delta_B \bowtie \Delta_D;\Omega_L \Omega_D (D \rightarrowtail B)\Omega_R \Longrightarrow C}\,{\rightarrowtail_L}$ and $\Gamma;\Delta;\Omega_{LC} C \Omega_{RC} \Longrightarrow A$

Then

$$\begin{array}{ll}
\Gamma;\Delta_B \bowtie \Delta;\Omega_{LC}\Omega_L B \Omega_R \Omega_{RC} \Longrightarrow A & \text{ind. hyp.} \\
\Gamma;\Delta_B \bowtie \Delta \bowtie \Delta_D;\Omega_{LC}\Omega_L \Omega_D (D \rightarrowtail B)\Omega_R \Omega_{RC} \Longrightarrow A & \rightarrowtail_L
\end{array}$$

case: $\dfrac{\Gamma_L D \Gamma_R;\Delta_C;\Omega_{LC} D \Omega_{RC} \Longrightarrow C}{\Gamma_L D \Gamma_R;\Delta_C;\Omega_{LC} \Omega_{RC} \Longrightarrow C}\,\textbf{copy}$ and $\Gamma_L D \Gamma_R;\Delta;\Omega_L C \Omega_R \Longrightarrow A$

Then

$$\begin{array}{ll}
\Gamma_L D \Gamma_R;\Delta_C \bowtie \Delta;\Omega_L \Omega_{LC} D \Omega_{RC} \Omega_R \Longrightarrow A & \text{ind. hyp.(Cut}_\Omega\text{)} \\
\Gamma_L D \Gamma_R;\Delta_C \bowtie \Delta;\Omega_L \Omega_{LC} \Omega_{RC} \Omega_R \Longrightarrow A & \textbf{copy}
\end{array}$$

case 5: commutative cases where cut formula is not principal in end-sequent of second given derivation.

case: $\Gamma;\Delta_C;\Omega_C \Longrightarrow C$ and $\dfrac{\Gamma;\Delta;A\Omega_L C\Omega_R \Longrightarrow B}{\Gamma;\Delta;\Omega_L C\Omega_R \Longrightarrow A \rightarrowtail B}\rightarrowtail_R$

Then

$$\begin{array}{ll}
\Gamma;\Delta_C \bowtie \Delta;A\Omega_L\Omega_C\Omega_R \Longrightarrow B & \text{ind. hyp.}\\
\Gamma;\Delta_C \bowtie \Delta;\Omega_L\Omega_C\Omega_R \Longrightarrow A \rightarrowtail B & \rightarrowtail_R
\end{array}$$

case: $\Gamma;\Delta_C;\Omega_C \Longrightarrow C$ and $\dfrac{\Gamma;\Delta_B;\Omega_{LL}C\Omega_{LR}B\Omega_R \Longrightarrow A \qquad \Gamma;\Delta_D;\Omega_D \Longrightarrow D}{\Gamma;\Delta_B \bowtie \Delta_D;\Omega_{LL}C\Omega_{LR}\Omega_D(D \rightarrowtail B)\Omega_R \Longrightarrow A}\rightarrowtail_L$

Then

$$\begin{array}{ll}
\Gamma;\Delta_C \bowtie \Delta_B;\Omega_{LL}\Omega_C\Omega_{LR}B\Omega_R \Longrightarrow A & \text{ind. hyp.}\\
\Gamma;\Delta_C \bowtie \Delta_B \bowtie \Delta_D;\Omega_{LL}\Omega_C\Omega_{LR}\Omega_D(D \rightarrowtail B)\Omega_R \Longrightarrow A & \rightarrowtail_L
\end{array}$$

case: $\Gamma;\Delta_C;\cdot \Longrightarrow C$ and $\dfrac{\Gamma;\Delta_L C\Delta_R;A\Omega \Longrightarrow B}{\Gamma;\Delta_L C\Delta_R;\Omega \Longrightarrow A \rightarrowtail B}\rightarrowtail_R$

Then

$$\begin{array}{ll}
\Gamma;\Delta_L \bowtie \Delta_C \bowtie \Delta_R;A\Omega \Longrightarrow B & \text{ind. hyp.}\\
\Gamma;\Delta_L \bowtie \Delta_C \bowtie \Delta_R;\Omega \Longrightarrow A \rightarrowtail B & \rightarrowtail_R
\end{array}$$

case: $\Gamma;\Delta_C;\cdot \Longrightarrow C$ and $\dfrac{\Gamma;\Delta_{BL}C\Delta_{BR};\Omega_L B\Omega_R \Longrightarrow A \qquad \Gamma;\Delta_D;\Omega_D \Longrightarrow D}{\Gamma;(\Delta_{BL}C\Delta_{BR}) \bowtie \Delta_D;\Omega_L\Omega_D(D \rightarrowtail B)\Omega_R \Longrightarrow A}\rightarrowtail_L$

Then

$$\begin{array}{ll}
\Gamma;\Delta_C \bowtie (\Delta_{BL}\Delta_{BR});\Omega_L B\Omega_R \Longrightarrow A & \text{ind. hyp.}\\
\Gamma;\Delta_C \bowtie (\Delta_{BL}\Delta_{BR}) \bowtie \Delta_D;\Omega_L\Omega_D(D \rightarrowtail B)\Omega_R \Longrightarrow A & \rightarrowtail_L
\end{array}$$

□

4.3 Sequent Calculus With Cut

Theorem 8 lets us define a second sequent system with cut which is equivalent to the previous system. We write $\Gamma;\Delta;\Omega \stackrel{+}{\Longrightarrow} A$ to denote a sequent derivation of A which may contain all of the previous sequent rules in addition to the three types of cut:

$$\dfrac{\Gamma;\Delta_C;\Omega_C \stackrel{+}{\Longrightarrow} C \qquad \Gamma;\Delta;\Omega_L C\Omega_R \stackrel{+}{\Longrightarrow} A}{\Gamma;\Delta_C \bowtie \Delta;\Omega_L\Omega_C\Omega_R \stackrel{+}{\Longrightarrow} A}\mathrm{Cut}_\Omega$$

$$\frac{\Gamma_L; \cdot; \cdot \overset{+}{\Longrightarrow} C \qquad \Gamma_L C \Gamma_R; \Delta; \Omega \overset{+}{\Longrightarrow} A}{\Gamma_L \Gamma_R; \Delta; \Omega \overset{+}{\Longrightarrow} A}\mathrm{Cut}_{\Gamma} \qquad \frac{\Gamma; \Delta_C; \cdot \overset{+}{\Longrightarrow} C \qquad \Gamma; \Delta_L C \Delta_R; \Omega \overset{+}{\Longrightarrow} A}{\Gamma; (\Delta_L \bowtie \Delta_C)\Delta_R; \Omega \overset{+}{\Longrightarrow} A}\mathrm{Cut}_{\Delta}$$

Then cut elimination follows directly.

Theorem 9 (Cut Elimination) *If* $\Gamma; \Delta; \Omega \overset{+}{\Longrightarrow} A$ *then* $\Gamma; \Delta; \Omega \Longrightarrow A$.

Proof: By structural induction on the given derivation. In the case of a cut we appeal to the induction hypothesis on both premises and then to admissibility of cut on the resulting cut-free derivations. □

Chapter 5

Normalization for Ordered Linear Logic

At this point, we have introduced four[1] different logical systems– two natural deduction systems and two sequent calculi. Additionally two of the systems are restricted versions of their respective counterparts. We have been referring (at times tacitly) to all the systems as presentations of ordered linear logic. It is now time to justify this claim. In this chapter we will formally show the correspondences between the systems of natural deduction and sequent calculus. Our methods extend [20] to cover ordered linear logic. Figure 5.1 provides a map of the relationships between the systems.

We first show how the cut-free sequent system corresponds to the normal natural deduction system. We have already remarked that normal natural deductions are those where the top-down use of elimination rules meets the bottom-up use of introduction rules in the middle:

$$\frac{\downarrow\text{elim}}{\uparrow\text{intro}}\,\textbf{coercion}$$

In a sequent system we reason entirely bottom up: the top-down uses of elimination rules are turned around and become bottom-up uses of the left rules. The right rules correspond directly to the introduction rules. They meet, not in the middle, but at the initial sequents:

$$\frac{}{\uparrow\text{left} \Longrightarrow \uparrow\text{right}}\,\textbf{init}$$

[1] five if the directed natural deduction system, $\vdash^{+}$, is considered different from the original natural deduction system

$$\begin{tikzcd}[column sep=huge, row sep=large]
\text{Directed ND} & \text{SC w/ Cut} \\
\Gamma;\Delta;\Omega \vdash A \;\equiv\; \Gamma;\Delta;\Omega \vdash^{+} A\uparrow \arrow[r, Rightarrow, shift left=2, "\text{Thm 13}"] \arrow[d, shift right, "\text{Normalization}"'] & \Gamma;\Delta;\Omega \overset{+}{\Longrightarrow} A \arrow[l, Rightarrow, shift left=2, "\text{Thm 12}"] \arrow[d, shift right, "\text{Cut elim.}"'] \\
\Gamma;\Delta;\Omega \vdash A\uparrow \arrow[r, Rightarrow, shift left=2, "\text{Thm 11}"] \arrow[u, shift right, "\text{injection}"'] & \Gamma;\Delta;\Omega \Longrightarrow A \arrow[l, Rightarrow, shift left=2, "\text{Thm 10}"] \arrow[u, shift right, "\text{injection}"'] \\
\text{normal ND} & \text{cut-free SC}
\end{tikzcd}$$

Figure 5.1: Correspondences

5.1 Normal Deductions and Cut-Free Sequents

We now formally state the soundness of cut-free sequents with respect to normal natural deductions.

Theorem 10 $\Gamma; \Delta; \Omega \Longrightarrow A$ *implies* $\Gamma; \Delta; \Omega \vdash A \uparrow$

Proof: By structural induction on the given derivation. **init** rules are mapped to instances of **coercion**; **place** and **copy** rules are mapped to instances of the substitution principles; right rules are mapped to introduction rules; and left rules are mapped to elimination rules using the substitution principles when necessary. Note that the resulting derivation is *normal*, despite the use of the substitution principles, since we use it in the form of Lemma 4.

We show some representative cases.

case: $\dfrac{}{\Gamma; \cdot; A \Longrightarrow A}\,\textbf{init}$

Then

$$\begin{array}{ll} \Gamma; \cdot; A \vdash A \downarrow & \text{ohyp} \\ \Gamma; \cdot; A \vdash A \uparrow & \textbf{coercion} \end{array}$$

case: $\dfrac{\Gamma; \Delta_L \Delta_R; \Omega_L A \Omega_R \Longrightarrow B}{\Gamma; \Delta_L A \Delta_R; \Omega_L \Omega_R \Longrightarrow B}\,\textbf{place}$

Then

$$\begin{array}{ll} \Gamma; \Delta_L \Delta_R; \Omega_L A \Omega_R \vdash B \uparrow & \text{ind. hyp.} \\ \Gamma; A; \cdot \vdash A \downarrow & \text{lhyp} \\ \Gamma; \Delta_L A \Delta_R; \Omega_L \Omega_R \vdash B \uparrow & \text{lemma 4} \end{array}$$

case: $\dfrac{\Gamma; \Delta; \Omega A \Longrightarrow B}{\Gamma; \Delta; \Omega \Longrightarrow A \twoheadrightarrow B}\,{\twoheadrightarrow_R}$

Then

$$\begin{array}{ll} \Gamma; \Delta; \Omega A \vdash B \uparrow & \text{ind. hyp.} \\ \Gamma; \Delta; \Omega \vdash A \twoheadrightarrow B & \twoheadrightarrow_I \end{array}$$

case: $$\frac{\Gamma; \Delta; \Omega_L B \Omega_R \Longrightarrow C \qquad \Gamma; \Delta_A; \Omega_A \Longrightarrow A}{\Gamma; \Delta \bowtie \Delta_A; \Omega_L (A \twoheadrightarrow B) \Omega_A \Omega_R \Longrightarrow C} \twoheadrightarrow_L$$

Then

$$\begin{array}{ll}
\Gamma; \Delta; \Omega_L B \Omega_R \vdash C \uparrow & \text{ind. hyp.} \\
\Gamma; \Delta_A; \Omega_A \vdash A \uparrow & \text{ind. hyp.} \\
\Gamma; \cdot; A \twoheadrightarrow B \vdash A \twoheadrightarrow B \downarrow & \text{ohyp} \\
\Gamma; \Delta_A; (A \twoheadrightarrow B) \Omega_A \vdash B \downarrow & \twoheadrightarrow_E \\
\Gamma; \Delta \bowtie \Delta_A; \Omega_L (A \twoheadrightarrow B) \Omega_A \Omega_R \vdash C \uparrow & \text{lemma 4}
\end{array}$$

□

In the opposite direction we first need to generalize the induction hypothesis to make the proper statement about the atomic deduction—otherwise our induction would break down at the first coercion.

Theorem 11 *The following hold:*

1. $\Gamma; \Delta; \Omega \vdash A \uparrow$ *implies* $\Gamma; \Delta; \Omega \Longrightarrow A$
2. $\Gamma; \Delta; \Omega \vdash A \downarrow$ *and* $\Gamma; \Delta_C; \Omega_L A \Omega_R \Longrightarrow C$ *implies* $\Gamma; \Delta_C \bowtie \Delta; \Omega_L \Omega \Omega_R \Longrightarrow C$.

Proof: By structural induction on the given derivations. Instances of **coercion** translate to uses of the **init** rule from the result of the induction hypothesis. Introduction rules are mapped to right rules. Elimination rules are mapped to sequent derivations constructed from the corresponding left rule and the result of an appeal to the induction hypothesis.

We show some representative cases.

case: $$\frac{}{\Gamma; \cdot; A \vdash A \downarrow} \text{ohyp}$$

$$\begin{array}{ll}
\Gamma; \Delta_C; \Omega_L A \Omega_R \Longrightarrow C & \text{assumption}
\end{array}$$

case: $$\frac{}{\Gamma; A; \cdot \vdash A \downarrow} \text{lhyp}$$

Then

$$\begin{array}{ll}
\Gamma; \Delta_C; \Omega_L A \Omega_R \Longrightarrow C & \text{assumption} \\
\Gamma; \Delta_{CL} A \Delta_{CR}; \Omega_L \Omega_R \Longrightarrow C & \textbf{place} \\
\quad \text{where } \Delta_C = \Delta_{CL} \Delta_{CR} &
\end{array}$$

case: $$\frac{\Gamma;\Delta;\Omega \vdash A \twoheadrightarrow B\downarrow \qquad \Gamma;\Delta_A;\Omega_A \vdash A\uparrow}{\Gamma;\Delta \bowtie \Delta_A;\Omega\Omega_A \vdash B\downarrow}\twoheadrightarrow_E$$

Then

$$\begin{array}{ll}
\Gamma;\Delta_C;\Omega_L B\Omega_R \Longrightarrow C & \text{assumption}\\
\Gamma;\Delta_A;\Omega_A \Longrightarrow A & \text{ind. hyp.}\\
\Gamma;\Delta_A \bowtie \Delta_C;\Omega_L(A \twoheadrightarrow B)\Omega_A\Omega_R \Longrightarrow C & \twoheadrightarrow_L\\
\Gamma;\Delta \bowtie \Delta_A \bowtie \Delta_C;\Omega_L\Omega\Omega_A\Omega_R \Longrightarrow C & \text{ind. hyp.}
\end{array}$$

case: $$\frac{\Gamma;\Delta;\Omega \vdash A\downarrow}{\Gamma;\Delta;\Omega \vdash A\uparrow}\textbf{coercion}$$

Then

$$\begin{array}{ll}
\Gamma;\cdot;A \Longrightarrow A & \textbf{init}\\
\Gamma;\Delta;\Omega \Longrightarrow A & \text{ind. hyp.}
\end{array}$$

case: $$\frac{\Gamma;\Delta;\Omega A \vdash B\uparrow}{\Gamma;\Delta;\Omega \vdash A \twoheadrightarrow B\uparrow}\twoheadrightarrow_I$$

Then

$$\begin{array}{ll}
\Gamma;\Delta;\Omega A \Longrightarrow B & \text{ind. hyp.}\\
\Gamma;\Delta;\Omega \Longrightarrow A \twoheadrightarrow B & \twoheadrightarrow_R
\end{array}$$

□

We remark that the mapping just shown takes **init** sequents to **coercion** rules. This gives further credence to our intuition on the relation between normal deductions and cut-free proofs. The **coercion** rule represents the end of a normal proof from an operational viewpoint just as the **init** sequents represent the end of a sequent proof.

The proofs above are constructive and inherently contain a method for translation between sequent derivations in ordered linear logic and natural deductions. Although the correspondence is very close, it is not a bijection, because the order in which left rules are applied in a sequent derivation may be irrelevant to the resulting natural deduction. If one wants to establish a bijection, one has to further restrict the sequent rules. This has been investigated by Herbelin [25] for intuitionistic logic.

5.2 Directed Deductions and Sequents with Cut

We now show that this correspondence extends to directed natural deductions (using the $\vdash^{+}$ judgements) and sequents with explicit cut rules (using the $\stackrel{+}{\Longrightarrow}$ sequents). Specifically, coercing a normal derivation into an atomic derivation will correspond to using cut in the sequent calculus.

Theorem 12 $\Gamma;\Delta;\Omega \stackrel{+}{\Longrightarrow} A$ *implies* $\Gamma;\Delta;\Omega \vdash^{+} A\uparrow$

Proof: By induction on structure of the given derivation. The proof is exactly the same as the proof of theorem 10 with three additional cases. The cut rules are translated into a **lemma** rule followed by an appeal to the substitution principles.

We show the new cases.

case: $$\dfrac{\Gamma;\Delta_A;\Omega_A \stackrel{+}{\Longrightarrow} A \qquad \Gamma;\Delta;\Omega_L A \Omega_R \stackrel{+}{\Longrightarrow} C}{\Gamma;\Delta \bowtie \Delta_A;\Omega_L\Omega_A\Omega_R \stackrel{+}{\Longrightarrow} C}\,\mathrm{Cut}_{\Omega}$$

Then

$$\begin{array}{ll}
\Gamma;\Delta;\Omega_L A\Omega_R \vdash^{+} C\uparrow & \text{ind. hyp.}\\
\Gamma;\Delta_A;\Omega_A \vdash^{+} A\uparrow & \text{ind. hyp.}\\
\Gamma;\Delta_A;\Omega_A \vdash^{+} A\downarrow & \text{rule } \textbf{lemma}\\
\Gamma;\Delta \bowtie \Delta_A;\Omega_L\Omega_A\Omega_R \vdash^{+} C\uparrow & \text{lemma 4}
\end{array}$$

case: $$\dfrac{\Gamma;\Delta_A;\cdot \stackrel{+}{\Longrightarrow} A \qquad \Gamma;\Delta_L A\Delta_R;\Omega \stackrel{+}{\Longrightarrow} C}{\Gamma;\Delta_L \bowtie \Delta_A \bowtie \Delta_A;\Omega \stackrel{+}{\Longrightarrow} C}\,\mathrm{Cut}_{\Delta}$$

Then

$$\begin{array}{ll}
\Gamma;\Delta_L A\Delta_R;\Omega \vdash^{+} C\uparrow & \text{ind. hyp.}\\
\Gamma;\Delta_A;\cdot \vdash^{+} A\uparrow & \text{ind. hyp.}\\
\Gamma;\Delta_A;\cdot \vdash^{+} A\downarrow & \text{rule } \textbf{lemma}\\
\Gamma;\Delta_L \bowtie \Delta_A \bowtie \Delta_R;\Omega \vdash^{+} C\uparrow & \text{lemma 4}
\end{array}$$

case: $$\frac{\Gamma; \cdot; \cdot \overset{+}{\Longrightarrow} A \qquad \Gamma_L A \Gamma_R; \Delta; \Omega \overset{+}{\Longrightarrow} C}{\Gamma_L \Gamma \Gamma_R; \Delta; \Omega \overset{+}{\Longrightarrow} C}\,\mathrm{Cut}_\Gamma$$

Then

$$\begin{array}{ll}
\Gamma_L A \Gamma_R; \Delta; \Omega \vdash^+ C \uparrow & \text{ind. hyp.} \\
\Gamma_L \Gamma A \Gamma_R; \Delta; \Omega \vdash^+ C \uparrow & \text{lemma 3} \\
\Gamma; \cdot; \cdot \vdash^+ A \uparrow & \text{ind. hyp.} \\
\Gamma; \cdot; \cdot \vdash^+ A \downarrow & \text{rule } \textbf{lemma} \\
\Gamma_L \Gamma; \cdot; \cdot \vdash^+ A \downarrow & \text{lemma 3} \\
\Gamma_L \Gamma \Gamma_R; \Delta; \Omega \vdash^+ C \uparrow & \text{lemma 4}
\end{array}$$

□

Theorem 13 *The following hold:*

1. $\Gamma; \Delta; \Omega \vdash^+ A \uparrow$ *implies* $\Gamma; \Delta; \Omega \overset{+}{\Longrightarrow} A$
2. $\Gamma; \Delta; \Omega \vdash^+ A \downarrow$ *and* $\Gamma; \Delta_C; \Omega_L A \Omega_R \overset{+}{\Longrightarrow} C$ *implies* $\Gamma; \Delta_C \bowtie \Delta; \Omega_L \Omega \Omega_R \overset{+}{\Longrightarrow} C$

Proof: By induction on structure of the given derivations. The proof is exactly the same as the proof of theorem 11 with one additional case: from the **lemma** coercion we construct a use of the Cut_Ω rule.

case: $$\frac{\Gamma; \Delta; \Omega \vdash^+ A \uparrow}{\Gamma; \Delta; \Omega \vdash^+ A \downarrow}\,\textbf{lemma}$$

Then

$$\begin{array}{ll}
\Gamma; \Delta_C; \Omega_L A \Omega_R \overset{+}{\Longrightarrow} C & \text{assumption} \\
\Gamma; \Delta; \Omega \overset{+}{\Longrightarrow} A & \text{ind. hyp.} \\
\Gamma; \Delta \bowtie \Delta_C; \Omega_L \Omega \Omega_R \overset{+}{\Longrightarrow} C & \mathrm{Cut}_\Omega
\end{array}$$

□

5.3 Normalization

The previous observations give a syntactic proof of normalization of the natural deduction system.

Theorem 14 (Normalization) $\Gamma; \Delta; \Omega \vdash A$ *iff* $\Gamma; \Delta; \Omega \vdash A \uparrow$.

Proof: Given $\Gamma;\Delta;\Omega \vdash A$, we know $\Gamma;\Delta;\Omega \vdash^{+} A \uparrow$ from Theorem 6. Then $\Gamma;\Delta;\Omega \overset{+}{\Longrightarrow} A$ from Theorem 13. Then $\Gamma;\Delta;\Omega \Longrightarrow A$ from Theorem 9 (Cut Elimination). Then $\Gamma;\Delta;\Omega \ \vdash \ A \ \uparrow$ from Theorem 10. The other direction is the contents of Theorem 5. □

This concludes the basic development of ordered linear logic. We have constructed the logic from the basic notion of a hypothetical judgement and an analysis of the structural properties of hypotheses. We exhibited a natural deduction system for the logic and argued that the system is coherent by proving a normalization result. We also presented a sequent calculus for the logic, corresponding to the natural deduction system, and a cut-elimination result.

Part II

Ordered Linear Logic Programming

Chapter 6

Uniform Derivations

We now turn our attention to proof search in ordered linear logic (OLL). All of the analysis in this chapter will use the sequent calculus of Chapter 4 which is better suited, by allowing direct manipulation of hypotheses, to examining proof search than the natural deduction system.

Our analysis aims at achieving a logic programming language, where we view computation as the bottom-up construction of a derivation. The difficulty with the sequent system, as given, is that in any situation many left or right rules can be applied, leading to unacceptable non-determinism. To solve this problem, we design an alternative, more restricted system with the following properties (which are enforced *syntactically*):

- Derivations are *goal-directed* in that a derivation of a sequent with a non-atomic goal[1] always ends in a right rule. This allows us to view *logical connectives in goals as search instructions.*

- Derivations are *focussed* in that when deriving a sequent with an atomic goal we single out a particular hypothesis and apply a sequence of left rules until it is also atomic and immediately implies the goal. This allows us to view *atomic goals as procedure calls.*

In a minor departure from [37] we call derivations which are both goal-directed and focussed *uniform.*

[1] We refer to the succedent of a given sequent as the *goal.*

This chapter identifies the uniform fragment of ordered linear logic, the fragment of the logic for which uniform proofs are complete, and proves that this fragment may be thought of as an *abstract logic programming language* [37], *i.e.*, goal-directed and focussed proof search is complete for the fragment. In subsequent chapters we will further analyse the uniform fragment to obtain a concrete logic programming language. Furthermore, the uniform fragment is of additional interest since it exactly corresponds to the canonical fragment of an ordered lambda calculus which serves as the basis for a logical framework[2].

6.1 Uniform Fragment

In this section we identify the goal-directed fragment of ordered linear logic. As one can write a focussing derivation system for the entire logic[3], goal-directedness characterizes the uniform fragment. Clearly the entire logic is not goal-directed, consider the sequent

$$\cdot ; \cdot ; A \bullet B \longrightarrow A \bullet B$$

whose derivation is either an **init** rule (which is not goal-directed since the goal is non-atomic) or requires applying the $\bullet_L$ rule below the $\bullet_R$ rule.

As it turns out, the uniform fragment of OLL is:

$$\begin{array}{lrcll}
\textit{Uniform Formulas} & A & ::= & P & \text{atomic propositions} \\
 & & \mid & A \rightarrow B & \text{unrestricted implication} \\
 & & \mid & A \multimap B & \text{linear implication} \\
 & & \mid & A \twoheadrightarrow B & \text{ordered right implication} \\
 & & \mid & A \rightarrowtail B & \text{ordered left implication} \\
 & & \mid & A \mathbin{\&} B & \text{additive conjunction} \\
 & & \mid & \top & \text{additive unit} \\
 & & \mid & \forall x.\, A & \text{universal quantifier}
\end{array}$$

which is the uniform fragment of linear logic extended with ordered implications. That no other connectives are goal-directed may be seen from examining derivations of the form $\cdot ; \cdot ; C \longrightarrow C$ similarly to the previous example.

We now formally state and prove the goal-directedness property.

[2] see chapters 12 and 13

[3] The techniques in [3] for full linear logic are easily extended to OLL.

Lemma 15 (Goal-Directedness) *The following all hold:*

1. $\Gamma;\Delta;\Omega \Longrightarrow A \rightarrow B$ *implies* $\Gamma A;\Delta;\Omega \Longrightarrow B$
2. $\Gamma;\Delta;\Omega \Longrightarrow A \multimap B$ *implies* $\Gamma;\Delta A;\Omega \Longrightarrow B$
3. $\Gamma;\Delta;\Omega \Longrightarrow A \twoheadrightarrow B$ *implies* $\Gamma;\Delta;\Omega A \Longrightarrow B$
4. $\Gamma;\Delta;\Omega \Longrightarrow A \rightarrowtail B$ *implies* $\Gamma;\Delta;A\Omega \Longrightarrow B$
5. $\Gamma;\Delta;\Omega \Longrightarrow A \mathbin{\&} B$ *implies* $\Gamma;\Delta;\Omega \Longrightarrow A$ *and* $\Gamma;\Delta;\Omega \Longrightarrow B$
6. $\Gamma;\Delta;\Omega \Longrightarrow \forall x.\, A$ *implies* $\Gamma;\Delta;\Omega \Longrightarrow A[a/x]$ *where a is fresh.*

Proof: By using Theorem 8 (admissibility of the cut rules) we can use the given derivation to directly construct the result. We show a representative case.

case: $\overset{\mathcal{D}}{\Gamma;\Delta;\Omega \Longrightarrow A \twoheadrightarrow B}$

Then

$$\dfrac{\overset{\mathcal{D}}{\Gamma;\Delta;\Omega \Longrightarrow A \twoheadrightarrow B} \qquad \dfrac{\dfrac{}{\Gamma;\cdot;A \Longrightarrow A}\,\textbf{init} \qquad \dfrac{}{\Gamma;\cdot;B \Longrightarrow B}\,\textbf{init}}{\Gamma;\cdot;(A \twoheadrightarrow B)A \Longrightarrow B}\,{\twoheadrightarrow}_L}{\Gamma;\Delta;\Omega A \Longrightarrow B}\,\textbf{Cut}_\Omega$$

□

6.2 Uniform Derivation System

Uniform derivations may now be conceived as a focussing system for the uniform fragment which forces right rules to appear below left rules along the major branch of a proof.

We formalize our system with two judgements:

$\Gamma;\Delta;\Omega \longrightarrow A$	goal A is uniformly derivable
$\Gamma;\Delta;(\Omega_L;\Omega_R) \longrightarrow A \gg P$	hypothesis A immediately entails atomic goal P

where Γ, Δ and Ω are unrestricted, linear, and ordered hypotheses, respectively. In the latter judgment the ordered hypotheses are syntactically divided into a left part Ω_L and a right part Ω_R. It corresponds to the sequent

$$\Gamma; \Delta; (\Omega_L A \Omega_R) \implies P$$

so that the split in the ordered context tracks the location of the hypothesis we have focused on. This correspondence is stated formally in the soundness and completeness theorems for uniform derivations below.

All of the right rules are exactly the same as in the sequent calculus. Since no left rules apply when the goal is non-atomic, the derivation is completely determined by the structure of the goal, as desired.

$$\frac{\Gamma A; \Delta; \Omega \longrightarrow B}{\Gamma; \Delta; \Omega \longrightarrow A \rightarrow B}{\rightarrow_R} \qquad \frac{\Gamma; \Delta A; \Omega \longrightarrow B}{\Gamma; \Delta; \Omega \longrightarrow A \multimap B}{\multimap_R}$$

$$\frac{\Gamma; \Delta; \Omega A \longrightarrow B}{\Gamma; \Delta; \Omega \longrightarrow A \twoheadrightarrow B}{\twoheadrightarrow_R} \qquad \frac{\Gamma; \Delta; A \Omega \longrightarrow B}{\Gamma; \Delta; \Omega \longrightarrow A \rightarrowtail B}{\rightarrowtail_R}$$

$$\frac{\Gamma; \Delta; \Omega \longrightarrow A \qquad \Gamma; \Delta; \Omega \longrightarrow B}{\Gamma; \Delta; \Omega \longrightarrow A \,\&\, B}{\&_R} \qquad \frac{}{\Gamma; \Delta; \Omega \longrightarrow \top}{\top_R}$$

$$\frac{\Gamma; \Delta; \Omega \longrightarrow A[a/x]}{\Gamma; \Delta; \Omega \longrightarrow \forall x.\, A}{\forall_R^a}$$

When the goal has become atomic, we need to single out a hypothesis and determine if it immediately entails the goal. This is achieved by the three **choice** rules which apply to unrestricted, linear, or ordered hypotheses.

$$\frac{\Gamma_L A \Gamma_R; \Delta; (\Omega_L; \Omega_R) \longrightarrow A \gg P}{\Gamma_L A \Gamma_R; \Delta; \Omega_L \Omega_R \longrightarrow P}\,\textbf{choice}_\Gamma \qquad \frac{\Gamma; \Delta_L \Delta_R; (\Omega_L; \Omega_R) \longrightarrow A \gg P}{\Gamma; \Delta_L A \Delta_R; \Omega_L \Omega_R \longrightarrow P}\,\textbf{choice}_\Delta$$

$$\frac{\Gamma; \Delta; (\Omega_L; \Omega_R) \longrightarrow A \gg P}{\Gamma; \Delta; \Omega_L A \Omega_R \longrightarrow P}\,\textbf{choice}_\Omega$$

$\textbf{choice}_\Gamma$ is justified by **copy** in the sequent calculus, and $\textbf{choice}_\Delta$ by **place**. The premise and conclusion of $\textbf{choice}_\Omega$ correspond to identical sequents. An initial sequent corresponds to an immediate entailment between identical atomic formulas.

$$\frac{}{\Gamma; \cdot; (\cdot; \cdot) \longrightarrow P \gg P}\,\textbf{init}$$

The remaining left rules for immediate entailment directly correspond to the left sequent rules, keeping in mind that we have to consider the focussing formula as being between the left and right parts of the ordered context.

$$\dfrac{\Gamma; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; B \gg P \qquad \Gamma; \cdot; \cdot \;\longrightarrow\; A}{\Gamma; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; A \rightarrow B \gg P}\rightarrow_L$$

$$\dfrac{\Gamma; \Delta_B; (\Omega_L; \Omega_R) \;\longrightarrow\; B \gg P \qquad \Gamma; \Delta_A; \cdot \;\longrightarrow\; A}{\Gamma; \Delta_A \bowtie \Delta_B; (\Omega_L; \Omega_R) \;\longrightarrow\; A \multimap B \gg P}\multimap_L$$

$$\dfrac{\Gamma; \Delta_B; (\Omega_L; \Omega_R) \;\longrightarrow\; B \gg P \qquad \Gamma; \Delta_A; \Omega_A \;\longrightarrow\; A}{\Gamma; \Delta_A \bowtie \Delta_B; (\Omega_L; \Omega_A\Omega_R) \;\longrightarrow\; A \twoheadrightarrow B \gg P}\twoheadrightarrow_L$$

$$\dfrac{\Gamma; \Delta_B; (\Omega_L; \Omega_R) \;\longrightarrow\; B \gg P \qquad \Gamma; \Delta_A; \Omega_A \;\longrightarrow\; A}{\Gamma; \Delta_A \bowtie \Delta_B; (\Omega_L\Omega_A; \Omega_R) \;\longrightarrow\; A \rightarrowtail B \gg P}\rightarrowtail_L$$

$$\dfrac{\Gamma; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; A \gg P}{\Gamma; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; A \,\&\, B \gg P}\&_{L1} \qquad \dfrac{\Gamma; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; B \gg P}{\Gamma; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; A \,\&\, B \gg P}\&_{L2}$$

$$\dfrac{\Gamma; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; A[t/x] \gg P}{\Gamma; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; \forall x.\, A \gg P}\forall_L$$

In the uniform system, we rewrite our sample parsing proof from Chapter 4 as follows:

$$\Gamma = [(\mathtt{np} \twoheadrightarrow \mathtt{vp} \twoheadrightarrow \mathtt{snt})\,(\mathtt{tv} \twoheadrightarrow \mathtt{np} \twoheadrightarrow \mathtt{vp})\,(\mathtt{loves} \twoheadrightarrow \mathtt{tv})\,(\mathtt{mary} \twoheadrightarrow \mathtt{np})\,(\mathtt{bob} \twoheadrightarrow \mathtt{np})]$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma; \cdot; (\cdot; \cdot) \longrightarrow \mathtt{snt} \gg \mathtt{snt}}\,\mathbf{init} \qquad \begin{array}{c}\Theta\\ \Gamma; \cdot; \mathtt{loves\ bob} \longrightarrow \mathtt{vp}\end{array}}{\Gamma; \cdot; (\cdot; \mathtt{loves\ bob}) \longrightarrow (\mathtt{vp} \twoheadrightarrow \mathtt{snt}) \gg \mathtt{snt}}\twoheadrightarrow_L \qquad \begin{array}{c}\Theta_{\mathtt{mary}}\\ \Gamma; \cdot; \mathtt{mary} \;\longrightarrow\; \mathtt{np}\end{array}}{\Gamma; \cdot; (\cdot; \mathtt{mary\ loves\ bob}) \longrightarrow \mathtt{np} \twoheadrightarrow \mathtt{vp} \twoheadrightarrow \mathtt{snt} \gg \mathtt{snt}}\twoheadrightarrow_L}{\Gamma; \cdot; \mathtt{mary\ loves\ bob} \longrightarrow \mathtt{snt}}\,\mathbf{choice}_\Gamma$$

where $\Theta =$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma;\cdot;(\cdot;\cdot)\longrightarrow \mathtt{vp}\gg\mathtt{vp}}\,\mathbf{init}\qquad \begin{matrix}\Theta_{\mathtt{bob}}\\ \Gamma;\cdot;\mathtt{bob}\longrightarrow\mathtt{np}\end{matrix}}{\Gamma;\cdot;(\cdot;\mathtt{bob})\longrightarrow \mathtt{np}\twoheadrightarrow\mathtt{vp}\gg\mathtt{vp}}\,\twoheadrightarrow_L \qquad \begin{matrix}\Theta_{\mathtt{loves}}\\ \Gamma;\cdot;\mathtt{loves}\longrightarrow\mathtt{tv}\end{matrix}}{\Gamma;\cdot;(\cdot;\mathtt{loves\ bob})\longrightarrow \mathtt{tv}\twoheadrightarrow\mathtt{np}\twoheadrightarrow\mathtt{vp}\gg\mathtt{vp}}\,\twoheadrightarrow_L}{\Gamma;\cdot;\mathtt{loves\ bob}\longrightarrow\mathtt{vp}}\,\mathbf{choice}_\Gamma$$

where $\Theta_{\mathtt{bob}} =$

$$\dfrac{\dfrac{\dfrac{}{\Gamma;\cdot;(\cdot;\cdot)\longrightarrow \mathtt{np}\gg\mathtt{np}}\,\mathbf{init}\qquad \dfrac{\dfrac{}{\Gamma;\cdot;(\cdot;\cdot)\longrightarrow \mathtt{bob}\gg\mathtt{bob}}\,\mathbf{init}}{\Gamma;\cdot;\mathtt{bob}\longrightarrow\mathtt{bob}}\,\mathbf{choice}_\Omega}{\Gamma;\cdot;(\cdot;\mathtt{bob})\longrightarrow \mathtt{bob}\twoheadrightarrow\mathtt{np}\gg\mathtt{np}}\,\twoheadrightarrow_L}{\Gamma;\cdot;\mathtt{bob}\longrightarrow\mathtt{np}}\,\mathbf{choice}_\Gamma$$

and $\Theta_{\mathtt{mary}}, \Theta_{\mathtt{loves}}$ are similar.

Unlike the example given in Chapter 4, this is the only proof of the end-sequent. The first choice is forced since $\mathtt{np}\twoheadrightarrow\mathtt{vp}\twoheadrightarrow\mathtt{snt}$ is the only formula in Γ whose head, $\mathtt{snt}$, matches the goal. The same is true for all the other choices made.

We now show that uniform derivations are sound and complete with respect to the sequent calculus. The soundness result is easy to show.

Theorem 16 (Soundness of Uniform Derivations)

1. *If* $\Gamma;\Delta;\Omega \longrightarrow A$ *then* $\Gamma;\Delta;\Omega \Longrightarrow A$.
2. *If* $\Gamma;\Delta;(\Omega_L;\Omega_R) \longrightarrow A \gg P$ *then* $\Gamma;\Delta;\Omega_L A \Omega_R \Longrightarrow P$.

Proof: By mutual structural induction on the derivations of the given judgements using the correspondences mentioned throughout this section. □

The completeness result is harder, but largely follows techniques of [3] and [37], adapted to the ordered case. The main difficulty arises from the mismatch between the left rules of the uniform system and the original sequent system. Therefore we will need to show how to construct a uniform derivation whose premises and conclusion match those of the sequent system's left rules. To be concrete, suppose we have derivations of

$$\Gamma; \Delta; \Omega_L B \Omega_R \longrightarrow C \quad \text{and} \quad \Gamma; \Delta_A; \Omega_A \longrightarrow A$$

then we need to show that we can construct a derivation of

$$\Gamma; \Delta \bowtie \Delta_A; \Omega_L(A \twoheadrightarrow B)\Omega_A \Omega_R \longrightarrow C$$

in order to show completeness of the uniform derivation system.

We will have to form so called "inversion" principles for each connective in the uniform system. The proofs for these inversion principles are non-trivial. Lemma 17, and its proof, presents a detailed proof of the inversion principle for $\rightarrowtail$. Lemma 18 then states all of the necessary inversion principles, each of which may be proven similarly to Lemma 17.

Lemma 17

1. $\Gamma; \Delta; \Omega_L B \Omega_R \longrightarrow C$ *and* $\Gamma; \Delta_A; \Omega_A \longrightarrow A$
   implies $\Gamma; \Delta \bowtie \Delta_A; \Omega_L \Omega_A (A \rightarrowtail B)\Omega_R \longrightarrow P.$
2. $\Gamma; \Delta; (\Omega_{LL} B \Omega_{LR}; \Omega_R) \longrightarrow C \gg P$ *and* $\Gamma; \Delta_A; \Omega_A \longrightarrow A$
   implies $\Gamma; \Delta \bowtie \Delta_A; (\Omega_{LL}\Omega_A(A \rightarrowtail B)\Omega_{LR}; \Omega_R) \longrightarrow C \gg P.$
3. $\Gamma; \Delta; (\Omega_L; \Omega_{RL} B \Omega_{RR}) \longrightarrow C \gg P$ *and* $\Gamma; \Delta_A; \Omega_A \longrightarrow A$
   implies $\Gamma; \Delta \bowtie \Delta_A; (\Omega_L; \Omega_{RL}\Omega_A(A \rightarrowtail B)\Omega_{RR}) \longrightarrow C \gg P.$

Proof: By mutual induction on the structure of the given derivations.
Assume $\Gamma; \Delta_A; \Omega_A \longrightarrow A$.

part 1:

Cases when C is atomic (*i.e.* $C = P$).

Assume $\overset{\Pi}{\Gamma; \Delta; \Omega_L B \Omega_R \longrightarrow P}$. Then there are 7 possibilities for Π:

case 1: Π ends with $\mathbf{choice}_\Omega$ and $\Gamma; \Delta; (\Omega_L; \Omega_R) \longrightarrow B \gg P$

Then

$$\begin{array}{ll} \Gamma; \Delta \bowtie \Delta_A; (\Omega_L \Omega_A; \Omega_R) \longrightarrow A \rightarrowtail B \gg P & \rightarrowtail_L \\ \Gamma; \Delta \bowtie \Delta_A; \Omega_L \Omega_A (A \rightarrowtail B)\Omega_R \longrightarrow P & \mathbf{choice}_\Omega \end{array}$$

case 2: Π ends with $\mathbf{choice}_\Omega$ and $\Gamma; \Delta; (\Omega_{LL}; \Omega_{LR} B \Omega_R) \longrightarrow C \gg P$ where $\Omega_L = \Omega_{LL} C \Omega_{LR}$.
Then

$$\begin{array}{ll}\Gamma; \Delta \bowtie \Delta_A; (\Omega_{LL}; \Omega_{LR}\Omega_A(A \rightarrowtail B)\Omega_R) \longrightarrow C \gg P & \text{ind. hyp.} \\ \Gamma; \Delta \bowtie \Delta_A; \Omega_{LL} C \Omega_{LR}\Omega_A(A \rightarrowtail B)\Omega_R \longrightarrow P & \mathbf{choice}_\Omega\end{array}$$

case 3: Π ends with $\mathbf{choice}_\Omega$ and $\Gamma; \Delta; (\Omega_L B \Omega_{RL}; \Omega_{RR}) \longrightarrow C \gg P$ where $\Omega_R = \Omega_{RL} C \Omega_{RR}$.
Then

$$\begin{array}{ll}\Gamma; \Delta \bowtie \Delta_A; (\Omega_L\Omega_A(A \rightarrowtail B)\Omega_{RL}; \Omega_{RR}) \longrightarrow C \gg P & \text{ind. hyp.} \\ \Omega_L\Omega_A(A \rightarrowtail B)\Omega_{RL} C \Omega_{RR} \longrightarrow P & \mathbf{choice}_\Omega\end{array}$$

cases 4,5,6,7 : Π ends with $\mathbf{choice}_\Delta$ (2 cases) or $\mathbf{choice}_\Gamma$ (2 cases).
Similar to previous two cases.

Cases when C is non-atomic.

case: $$\frac{\Gamma; \Delta; \Omega_L B \Omega_R C_1 \longrightarrow C_2}{\Gamma; \Delta; \Omega_L B \Omega_R \longrightarrow C_1 \twoheadrightarrow C_2} \twoheadrightarrow_R$$

Then

$$\begin{array}{ll}\Gamma; \Delta \bowtie \Delta_A; \Omega_L\Omega_A(A \rightarrowtail B)\Omega_R C_1 \longrightarrow C_2 & \text{ind. hyp.} \\ \Gamma; \Delta \bowtie \Delta_A; \Omega_L\Omega_A(A \rightarrowtail B)\Omega_R \longrightarrow C_1 \twoheadrightarrow C_2 & \twoheadrightarrow_R\end{array}$$

cases for other connectives are similar.

part 2: Assume $\Gamma; \Delta; (\Omega_{LL} B \Omega_{LR}; \Omega_R) \longrightarrow C \gg P$. Note that C cannot be atomic since the ordered context is not empty.

case: $C = C_1 \rightarrowtail C_2$ and $\Delta = \Delta_2 \bowtie \Delta_1$ and $\Omega_{LR} = \Omega_{LRL}\Omega_{LRR}$ and $\Gamma; \Delta_2; (\Omega_{LL} B \Omega_{LRL}; \Omega_R) \longrightarrow C_2 \gg P$ and $\Gamma; \Delta_1; \Omega_{LRR} \longrightarrow C_1$.
Then

$$\begin{array}{ll}\Gamma; \Delta_1 \bowtie \Delta_A; (\Omega_{LL}\Omega_A(A \rightarrowtail B)\Omega_{LRL}; \Omega_R) \longrightarrow C_2 \gg P & \text{ind. hyp.} \\ \Gamma; \Delta \bowtie \Delta_A; (\Omega_{LL}\Omega_A(A \rightarrowtail B)\Omega_{LRL}\Omega_{LRR}; \Omega_R) \longrightarrow C_1 \rightarrowtail C_2 \gg P & \rightarrowtail_L\end{array}$$

case: $C = C_1 \rightarrowtail C_2$ and $\Delta = \Delta_2 \bowtie \Delta_1$ and $\Omega_{LL} = \Omega_{LLL}\Omega_{LLR}$ and $\Gamma; \Delta_2; (\Omega_{LLL}; \Omega_R) \longrightarrow C_2 \gg P$ and $\Gamma; \Delta_1; \Omega_{LLR} B \Omega_{LR} \longrightarrow C_1$.
Then

$$\Gamma; \Delta_1 \bowtie \Delta_A; \Omega_{LLR}\Omega_A(A \rightarrowtail B)\Omega_{LR} \longrightarrow C_1 \qquad \text{ind. hyp.}$$
$$\Gamma; \Delta \bowtie \Delta_A; (\Omega_{LLL}\Omega_{LLR}\Omega_A(A \rightarrowtail B)\Omega_{LR}; \Omega_R) \longrightarrow C_1 \rightarrowtail C_2 \gg P \qquad {\rightarrowtail}_L$$

case: $C = C_1 \twoheadrightarrow C_2$ and $\Delta = \Delta_2 \bowtie \Delta_1$ and $\Omega_R = \Omega_{RL}\Omega_{RR}$ and $\Gamma; \Delta_2; (\Omega_{LL}B\Omega_{LR}; \Omega_{RR}) \longrightarrow C_2 \gg P$ and $\Gamma; \Delta_1; \Omega_{RL} \longrightarrow C_1$.
Then

$$\Gamma; \Delta_2 \bowtie \Delta_A; (\Omega_{LL}\Omega_A(A \rightarrowtail B)\Omega_{LR}; \Omega_{RR}) \longrightarrow C_2 \gg P \qquad \text{ind. hyp.}$$
$$\Gamma; \Delta \bowtie \Delta_A; (\Omega_{LL}\Omega_A(A \rightarrowtail B)\Omega_{LR}; \Omega_{RL}\Omega_{RR}) \longrightarrow C_1 \twoheadrightarrow C_2 \gg P \qquad {\twoheadrightarrow}_L$$

cases: $C = C_1 \multimap C_2$, $C = C_1 \rightarrow C_2$, $C = C_1 \,\&\, C_2$, $C = \forall x.\, C'$
Similar to previous cases.

part 3: Assume $\Gamma; \Delta; (\Omega_L; \Omega_{RL}B\Omega_{RR}) \longrightarrow C \gg P$.
Then can be proven with reasoning symmetric to part 2.

□

We now state all the inversion principles needed to prove completeness.

Lemma 18 (Inversion) *The following all hold:*

1. $\Gamma; \Delta; \Omega_L B \Omega_R \longrightarrow C$ *and* $\Gamma; \Delta_A; \Omega_A \longrightarrow A$ *implies*
$\Gamma; \Delta \bowtie \Delta_A; \Omega_L\Omega_A(A \rightarrowtail B)\Omega_R \longrightarrow C$.

2. $\Gamma; \Delta; \Omega_L B \Omega_R \longrightarrow C$ *and* $\Gamma; \Delta_A; \Omega_A \longrightarrow A$ *implies*
$\Gamma; \Delta \bowtie \Delta_A; \Omega_L\Omega_A(A \twoheadrightarrow B)\Omega_R \longrightarrow C$.

3. $\Gamma; \Delta; \Omega_L B \Omega_R \longrightarrow C$ *and* $\Gamma; \Delta_A; \cdot \longrightarrow A$ *implies*
$\Gamma; \Delta \bowtie \Delta_A; \Omega_L(A \multimap B)\Omega_R \longrightarrow C$.

4. $\Gamma; \Delta; \Omega_L B \Omega_R \longrightarrow C$ *and* $\Gamma; \cdot; \cdot \longrightarrow A$ *implies*
$\Gamma; \Delta; \Omega_L(A \rightarrow B)\Omega_R \longrightarrow C$.

5. $\Gamma; \Delta; \Omega_L A \Omega_R \longrightarrow C$ *implies* $\Gamma; \Delta; \Omega_L(A \,\&\, B)\Omega_R \longrightarrow C$

6. $\Gamma; \Delta; \Omega_L B \Omega_R \longrightarrow C$ *implies* $\Gamma; \Delta; \Omega_L(A \,\&\, B)\Omega_R \longrightarrow C$

7. $\Gamma; \Delta; \Omega_L A[t/x]\Omega_R \longrightarrow C$ *implies* $\Gamma; \Delta; \Omega_L(\forall x.\, A)\Omega_R \longrightarrow C$

8. $\Gamma; \Delta_L\Delta_R; \Omega_L A \Omega_R \longrightarrow C$ *implies* $\Gamma; \Delta_L A \Delta_R; \Omega_L\Omega_R \longrightarrow C$.

9. $\Gamma_L A \Gamma_R; \Delta_L A \Delta_R; \Omega \longrightarrow C$ *implies* $\Gamma_L A \Gamma_R; \Delta_L\Delta_R; \Omega \longrightarrow C$.

Proof: Part 1 is immediate from the previous lemma. The other parts are similarly proved. □

We may now easily prove the completeness result.

Theorem 19 (Completeness of Uniform Derivations) *:*
$\Gamma; \Delta; \Omega \Longrightarrow A$ *implies* $\Gamma; \Delta; \Omega \longrightarrow A$.

Proof: By induction on the structure of the given derivation:

case: $\dfrac{}{\Gamma; \cdot; P \Longrightarrow P}\,\mathbf{init}$ Then $\dfrac{\dfrac{}{\Gamma; \cdot; (\cdot; \cdot) \longrightarrow P \gg P}\,\mathbf{init}}{\Gamma; \cdot; P \longrightarrow P}\,\mathbf{choice}_\Omega$

case: $\dfrac{\Gamma; \Delta; A\Omega \Longrightarrow B}{\Gamma; \Delta; \Omega \Longrightarrow A \rightarrowtail B}\,{\rightarrowtail_R}$

Then

$\Gamma; \Delta; A\Omega \longrightarrow B$	ind. hyp.
$\Gamma; \Delta; \Omega \longrightarrow A \rightarrowtail B$	$\rightarrowtail_R$

cases: $\twoheadrightarrow_R$,$\multimap_R$,$\rightarrow_R$, $\&_R$, $\forall_R$ all similar to previous case.

case: $\dfrac{\Gamma; \Delta_B; \Omega_L B \Omega_R \Longrightarrow C \qquad \Gamma; \Delta_A; \Omega_A \Longrightarrow A}{\Gamma; \Delta_B \bowtie \Delta_A; \Omega_L\Omega_A(A \rightarrowtail B)\Omega_R \Longrightarrow C}\,{\rightarrowtail_L}$

Then

$\Gamma; \Delta_B; \Omega_L B \Omega_R \longrightarrow C$ and $\Gamma; \Delta_A; \Omega_A \longrightarrow A$	ind. hyp.
$\Gamma; \Delta_B \bowtie \Delta_A; \Omega_L\Omega_A(A \rightarrowtail B)\Omega_R \longrightarrow C$	lemma 18

cases: $\twoheadrightarrow_L$,$\multimap_L$,$\rightarrow_L$,$\&_{Li}$,$\forall_L$,**copy**,**place** all similar to previous case.

□

We have now shown that ordered linear logic qualifies as an abstract logic programming language in the sense of [37]. However, directly coding up the uniform derivation system will result in hopelessly slow code due to the non-determinism still present in the inference rules. Even though the derivation system is goal-directed, many unspecified choices will be encountered during a bottom-up proof search.

A major source of non-determinism involves choosing which formula to focus on, and occurs in rules: $\mathbf{choice}_x$, and $\&_{Lx}$. It is impossible to completely eliminate this type of non-determinism (at least while staying within the bounds of logic programming). Most of the remaining non-determinism can be effectively removed from the system using standard techniques from logic programming. We can fix an order for proving each premise of of a multi-premise rule. We can remove existential choices, the need to pick a term in the $\forall_L$ rule, by introducing logic variables and unification.

After taking the above measures, the only non-determinism yet to consider comes from the need to split[4] the linear and ordered contexts, and occurs in rules: $\mathbf{choice}_\Gamma$, $\mathbf{choice}_\Delta$, $\twoheadrightarrow_L$, $\rightarrowtail_L$, and $\multimap_L$. The non-determinism of the $\twoheadrightarrow_L$, $\rightarrowtail_L$ and $\multimap_L$ rules can be handled with a non-trivial extension of the input-ouput model used by Lolli and LLF for linear logic; Chapter 8 provides the details. However, the non-deterministic context splits in the $\mathbf{choice}_\Gamma$ and $\mathbf{choice}_\Delta$ rules are not amenable to the same treatment. Therefore, we first explain, in Chapter 7, how to transform this non-determinism into a form which can be treated with the techniques of Chapter 8.

[4] Remember we are considering bottom-up proof search.

Chapter 7

Residuation

As mentioned in the previous chapter, the non-deterministic context splits in the $\mathbf{choice}_\Gamma$ and $\mathbf{choice}_\Delta$ rules is different than that in the $\twoheadrightarrow_L$, $\rightarrowtail_L$ and $\multimap_L$ rules. The former rules simply pick an arbitrary point at which the ordered context is to be split; while the latter rules actually divide the ordered (and linear) context between two premises. Note that the former context splitting arises from the need to fix the position of a non-ordered hypothesis in the ordered context.

It turns out that the sub-goals of a non-ordered formula place constraints upon where the formula must be located in the ordered context. Consider the following derivable sequent:

$$\cdot; A \twoheadrightarrow B \rightarrowtail C; BA \longrightarrow C$$

The $\twoheadrightarrow$ requires that A be to the right of $A \twoheadrightarrow B \rightarrowtail C$; similarly B must be to its left. Thus a derivation of this sequent requires $A \twoheadrightarrow B \rightarrowtail C$ to be placed in between the A and B in the ordered context. Using these observations, we may constrain the position of a non-ordered focus formula while solving its subgoals.

This chapter shows exactly how this process works. We first expand the formula language to allow restricted occurrences of the non-uniform connectives which maintain goal-directedness. We then use this expanded formula language to remove the non-deterministic context splits in the $\mathbf{choice}_\Gamma$ and $\mathbf{choice}_\Delta$ rules.

7.1 Extended Uniform Derivations

As mentioned in section 6.1, the connectives outside the uniform fragment are not goal-directed. However, we can allow restricted occurrences of non-uniform connectives, without compromising uniformity. Specifically, we can separate formulas into goal formulas, which may appear on the right of a sequent (*i.e.*, positively), and clause formulas which may appear to the left of a sequent (*i.e.*, negatively). It turns out that allowing positive occurrences, and forbidding negative occurences, of non-uniform connectives does not affect goal-directedness.

Clause Formulas	D	$::=$	P	atomic propositions
		$\mid$	$G \to D$	unrestricted implication
		$\mid$	$G \multimap D$	linear implication
		$\mid$	$G \twoheadrightarrow D$	ordered right implication
		$\mid$	$G \rightarrowtail D$	ordered left implication
		$\mid$	$D_1 \& D_2$	additive conjunction
		$\mid$	$\top$	additive unit
		$\mid$	$\forall x.\, D$	universal quantifier

Goal Formulas	G	$::=$	P	atomic propositions
		$\mid$	$D \to G$	unrestricted implication
		$\mid$	$!G$	unrestricted modality
		$\mid$	$D \multimap G$	unrestricted implication
		$\mid$	$¡G$	linear modality
		$\mid$	$D \twoheadrightarrow G$	ordered right implication
		$\mid$	$D \rightarrowtail G$	ordered left implication
		$\mid$	$G_1 \bullet G_2$	right multiplicative conjuction
		$\mid$	$G_1 \circ G_2$	left multiplicative conjuction
		$\mid$	$\mathbf{1}$	multiplicative unit
		$\mid$	$G_1 \& G_2$	additive conjunction
		$\mid$	$\top$	additive unit
		$\mid$	$G_1 \oplus G_2$	multiplicative disjunction
		$\mid$	$\mathbf{0}$	multiplicative falsehood
		$\mid$	$\forall x.\, G$	universal quantifier
		$\mid$	$\exists x.\, G$	existential quantifier

For the rest of this chapter, and the subsequent ones dealing with logic programming, we shall take care to maintain the distinction between clause formulas, G, and goal formulas, D. We now explicitly show the complete extended uniform derivation rules.

We have two types of mutually dependent sequents to capture uniform (i.e. focussing and goal-directed) derivations:

$$\begin{array}{lll}
\Gamma;\Delta;\Omega \;\longrightarrow\; G & \quad & \text{uniform derivability} \\
\Gamma;\Delta;(\Omega_L;\Omega_R) \;\longrightarrow\; D \gg P & \quad & \text{immediate entailment}
\end{array}$$

where Γ,Δ,Ω,Ω_L, and Ω_R are lists of clause formulas.

Here are the inference rules for extended uniform derivations:

$$\frac{}{\Gamma;\cdot;\cdot \;\longrightarrow\; \mathbf{1}}\,\mathbf{1}_R \qquad \frac{\Gamma;\cdot;\cdot \;\longrightarrow\; G}{\Gamma;\cdot;\cdot \;\longrightarrow\; !G}\,!_R \qquad \frac{\Gamma;\Delta;\cdot \;\longrightarrow\; G}{\Gamma;\Delta;\cdot \;\longrightarrow\; \text{¡}G}\,\text{¡}_R$$

$$\frac{\Gamma D;\Delta;\Omega \;\longrightarrow\; G}{\Gamma;\Delta;\Omega \;\longrightarrow\; D \rightarrow G}\,\rightarrow_R \qquad \frac{\Gamma;\Delta D;\Omega \;\longrightarrow\; G}{\Gamma;\Delta;\Omega \;\longrightarrow\; D \multimap G}\,\multimap_R$$

$$\frac{\Gamma;\Delta;\Omega D \;\longrightarrow\; G}{\Gamma;\Delta;\Omega \;\longrightarrow\; D \twoheadrightarrow G}\,\twoheadrightarrow_R \qquad \frac{\Gamma;\Delta;D\Omega \;\longrightarrow\; G}{\Gamma;\Delta;\Omega \;\longrightarrow\; D \rightarrowtail G}\,\rightarrowtail_R$$

$$\frac{\Gamma;\Delta_1;\Omega_1 \;\longrightarrow\; G_1 \qquad \Gamma;\Delta_2;\Omega_2 \;\longrightarrow\; G_2}{\Gamma;\Delta_1 \bowtie \Delta_2;\Omega_1\Omega_2 \;\longrightarrow\; G_1 \bullet G_2}\,\bullet_R \qquad \frac{\Gamma;\Delta_1;\Omega_1 \;\longrightarrow\; G_1 \qquad \Gamma;\Delta_2;\Omega_2 \;\longrightarrow\; G_2}{\Gamma;\Delta_1 \bowtie \Delta_2;\Omega_2\Omega_1 \;\longrightarrow\; G_1 \circ G_2}\,\circ_R$$

$$\frac{\Gamma;\Delta;\Omega \;\longrightarrow\; G_1}{\Gamma;\Delta;\Omega \;\longrightarrow\; G_1 \oplus G_2}\,\oplus_{R1} \qquad \frac{\Gamma;\Delta;\Omega \;\longrightarrow\; G_2}{\Gamma;\Delta;\Omega \;\longrightarrow\; G_1 \oplus G_2}\,\oplus_{R2} \qquad \text{no } \mathbf{0}_R \text{ rule}$$

$$\frac{\Gamma;\Delta;\Omega \;\longrightarrow\; G_1 \qquad \Gamma;\Delta;\Omega \;\longrightarrow\; G_2}{\Gamma;\Delta;\Omega \;\longrightarrow\; G_1 \,\&\, G_2}\,\&_R \qquad \frac{}{\Gamma;\Delta;\Omega \;\longrightarrow\; \top}\,\top_R$$

$$\frac{\Gamma;\Delta;\Omega \;\longrightarrow\; G[g/x]}{\Gamma;\Delta;\Omega \;\longrightarrow\; \forall x.\, G}\,\forall_R^a \qquad \frac{\Gamma;\Delta;\Omega \;\longrightarrow\; G[t/x]}{\Gamma;\Delta;\Omega \;\longrightarrow\; \exists x.\, G}\,\exists_R$$

(a not free in conclusion)

$$\frac{\Gamma_L D \Gamma_R; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; D \gg P}{\Gamma_L D \Gamma_R; \Delta; \Omega_L \Omega_R \;\longrightarrow\; P}\,\textbf{choice}_\Gamma \qquad \frac{\Gamma; \Delta_L \Delta_R; (\Omega_L; \Omega_R) \;\longrightarrow\; D \gg P}{\Gamma; \Delta_L D \Delta_R; \Omega_L \Omega_R \;\longrightarrow\; P}\,\textbf{choice}_\Delta$$

$$\frac{\Gamma; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; D \gg P}{\Gamma; \Delta; \Omega_L D \Omega_R \;\longrightarrow\; P}\,\textbf{choice}_\Omega$$

$$\frac{}{\Gamma; \cdot; (\cdot; \cdot) \;\longrightarrow\; P \gg P}\,\textbf{init}$$

$$\frac{\Gamma; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; D_1 \gg P}{\Gamma; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; D_1 \,\&\, D_2 \gg P}\,\&_{L1} \qquad \frac{\Gamma; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; D_2 \gg P}{\Gamma; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; D_1 \,\&\, D_2 \gg P}\,\&_{L1}$$

$$\frac{\Gamma; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; D[t/x] \gg P}{\Gamma; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; \forall x.\, D \gg P}\,\forall_L$$

$$\frac{\Gamma; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; D \gg P \qquad \Gamma; \cdot; \cdot \;\longrightarrow\; G}{\Gamma; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; G \rightarrow D \gg P}\,\rightarrow_L$$

$$\frac{\Gamma; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; D \gg P \qquad \Gamma; \Delta_G; \cdot \;\longrightarrow\; G}{\Gamma; \Delta \bowtie \Delta_G; (\Omega_L; \Omega_R) \;\longrightarrow\; G \multimap D \gg P}\,\multimap_L$$

$$\frac{\Gamma; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; D \gg P \qquad \Gamma; \Delta_G; \Omega_G \;\longrightarrow\; G}{\Gamma; \Delta \bowtie \Delta_G; (\Omega_L; \Omega_G \Omega_R) \;\longrightarrow\; G \twoheadrightarrow D \gg P}\,\twoheadrightarrow_L$$

$$\frac{\Gamma; \Delta; (\Omega_L; \Omega_R) \;\longrightarrow\; D \gg P \qquad \Gamma; \Delta_G; \Omega_G \;\longrightarrow\; G}{\Gamma; \Delta \bowtie \Delta_G; (\Omega_L \Omega_G; \Omega_R) \;\longrightarrow\; G \rightarrowtail D \gg P}\,\rightarrowtail_L$$

We can now state the correctness of this system.

Theorem 20 (Soundness of Extended Uniform Derivations)

1. $\Gamma; \Delta; \Omega \;\longrightarrow\; G$ *implies* $\Gamma; \Delta; \Omega \;\Longrightarrow\; G.$

2. $\Gamma;\Delta;(\Omega_L;\Omega_R) \longrightarrow G \gg P$ *implies* $\Gamma;\Delta;\Omega_L G \Omega_R \Longrightarrow P$.

where only clause formulas occur in each context.

Proof: By mutual structural induction on the derivations of the given judgements. □

Theorem 21 (Completeness of Uniform Derivations)

$\Gamma;\Delta;\Omega \Longrightarrow G$ *implies* $\Gamma;\Delta;\Omega \longrightarrow G$
where only clause formulas occur in each context.

Proof: We may use the proof of theorem 19. We need only add cases for the non-uniform connectives; each of which is simply an appeal to the induction hypothesis since non-uniform connectives only appear positively. □

From this point forward, we will assume that all contexts only contain clause formulas.

7.2 Residuation of Mobile Hypotheses

Recall from chapter 6 our interpretation of goals as search instructions. Following this lead, we may interpret clause formulas as compound search instructions, since all of the subgoals must be solved in order to successfully use a hypothesis. To better illustrate, consider:

$$\cdot;P_1 \twoheadrightarrow P_2 \rightarrowtail P;P_2P_1 \longrightarrow P$$

whose derivation is:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot;\cdot;(\cdot;\cdot) \longrightarrow P \gg P}\,\mathbf{init} \qquad \begin{matrix}\vdots\\ \cdot;\cdot;P_2 \longrightarrow P_2\end{matrix}}{\cdot;\cdot;(P_2;\cdot) \longrightarrow P_2 \rightarrowtail P \gg P}\,{\rightarrowtail_L} \qquad \begin{matrix}\vdots\\ \cdot;\cdot;P_1 \longrightarrow P_1\end{matrix}}{\cdot;\cdot;(P_2;P_1) \longrightarrow P_1 \twoheadrightarrow P_2 \rightarrowtail P \gg P}\,{\twoheadrightarrow_L}}{\cdot;P_1 \twoheadrightarrow P_2 \rightarrowtail P;P_2P_1 \longrightarrow P}\,\mathbf{choice}_{\Delta}$$

Both P_1 and P_2 must be derived in order to successfully use $P_1 \twoheadrightarrow P_2 \rightarrowtail P$. Further more, we know, from the $\twoheadrightarrow$ and $\rightarrowtail$ that the ordered hypotheses used to prove P_2

must be to the left of those used to solve P_1. But this is exactly what the connective $\circ$ requires. Thus we could think about changing the above derivation into something like the following:

$$\dfrac{\dfrac{\begin{matrix}\vdots & \qquad & \vdots\\ \cdot;\cdot;P_2 \longrightarrow P_2 & \qquad & \cdot;\cdot;P_1 \longrightarrow P_1\end{matrix}}{\cdot;\cdot;P_2P_1 \longrightarrow P_1 \circ P_2}\circ_R}{\cdot;P_1 \twoheadrightarrow P_2 \rightarrowtail P;P_2P_1 \longrightarrow P}\,\textbf{choice}'_{\Delta}$$

where we have transformed the two subgoals, P_1 and P_2, into a new compound subgoal, $P_1 \circ P_2$. We call this sort of transformation *residuation*.

It turns out that *mobile* hypotheses, hypotheses in the unrestricted and linear contexts, can be residuated (logically compiled) into goal formulas (with some minor extensions) following the general development in [9]. This observation will allow us to focus on unrestricted formulas without having to split the ordered context. Rather, the splitting will be delayed and taken care of by the multiplicative conjunction rules, which can be handled similarly to the $\rightarrowtail_L$, $\twoheadrightarrow_L$, and $\multimap_L$ rules as explained in chapter 8.

First we extend goal formulas with an atomic equality:

$$\textit{Goal Formulas} \quad G \quad ::= \quad \ldots \quad \mid P_1 \doteq P_2 \quad \text{Equality}$$

then add a new inference rule:

$$\dfrac{}{\Gamma;\cdot;\cdot \xrightarrow{r} P \doteq P}\doteq_R$$

We need only add a right rule for this new connective since it is a goal formula.

We now present a residuation derivation system, which is sound and complete wrt uniform derivations, in which the ordered context need not be split when focusing on an unrestricted formula. This derivation system will consist of two types of sequent

$$\begin{array}{l}\Gamma;\Delta;\Omega \xrightarrow{r} G\\ \Gamma;\Delta;(\Omega_L;\Omega_R) \xrightarrow{r} D \gg P\end{array}$$

which exactly match their uniform counterparts, except as noted below, plus the new judgement

$$G';D \gg P \setminus G$$

which shall be used to residuate clause formulas into goal formulas. In this new judgement, G', D, and P are considered input and G is considered the output; specifically, G' is an accumulating result (goal formula), D is the formula being residuated, P is the atom which must match the head[1] of D, and G is the resulting goal formula.

We use the following inference rules for the new judgements:

$$\frac{}{G; P' \gg P \setminus (P' \doteq P) \bullet G} \qquad \frac{}{G; \top \gg P \setminus \mathbf{0}}$$

$$\frac{G'; D_1 \gg P \setminus G_1 \qquad G'; D_2 \gg P \setminus G_2}{G'; D_1 \mathbin{\&} D_2 \gg P \setminus G_1 \oplus G_2}$$

$$\frac{(!G_1) \bullet G'; D \gg P \setminus G}{G'; G_1 \rightarrow D \gg P \setminus G} \qquad \frac{(¡G_1) \bullet G'; D \gg P \setminus G}{G'; G_1 \multimap D \gg P \setminus G}$$

$$\frac{G_1 \circ G'; D \gg P \setminus G}{G'; G_1 \twoheadrightarrow D \gg P \setminus G} \qquad \frac{G_1 \bullet G'; D \gg P \setminus G}{G'; G_1 \rightarrowtail D \gg P \setminus G}$$

Note that these inference rules will not fail when the input/output conventions mentioned above are respected; thus every clause formula can be successfully residuated.

Lemma 22

For every D,G',P there exists a unique, non-atomic G such that $G'; D \gg P \setminus G$

The example from the beginning of this section, $P_1 \twoheadrightarrow P_2 \rightarrowtail P$, can now be residuated as follows:

$$\cfrac{\cfrac{\cfrac{}{P_2 \bullet (P_1 \circ \mathbf{1}); P \gg P \setminus (P \doteq P) \bullet (P_2 \bullet (P_1 \circ \mathbf{1}))}}{P_1 \circ \mathbf{1}; P_2 \rightarrowtail P \gg P \setminus (P \doteq P) \bullet (P_2 \bullet (P_1 \circ \mathbf{1}))}}{\mathbf{1}; P_1 \twoheadrightarrow P_2 \rightarrowtail P \gg P \setminus (P \doteq P) \bullet (P_2 \bullet (P_1 \circ \mathbf{1}))}$$

where we use $\mathbf{1}$ to initialize the accumulated result.

We will remove the non-determinism from choice$_\Gamma$ and choice$_\Delta$ by replacing them with the following rules:

$$\frac{\mathbf{1}; D \gg P \setminus G \qquad \Gamma_L D \Gamma_R; \Delta; \Omega \xrightarrow{r} G}{\Gamma_L D \Gamma_R; \Delta; \Omega \xrightarrow{r} P}\,\text{choiceR}_\Gamma$$

[1] Head in the sense of Prolog or λProlog.

$$\frac{\mathbf{1}; D \gg P \setminus G \qquad \Gamma; \Delta_L \Delta_R; \Omega \xrightarrow{r} G}{\Gamma; \Delta_L D \Delta_R; \Omega \xrightarrow{r} P}\,\text{choiceR}_\Delta$$

All of the other inference rules remained unchanged from Section 7.1.

We now prove the following theorems which justifies such a change.

Theorem 23 (Soundness of Residuation)

1. $\Gamma; \Delta; \Omega \xrightarrow{r} G$ *implies* $\Gamma; \Delta; \Omega \longrightarrow G.$

2. $\Gamma; \Delta; (\Omega_L; \Omega_R) \xrightarrow{r} D \gg P$ *implies* $\Gamma; \Delta; (\Omega_L; \Omega_R) \longrightarrow D \gg P.$

3. $G'; D \gg P \setminus G$ *and* $\Gamma; \Delta; \Omega \xrightarrow{r} G$ *implies*
there exists Δ', Δ_D, Ω_L, Ω', *and* Ω_R *such that*
$\Omega = \Omega_L \Omega' \Omega_R$ *and* $\Delta = \Delta_D \bowtie \Delta'$ *and*
$\Gamma; \Delta'; \Omega' \xrightarrow{r} G'$ *and* $\Gamma; \Delta_D; (\Omega_L; \Omega_R) \longrightarrow D \gg P.$

Proof:
Parts 1 and 2: by structural induction on the given residuation derivation.
We show a representative case.

case: $$\frac{\mathbf{1}; D \gg P \setminus G \qquad \Gamma; \Delta_L \Delta_R; \Omega \xrightarrow{r} G}{\Gamma; \Delta_L D \Delta_R; \Omega \xrightarrow{r} P}\,\mathbf{choiceR}_\Delta$$

$\Gamma; \Delta_L \Delta_R; (\Omega_L; \Omega_R) \longrightarrow D \gg P$ and $\Gamma; \Delta'; \Omega' \xrightarrow{r} \mathbf{1}$
where $\Omega = \Omega_L \Omega' \Omega_R$ and $\Delta = \Delta_L \Delta_R \bowtie \Delta'$ — ind. hyp. (Pt. 3)

$\Delta' = \cdot = \Omega'$ — inversion

$\Gamma; \Delta_L D \Delta_R; \Omega_L \Omega_R \longrightarrow P$ — $\mathbf{choice}_\Omega$

Part 3: by structural induction on given residual derivation using inversion on the residuation derivation rules for multiplicative conjunctions. Note that G cannot be atomic. We give a representative case for part 3.

case: $\dfrac{G_1 \circ G'; D \gg P \setminus G}{G'; G_1 \twoheadrightarrow D \gg P \setminus G}$ and $\Gamma; \Delta; \Omega \xrightarrow{r} G$

$$\begin{array}{ll}
\Omega_L \Omega' \Omega_R = \Omega \text{ and } \Delta = \Delta_D \bowtie \Delta' \text{ and} & \\
\quad \Gamma; \Delta'; \Omega' \longrightarrow G_1 \circ G' \text{ and } \Gamma; \Delta_D; (\Omega_L; \Omega_R) \longrightarrow D \gg P & \text{ind. hyp.} \\
\Omega' = \Omega'_2 \Omega'_1 \text{ and } \Delta' = \Delta'_1 \bowtie \Delta'_2 \text{ and} & \\
\quad \Gamma; \Delta'_1; \Omega'_1 \xrightarrow{r} G_1 \text{ and } \Gamma; \Delta'_2; \Omega'_2 \xrightarrow{r} G' & \text{inversion on } \circ_R \\
\Gamma; \Delta_D \bowtie \Delta'_1; (\Omega_L; \Omega'_1 \Omega_R) \longrightarrow G_1 \twoheadrightarrow D \gg P & \text{by rule } \twoheadrightarrow_L
\end{array}$$

□

Theorem 24 (Completeness of Residuation)

1. $\Gamma; \Delta; \Omega \longrightarrow G$ *implies* $\Gamma; \Delta; \Omega \xrightarrow{r} G$.

2. $\Gamma; \Delta; (\Omega_L; \Omega_R) \longrightarrow D \gg P$ *implies* $\Gamma; \Delta; (\Omega_L; \Omega_R) \xrightarrow{r} D \gg P$.

3. $\Gamma; \Delta; (\Omega_L; \Omega_R) \longrightarrow D \gg P$ *and* $G'; D \gg P \setminus G$ *and* $\Gamma; \Delta'; \Omega \xrightarrow{r} G'$
implies $\Gamma; \Delta \bowtie \Delta'; \Omega_L \Omega \Omega_R \xrightarrow{r} G$.

Proof: Structural induction on given uniform derivation. We give a representative case for part 1, and for part 3.

case (for Part 1): $\dfrac{\Gamma; \Delta_L \Delta_R; (\Omega_L; \Omega_R) \longrightarrow D \gg P}{\Gamma; \Delta_L D \Delta_R; \Omega_L \Omega_R \longrightarrow P}\,\mathbf{choice}_\Delta$

$$\begin{array}{ll}
\Gamma; \cdot; \cdot \xrightarrow{r} \mathbf{1} & \text{rule } \mathbf{1}_R \\
\mathbf{1}; D \gg P \setminus G & \text{Lemma 22} \\
\Gamma; \Delta_L \Delta_R; \Omega_L \Omega_R \xrightarrow{r} G & \text{ind. hyp. (Pt. 3)} \\
\Gamma; \Delta_L D \Delta_R; \Omega_L \Omega_R \xrightarrow{r} P & \mathbf{choiceR}_\Omega
\end{array}$$

case (for Part 3): $\dfrac{\Gamma; \Delta; (\Omega_L; \Omega_R) \longrightarrow D \gg P \qquad \Gamma; \Delta_G; \Omega_1 \longrightarrow G_1}{\Gamma; \Delta \bowtie \Delta_G; (\Omega_L; \Omega_1 \Omega_R) \longrightarrow G_1 \twoheadrightarrow D \gg P}\twoheadrightarrow_L$

$$\begin{array}{ll}
G'; G_1 \twoheadrightarrow D \gg P \setminus G \text{ and } \Gamma; \Delta'; \Omega \xrightarrow{r} G' & \text{assumptions} \\
G_1 \circ G'; D \gg P \setminus G & \text{inversion} \\
\Gamma; \Delta_G; \Omega_1 \xrightarrow{r} G_1 & \text{ind. hyp. (Pt. 1)} \\
\Gamma; \Delta_G \bowtie \Delta'; \Omega\Omega_1 \xrightarrow{r} G_1 \circ G' & \text{by rule } \circ_R \\
\Gamma; \Delta \bowtie \Delta_G \bowtie \Delta'; \Omega_L\Omega\Omega_1\Omega_R \xrightarrow{r} G & \text{ind. hyp. (Pt. 3)}
\end{array}$$

□

Note that only mobile formulas can be residuated into goal formulas. The fact that these formulas can be placed anywhere in the ordered context is crucial for the correctness of the residuation transformation. There seems to be no way to logically compile ordered clause formulas into goal formulas because it would require recording the location of the clause formula in the resulting goal formula.

Chapter 8

Lazy Context Splitting

We now deal with the non-deterministic context splitting required of the $\twoheadrightarrow_L$, $\rightarrowtail_L$, $\multimap_L$, $\bullet_R$, and $\circ_R$ rules. For linear logic, it is well known that this non-determinism can be removed by using an input/output resource management system [27]. This approach, which "lazily" splits contexts by passing all formulas, or resources, to one premise and then giving the left-overs to the other premise, operationally requires fixing an order for proving the premises of an inference rule.

More general constraint based approaches, which do not require a fixed order for proving premises, are also possible for linear logic [22, 4]. It seems quite likely that similar constraint based systems can be developed for ordered linear logic. However these approaches, while of interest for general theorem proving, are not necessarily suited for a logic programming interpreter whose operational semantics must have a strong computational interpretation. Since we are presently aiming at developing a logic programming system, we will not pursue general constraint-based approaches to ordered resource management in this thesis.

As shown in [49] the input/output (IO) model can be extended to ordered linear logic. In this chapter, we present those results applied to the residuation derivation system of chapter 7.

8.1 An Ordered IO System

Linear hypotheses can be treated as in the so-called IO system of Hodas and Miller [27]. The rules $\bullet_R$, $\circ_R$, $\multimap_L$, $\twoheadrightarrow_L$, and $\rightarrowtail_L$ propagate all linear hypotheses to the first premise which returns the list of unused hypotheses when it has been solved successfully. These are then passed on to the second premise. The hypotheses used in neither premise are then returned as unused in the conclusion.

This model of deterministic *resource consumption* is intuitively attractive and easy to reason about for the programmer, but its extension to the ordered context is complicated by the need to preserve the order of the hypotheses:

$$\cdot;\cdot; P_2 P_1 (P_1 \rightarrowtail P_2 \rightarrowtail P) \longrightarrow P$$

is a derivable sequent while

$$\cdot;\cdot; P_1 P_2 (P_1 \rightarrowtail P_2 \rightarrowtail P) \longrightarrow P$$

is not. For the main judgement of uniform derivability, adding input and output contexts is straightforward.

$$\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \longrightarrow G$$

During the search, the *input contexts* Γ, Δ_I, and Ω_I and the goal G are given, while the *output contexts* Δ_O and Ω_O are returned. In the interest of economy (both for the presentation of the rules and the implementation) we do not actually delete formulas from Δ_I and Ω_I but replace them with a placeholder $\Box$.

The right rules for the ordered resource management judgement are constructed by expanding the contexts from the uniform derivation system into pairs of input/output contexts. Thus the $\twoheadrightarrow_R$ rule is just:

$$\frac{\Gamma \; ; \; \Delta_I \backslash \Delta_O \; ; \; \Omega_I D \backslash \Omega_O \Box \; \longrightarrow \; G}{\Gamma \; ; \; \Delta_I \backslash \Delta_O \; ; \; \Omega_I \backslash \Omega_O \; \longrightarrow \; D \twoheadrightarrow G} \twoheadrightarrow_R$$

We require the $\Box$ in the output context to ensure the hypothesis has actually been used. The other right rules (excluding those for $\bullet$ and $\circ$), as well as the $\mathbf{choice}_\Gamma$ and $\mathbf{choice}_\Delta$ rules, are constructed similarly. Section 8.2 contains the complete resource management system for ordered linear logic. Note that this lazy splitting of resources actually introduces non-determinism, the construction of linear and ordered output contexts, into the $\top_R$ rule.

Next we come to the $\mathbf{choice}_\Omega$ rule, that is, we choose to focus on an ordered assumption. This determines the division of the remaining ordered hypotheses. We therefore divide the input contexts and join the output contexts at the chosen assumption. The new judgement reads

$$\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \ ; \ \Omega_{RI} \backslash \Omega_{RO}) \ \longrightarrow \ D \gg P$$

where Ω_{LI} and Ω_{RI} are the parts to the left and right of the focussed formula D, and Ω_{LO} and Ω_{RO} are the corresponding output contexts. The **choice**$_\Omega$ rule for this system then looks as follows:

$$\frac{\Gamma \ ; \ \Delta_I \backslash \Delta_O \ ; \ (\Omega_{LI} \backslash \Omega_{LO} \ ; \ \Omega_{RI} \backslash \Omega_{RO}) \ \longrightarrow \ D \gg P}{\Gamma \ ; \ \Delta_I \backslash \Delta_O \ ; \ \Omega_{LI} D \Omega_{RI} \backslash \Omega_{LO} \Box \Omega_{RO} \ \longrightarrow \ P} \textbf{choice}_\Omega$$

Replacing D from the input context with $\Box$ in the output context indicates that D was consumed.

The **init** rule does not consume any resources except for the focus formula. Therefore all input resources are passed on.

$$\frac{}{\Gamma; \Delta \backslash \Delta; (\Omega_L \backslash \Omega_L; \Omega_R \backslash \Omega_R) \longrightarrow P \gg P} \textbf{init}$$

This effectively states that the linear and ordered contexts of the initial sequent should be empty.

The $\twoheadrightarrow_L$, $\multimap_L$, $\&_L$, and $\forall_L$ rules for this judgement introduce no new ideas.

We now consider the left rule for right implication. We are trying to derive a judgement of the form

$$\Gamma \ ; \ \Delta_I \backslash ? \ ; \ (\Omega_{LI} \backslash ? \ ; \ \Omega_{RI} \backslash ?) \longrightarrow G \twoheadrightarrow D \gg P$$

where the output contexts denoted by **?** have yet to be computed. The linear output context can be threaded through each of the premises without further analysis. However for the ordered output contexts, we need to do a bit more work. Because $G \twoheadrightarrow D$ is a right implication situated between Ω_{LI} and Ω_{RI}, the derivation of G must consume some initial segment of Ω_{RI}. Before that, we need to see if D immediately entails P (the left premise of the $\twoheadrightarrow_L$ rule)[1] which we obtain from

$$\Gamma \ ; \ \Delta_I \backslash \Delta_M \ ; \ (\Omega_{LI} \backslash \Omega_{LO} \ ; \ \Omega_{RI} \backslash \Omega) \longrightarrow D \gg P$$

Then we need to take the unconsumed parts at the left end of Ω, denoted by Ω_{GI}, and allow them as the ordered input context for the solution to G.

$$\Gamma \ ; \ \Delta_M \backslash \Delta_O \ ; \ \Omega_{GI} \backslash \Omega_{GO} \longrightarrow G$$

[1] In Prolog terminology: we need to unify the clause head with P before solving any subgoals.

Now we can fill the holes in the conclusion with Ω_{LO} and $\Omega_{GO}\Omega_{RO}$, respectively. In summary, the rule reads

$$\frac{\Gamma; \Delta_I \backslash \Delta_M; (\Omega_{LI} \backslash \Omega_{LO} \ ; \ \Omega_{RI} \backslash \Omega_{GI}\Omega_{RO}) \ \longrightarrow \ D \gg P \qquad \Gamma; \Delta_M \backslash \Delta_O; \Omega_{GI} \backslash \Omega_{GO} \ \longrightarrow \ G}{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \ ; \ \Omega_{RI} \backslash \Omega_{GO}\Omega_{RO}) \ \longrightarrow \ G \twoheadrightarrow D \gg P} \twoheadrightarrow_{L1}$$

where Ω_{GI} is the longest leftmost segment of $\Omega_{GI}\Omega_{RO}$ not containing $\square$, and Ω_{RO} the remainder. We will denote this with the following notation:

$$\square \notin \Omega_{GI} \quad \text{and} \quad \Omega_{RO} = \square\_ \text{ or } \cdot$$

Note that $\Omega_{RO} = \square\_$ or $\cdot$ really stands for $\exists\Omega.\ \Omega_{RO} = \square\Omega$ or $\Omega_{RO} = \cdot$. We will use the symmetric notation, $\Omega_{LO} = \_\square$ or $\cdot$ to form the $\rightarrowtail_L$ and $\bullet_R$ rules.

Further note that the context split appearing in the $\twoheadrightarrow_L$ rule, $\Omega_{GI}\Omega_{RO}$, *is* deterministic.

In order to demonstrate the intended operational use of the input/output system, we now walk through a short example derivation (where we elide the always empty unrestricted and linear contexts); figure 8.1 contains the complete formal derivation (still with elided contexts). Consider trying to prove the closed formula:

$$(P_1 \twoheadrightarrow P_2 \twoheadrightarrow P) \twoheadrightarrow P_1 \twoheadrightarrow P_2 \twoheadrightarrow P$$

Since we want a self-contained proof, we begin with the sequent:

$$\cdot\backslash\cdot \ \longrightarrow \ (P_1 \twoheadrightarrow P_2 \twoheadrightarrow P) \twoheadrightarrow P_1 \twoheadrightarrow P_2 \twoheadrightarrow P$$

where we specify that the ordered output context is empty. After three uses of the $\twoheadrightarrow_R$ rule we arrive at the following situation:

$$(P_1 \twoheadrightarrow P_2 \twoheadrightarrow P)P_1P_2\backslash ? \ \longrightarrow \ P$$

where we have not yet computed the ordered output context. We must now choose a formula to focus on. Of course we will choose the only one which works:

$$\cdot\backslash\cdot\ ;\ P_1P_2\backslash ? \ \longrightarrow \ P_1 \twoheadrightarrow P_2 \twoheadrightarrow P \gg P$$

We now start applying $\twoheadrightarrow_L$ rules. We will always try the major premise first and put the minor premise on a stack of things to do. Thus we arrive at the sequent:

$$\cdot\backslash\cdot\ ;\ P_1P_2\backslash P_1P_2 \ \longrightarrow \ P \gg P$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot\backslash\cdot\,;\,P_1P_2\backslash P_1P_2 \longrightarrow P \gg P}\,\mathbf{init} \qquad \dfrac{\dfrac{}{P_1\backslash P_1\,;\,\cdot\backslash\cdot \longrightarrow P_2 \gg P_2}\,\mathbf{init}}{P_1P_2\backslash P_1\Box \longrightarrow P_2}\,\mathbf{choice}_\Omega}{\cdot\backslash\cdot\,;\,P_1P_2\backslash P_1\Box \longrightarrow P_2 \twoheadrightarrow P \gg P}\twoheadrightarrow_L \qquad \dfrac{\dfrac{}{\cdot\backslash\cdot\,;\,\cdot\backslash\cdot \longrightarrow P_1 \gg P_1}\,\mathbf{init}}{P_1\backslash\Box \longrightarrow P_1}\,\mathbf{choice}_\Omega}{\cdot\backslash\cdot\,;\,P_1P_2\backslash\Box\Box \longrightarrow P_1 \twoheadrightarrow P_2 \twoheadrightarrow P \gg P}\twoheadrightarrow_L}{(P_1 \twoheadrightarrow P_2 \twoheadrightarrow P)P_1P_2\backslash\Box\Box\Box \longrightarrow P}\,\mathbf{choice}_\Omega}{(P_1 \twoheadrightarrow P_2 \twoheadrightarrow P)P_1\backslash\Box\Box \longrightarrow P_2 \twoheadrightarrow P}\twoheadrightarrow_R}{(P_1 \twoheadrightarrow P_2 \twoheadrightarrow P)\backslash\Box \longrightarrow P_1 \twoheadrightarrow P_2 \twoheadrightarrow P}\twoheadrightarrow_R}{\cdot\backslash\cdot \longrightarrow (P_1 \twoheadrightarrow P_2 \twoheadrightarrow P) \twoheadrightarrow P_1 \twoheadrightarrow P_2 \twoheadrightarrow P}\twoheadrightarrow_R$$

Figure 8.1: Sample derivation

Because this sequent is initial, we know that the input and output ordered contexts are equal. We can now move on to the proof of P_2:

$$P_1P_2\backslash\mathbf{?} \longrightarrow P_2$$

We will prove this by focussing on P_2. After a $\mathbf{choice}_\Omega$ rule, we get to the initial sequent:

$$P_1\backslash P_1\,;\,\cdot\backslash\cdot \longrightarrow P_2 \gg P_2$$

We know then, from the form of the $\mathbf{choice}_\Omega$ rule that the output context for the proof of P_2, the immediately preceding $\mathbf{?}$, is $P_1\Box$. We may now move on to proving P_1. Note that the input context for the proof of P_1 will only consist of P_1:

$$P_1\backslash\mathbf{?} \longrightarrow P_1$$

We will focus on P_1, which will give us an initial sequent. Then, coming out of the $\mathbf{choice}_\Omega$ rule, we will be able to compute that the output context above is just $\Box$. At this point, we have no presmises left to prove, however we are not done. We must continue back "down" the proof tree, filling in output contexts and checking to make sure the conditions on the output contexts required by the $\twoheadrightarrow_R$ rules are met.

The right rule for $\bullet$ is handled similarly. The left rule for $\rightarrowtail$ and the right rule for $\circ$ are symmetrically fashioned.

8.2 Resource Management System for OLL

We now present the complete resource management system (RMS) for ordered linear logic. We have two types of sequent for our RMS derivations

$$\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \;\longrightarrow\; G$$
$$\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\;;\;\Omega_{RI}\backslash\Omega_{RO}) \;\longrightarrow\; D \gg P$$

corresponding to the two types of sequents in the previous sections. are the output contexts. The unrestricted and ordered input contexts, Γ, Ω_I, Ω_{LI}, and Ω_{RI}, are lists of clauses which *do not* contain placeholders, $\Box$. The linear input context, Δ_I, is a list which may contain both formulas and $\Box$. Likewise, all of the output contexts, Δ_O, Ω_O, Ω_{LO}, and Ω_{RO}, may contain both formulas and $\Box$.

We define the following "superset" relation on input/output contexts, where we use Ψ to stand for any kind of context (unrestricted, linear, or ordered):

$$\frac{}{\cdot \sqsupseteq \cdot} \qquad \frac{\Psi \sqsupseteq \Psi'}{\Psi D \sqsupseteq \Psi' D} \qquad \frac{\Psi \sqsupseteq \Psi'}{\Psi D \sqsupseteq \Psi'\Box} \qquad \frac{\Psi \sqsupseteq \Psi'}{\Psi\Box \sqsupseteq \Psi'\Box}$$

It will be an invariant of our derivation rules that input contexts will always be supersets of their associated output contexts, *i.e.*, anytime $\Psi_I\backslash\Psi_O$ appears in a valid derivation, $\Psi_I \sqsupseteq \Psi_O$ holds.

We use the notation $\|\Psi\|$ to denote the length of list Ψ. Note that $\Psi \sqsupseteq \Psi'$ implies that $\|\Psi\| = \|\Psi'\|$.

Here are the RMS derivation rules:

$$\frac{}{\Gamma;\Delta\backslash\Delta;\Omega\backslash\Omega \;\longrightarrow\; P \doteq P}\,{\doteq_R} \qquad \frac{}{\Gamma;\Delta\backslash\Delta;\Omega\backslash\Omega \;\longrightarrow\; \mathbf{1}}\,\mathbf{1}_R$$

$$\frac{\Gamma;\cdot\backslash\cdot;\cdot\backslash\cdot \;\longrightarrow\; G}{\Gamma;\Delta\backslash\Delta;\Omega\backslash\Omega \;\longrightarrow\; !G}\,!_R \qquad \frac{\Gamma;\Delta_I\backslash\Delta_O;\cdot\backslash\cdot \;\longrightarrow\; G}{\Gamma;\Delta_I\backslash\Delta_O;\Omega\backslash\Omega \;\longrightarrow\; \text{¡}G}\,\text{¡}_R$$

$$\frac{\Gamma D;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \;\longrightarrow\; G}{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \;\longrightarrow\; D \rightarrow G}\,{\rightarrow_R} \qquad \frac{\Gamma;\Delta_I D\backslash\Delta_O\Box;\Omega_I\backslash\Omega_O \;\longrightarrow\; G}{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \;\longrightarrow\; D \multimap G}\,{\multimap_R}$$

$$\frac{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I D\backslash\Omega_O\Box \;\longrightarrow\; G}{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \;\longrightarrow\; D \twoheadrightarrow G}\,{\twoheadrightarrow_R} \qquad \frac{\Gamma;\Delta_I\backslash\Delta_O;D\Omega_I\backslash\Box\Omega_O \;\longrightarrow\; G}{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \;\longrightarrow\; D \rightarrowtail G}\,{\rightarrowtail_R}$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_M; \Omega_I \backslash \Omega_L \Omega_2 \;\longrightarrow\; G_1 \qquad \Gamma; \Delta_M \backslash \Delta_O; \Omega_2 \backslash \Omega_R \;\longrightarrow\; G_2}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_L \Omega_R \;\longrightarrow\; G_1 \bullet G_2} \bullet_R$$

$(\Box \notin \Omega_2$ and $(\Omega_L = \_\Box$ or $\cdot))$

$$\frac{\Gamma; \Delta_I \backslash \Delta_M; \Omega_I \backslash \Omega_2 \Omega_R \;\longrightarrow\; G_1 \qquad \Gamma; \Delta_M \backslash \Delta_O; \Omega_2 \backslash \Omega_L \;\longrightarrow\; G_2}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_L \Omega_R \;\longrightarrow\; G_1 \circ G_2} \circ_R$$

$(\Box \notin \Omega_2$ and $(\Omega_R = \Box\_$ or $\cdot))$

$$\frac{\Omega_I \sqsupseteq \Omega_O \;\text{ and }\; \Delta_I \sqsupseteq \Delta_O}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \;\longrightarrow\; \top} \top_R \qquad \frac{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \;\longrightarrow\; G_1 \qquad \Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \;\longrightarrow\; G_2}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \;\longrightarrow\; G_1 \,\&\, G_2} \&_R$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \;\longrightarrow\; G_1}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \;\longrightarrow\; G_1 \oplus G_2} \oplus_{R1} \qquad \frac{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \;\longrightarrow\; G_2}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \;\longrightarrow\; G_1 \oplus G_2} \oplus_{R2}$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \;\longrightarrow\; G[a/x]}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \;\longrightarrow\; \forall x.\, G} \forall_R^a \qquad \frac{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \;\longrightarrow\; G[t/x]}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \;\longrightarrow\; \exists x.\, G} \exists_R$$

(a not free in conclusion)

$$\frac{\mathbf{1}; D \gg P \setminus G \qquad \Gamma_L D \Gamma_R; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \;\longrightarrow\; G}{\Gamma_L D \Gamma_R; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \;\longrightarrow\; P} \mathbf{choice}_\Gamma$$

$$\frac{\mathbf{1}; D \gg P \setminus G \qquad \Gamma; \Delta_{LI} \Box \Delta_{RI} \backslash \Delta_O; \Omega_I \backslash \Omega_O \;\longrightarrow\; G}{\Gamma; \Delta_{LI} D \Delta_{RI} \backslash \Delta_O; \Omega_I \backslash \Omega_O \;\longrightarrow\; P} \mathbf{choice}_\Delta$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \;;\; \Omega_{RI} \backslash \Omega_{RO}) \;\longrightarrow\; D \gg P}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_{LI} D \Omega_{RI} \backslash \Omega_{LO} \Box \Omega_{RO} \;\longrightarrow\; P} \mathbf{choice}_\Omega$$

$$\frac{}{\Gamma; \Delta \backslash \Delta; (\Omega_L \backslash \Omega_L \;;\; \Omega_R \backslash \Omega_R) \;\longrightarrow\; P \gg P} \mathbf{init}$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \;;\; \Omega_{RI} \backslash \Omega_{RO}) \;\longrightarrow\; D_1 \gg P}{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \;;\; \Omega_{RI} \backslash \Omega_{RO}) \;\longrightarrow\; D_1 \,\&\, D_2 \gg P} \&_{L1}$$

$$\frac{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{RO})\ \longrightarrow\ D_2 \gg P}{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{RO})\ \longrightarrow\ D_1 \mathbin{\&} D_2 \gg P}\,\&_{L2}$$

$$\frac{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{RO})\ \longrightarrow\ D[t/x] \gg P}{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{RO})\ \longrightarrow\ \forall x.\, D \gg P}\,\forall_L$$

$$\frac{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{RO})\ \longrightarrow\ D \gg P \qquad \Gamma;\cdot\backslash\cdot;\cdot\backslash\cdot\ \longrightarrow\ G}{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{RO})\ \longrightarrow\ G \to D \gg P}\,\to_L$$

$$\frac{\Gamma;\Delta_I\backslash\Delta_M;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{RO})\ \longrightarrow\ D \gg P \qquad \Gamma;\Delta_M\backslash\Delta_O;\cdot\backslash\cdot\ \longrightarrow\ G}{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{RO})\ \longrightarrow\ G \multimap D \gg P}\,\multimap_L$$

$$\frac{\Gamma;\Delta_I\backslash\Delta_M;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{GI}\Omega_{RO})\ \longrightarrow\ D \gg P \qquad \Gamma;\Delta_M\backslash\Delta_O;\Omega_{GI}\backslash\Omega_{GO}\ \longrightarrow\ G}{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{GO}\Omega_{RO})\ \longrightarrow\ G \twoheadrightarrow D \gg P}\,\twoheadrightarrow_L$$
$(\square \notin \Omega_{GI}$ and $(\Omega_{RO} = \square\_$ or $\cdot))$

$$\frac{\Gamma;\Delta_I\backslash\Delta_M;(\Omega_{LI}\backslash\Omega_{LO}\Omega_{GI}\ ;\ \Omega_{RI}\backslash\Omega_{RO})\ \longrightarrow\ D \gg P \qquad \Gamma;\Delta_M\backslash\Delta_O;\Omega_{GI}\backslash\Omega_{GO}\ \longrightarrow\ G}{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\Omega_{GO}\ ;\ \Omega_{RI}\backslash\Omega_{RO})\ \longrightarrow\ G \twoheadrightarrow D \gg P}\,\rightarrowtail_L$$
$(\square \notin \Omega_{GI}$ and $(\Omega_{LO} = \_\square$ or $\cdot))$

Note that the **choice**$_\Delta$ rule only allows an actual formula, and not a $\square$, to be focussed upon.

The RMS system satisfies the following basic properties which we shall rely upon to prove its correctness.

Lemma 25

1. $\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O\ \longrightarrow\ G$ *implies* $\Omega_I \sqsupseteq \Omega_O$ *and* $\Delta_I \sqsupseteq \Delta_O$.

2. $\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{RO})\ \longrightarrow\ D \gg P$ *implies*
   $\Omega_{LI} \sqsupseteq \Omega_{LO}$ *and* $\Omega_{LI} \sqsupseteq \Omega_{LO}$ *and* $\Delta_I \sqsupseteq \Delta_O$.

3. $\Gamma; \Delta_I \backslash \Delta_O; \Omega_{IL}\Omega_{IR} \backslash \Omega_{OL}\Omega_{OR} \longrightarrow G$ *and* $\Box \notin \Omega, \Delta$ *and* $\Omega_{IL} \sqsupseteq \Omega_{OL}$
   implies $\Gamma; \Delta_I \bowtie \Delta \backslash \Delta_O \bowtie \Delta; \Omega_{IL}\Omega\Omega_{IR} \backslash \Omega_{OL}\Omega\Omega_{OR} \longrightarrow G.$

4. $\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LIL}\Omega_{LIR} \backslash \Omega_{LOL}\Omega_{LOR} \,;\, \Omega_{RI} \backslash \Omega_{RO}) \longrightarrow D \gg P$
   and $\Box \notin \Omega$ *and* $\Omega_{LIL} \sqsupseteq \Omega_{LOL}$ *implies*
   $\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LIL}\Omega\Omega_{LIR} \backslash \Omega_{LOL}\Omega\Omega_{LOR} \,;\, \Omega_{RI} \backslash \Omega_{RO}) \longrightarrow D \gg P.$

5. $\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \,;\, \Omega_{RIL}\Omega_{RIR} \backslash \Omega_{ROL}\Omega_{ROR}) \longrightarrow D \gg P$
   and $\Box \notin \Omega$ *and* $\Omega_{RIL} \sqsupseteq \Omega_{ROL}$ *implies*
   $\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \,;\, \Omega_{RIL}\Omega\Omega_{RIR} \backslash \Omega_{ROL}\Omega\Omega_{ROR}) \longrightarrow D \gg P.$

Proof: By structural induction on given derivation. □

8.3 Correctness of RMS

In order to prove that RMS derivations are sound and complete with respect to residuation derivations, we define the difference of an input context, and output context, $\Psi_I - \Psi_O$, as follows:

$$\begin{array}{rcl}
\cdot - \cdot & = & \cdot \\
\Psi_I D - \Psi_O D & = & (\Psi_I - \Psi_O) \\
\Psi_I \Box - \Psi_O \Box & = & (\Psi_I - \Psi_O) \\
\Psi_I D - \Psi_O \Box & = & (\Psi_I - \Psi_O)\, D
\end{array}$$

where input patterns not covered are undefined. Thus $\|\Psi_I\| \neq \|\Psi_O\|$ implies $\Psi_I - \Psi_O$ is undefined. Also note that $\Psi_I \sqsupseteq \Psi_O$ implies $\Psi_I - \Psi_O$ is defined; and conversely, $\Psi_I - \Psi_O$ is defined implies $\Psi_I \sqsupseteq \Psi_O$.

We begin by showing soundness.

Theorem 26 (Soundess of RMS)

1. $\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \longrightarrow G$ *implies* $\Gamma; \Delta_I - \Delta_O; \Omega_I - \Omega_O \xrightarrow{r} G.$

2. $\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \,;\, \Omega_{RI} \backslash \Omega_{RO}) \longrightarrow D \gg P$ *implies*
   $\Gamma; \Delta_I - \Delta_O; (\Omega_{LI} - \Omega_{LO}; \Omega_{RI} - \Omega_{RO}) \xrightarrow{r} D \gg P.$

Proof: By structural induction on given RMS derivation. We give representative cases.

case: $$\frac{}{\Gamma;\Delta\backslash\Delta;(\Omega_L\backslash\Omega_L\ ;\ \Omega_R\backslash\Omega_R)\ \longrightarrow\ P\gg P}\,\textbf{init}$$

Then

$\Gamma;\cdot;(\cdot;\cdot)\ \xrightarrow{r}\ P\gg P$ **init**

case: $$\frac{\mathbf{1};D\gg P\setminus G \qquad \Gamma;\Delta_{IL}\Box\Delta_{IR}\backslash\Delta_O;\Omega_I\backslash\Omega_O\ \longrightarrow\ G}{\Gamma;\Delta_{IL}D\Delta_{IR}\backslash\Delta_O;\Omega_I\backslash\Omega_O\ \longrightarrow\ P}\,\textbf{choice}_\Delta$$

Then

$\Gamma;\Delta_{IL}\Box\Delta_{IR}-\Delta_O;\Omega_I-\Omega_O\ \xrightarrow{r}\ G$ ind. hyp.

$\Delta_O=\Delta_{OL}\Box\Delta_{OR}$ where $\Delta_{IL}\sqsupseteq\Delta_{OL}$ $\Delta_{IL}\Box\Delta_{IR}\sqsupseteq\Delta_O$

$\Delta_{IL}\Box\Delta_{IR}-\Delta_O=(\Delta_{IL}-\Delta_{OL})(\Delta_{IR}-\Delta_{OR})$

$\Gamma;(\Delta_{IL}-\Delta_{OL})D(\Delta_{IR}-\Delta_{OR});\Omega_I-\Omega_O\ \xrightarrow{r}\ P$ $\textbf{choice}_\Delta$

Note $(\Delta_{IL}-\Delta_{OL})D(\Delta_{IR}-\Delta_{OR})=\Delta_{IL}D\Delta_{IR}-\Delta_{OL}\Box\Delta_{OR}$

case: $$\frac{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{RO})\ \longrightarrow\ D\gg P}{\Gamma;\Delta_I\backslash\Delta_O;\Omega_{LI}D\Omega_{RI}\backslash\Omega_{LO}\Box\Omega_{RO}\ \longrightarrow\ P}\,\textbf{choice}_\Omega$$

Then

$\Gamma;\Delta_I-\Delta_O;(\Omega_{LI}-\Omega_{LO};\Omega_{RI}-\Omega_{RO})\ \xrightarrow{r}\ D\gg P$ ind. hyp.

$\Gamma;\Delta_I-\Delta_O;(\Omega_{LI}-\Omega_{LO})D(\Omega_{RI}-\Omega_{RO})\ \xrightarrow{r}\ P$ $\textbf{choice}_\Omega$

Note $(\Omega_{LI}-\Omega_{LO})D(\Omega_{RI}-\Omega_{RO})=\Omega_{LI}D\Omega_{RI}-\Omega_{LO}\Box\Omega_{RO}$

case: $$\frac{\Gamma;\Delta_I\backslash\Delta_M;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{GI}\Omega_{RO})\ \longrightarrow\ D\gg P \qquad \Gamma;\Delta_M\backslash\Delta_O;\Omega_{GI}\backslash\Omega_{GO}\ \longrightarrow\ G}{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{GO}\Omega_{RO})\ \longrightarrow\ G\twoheadrightarrow D\gg P}\,{\twoheadrightarrow_L}$$

$(\Box\notin\Omega_{GI}$ and $(\Omega_{RO}=\Box\_$ or $\cdot))$

Then

$\Omega_{RI}=\Omega_{GI}\Omega_{RR}$ lemma 25

$\Omega_{RI}-\Omega_{GI}\Omega_{RO}=\Omega_{RR}-\Omega_{RO}$ $\Box\notin\Omega_{GI}$ and defn. of -

$\Omega_{RI}-\Omega_{GO}\Omega_{RO}=(\Omega_{GI}-\Omega_{GO})(\Omega_{RR}-\Omega_{RO})$ defn. of -

$\Gamma; \Delta_M - \Delta_O; \Omega_{GI} - \Omega_{GO} \xrightarrow{r} G$ ind. hyp.

$\Gamma; \Delta_I - \Delta_M; (\Omega_{LI} - \Omega_{LO}; \Omega_{RR} - \Omega_{RO}) \xrightarrow{r} D \gg P$ ind. hyp.

Note: $(\Delta_I - \Delta_M) \bowtie (\Delta_M - \Delta_O) = \Delta_I - \Delta_O$

$\Gamma; \Delta_I - \Delta_O; (\Omega_{LI} - \Omega_{LO}; (\Omega_{GI} - \Omega_{GO})(\Omega_{RR} - \Omega_{RO})) \xrightarrow{r} G \twoheadrightarrow D \gg P$ $\twoheadrightarrow_L$

□

We immediately get the following corollary:

Corollary 27

1. $\Gamma; \cdot\backslash\cdot; \cdot\backslash\cdot \longrightarrow G$ *implies* $\Gamma; \cdot; \cdot \xrightarrow{r} G.$
2. $\Gamma; \cdot\backslash\cdot; (\cdot\backslash\cdot\ ;\ \cdot\backslash\cdot) \longrightarrow G \gg P$ *implies* $\Gamma; \cdot; (\cdot; \cdot) \xrightarrow{r} G \gg P.$

We now move on to showing completeness.

Theorem 28 (Completeness of RMS)

1. $\Gamma; \Delta_I - \Delta_O; \Omega_I - \Omega_O \xrightarrow{r} G$ *and* $\Box \notin \Omega_I$
   implies $\Gamma; \Delta_I\backslash\Delta_O; \Omega_I\backslash\Omega_O \longrightarrow G.$

2. $\Gamma; \Delta_I - \Delta_O; (\Omega_{LI} - \Omega_{LO}; \Omega_{RI} - \Omega_{RO}) \xrightarrow{r} D \gg P$
   and $\Box \notin \Omega_{LI}$ *and* $\Box \notin \Omega_{RI}$ *implies*
   $\Gamma; \Delta_I\backslash\Delta_O; (\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{RO}) \longrightarrow D \gg P.$

Proof: By structural induction on given residuation derivation. We give representative cases.

case: $$\frac{}{\Gamma; \cdot; (\cdot; \cdot) \xrightarrow{r} P \gg P}\,\textbf{init}$$

Then

$\Delta_I - \Delta_O = \cdot$ and $\Omega_{LI} - \Omega_{LO} = \cdot$ and $\Omega_{LI} - \Omega_{LO} = \cdot$ assumptions

$\Delta_I = \Delta_O$ and $\Omega_{LI} = \Omega_{LO}$ and $\Omega_{RI} = \Omega_{RO}$ defn. of -

$\Gamma; \Delta_I\backslash\Delta_I; (\Omega_{LI}\backslash\Omega_{LI}\ ;\ \Omega_{RI}\backslash\Omega_{RI}) \longrightarrow P \gg P$ **init** case: $$\frac{\mathbf{1}; D \gg P \setminus G \qquad \Gamma; \Delta_L\Delta_R; \Omega \xrightarrow{r} G}{\Gamma; \Delta_L D \Delta_R; \Omega \xrightarrow{r} P}\,\textbf{choice}_{\Delta}$$

Then

$$\begin{array}{ll}
\Delta_I - \Delta_O = \Delta_L D \Delta_R \text{ and } \Omega_I - \Omega_O = \Omega & \text{assumptions} \\
\Delta_I = \Delta_{IL} D \Delta_{IR} \text{ and } \Delta_O = \Delta_{OL} \Box \Delta_{OR} \text{ and } \Delta_{IL} - \Delta_{OL} = \Delta_L & \\
\Delta_{IL} \Box \Delta_{IR} - \Delta_{OL} \Box \Delta_{OR} = \Delta_L \Delta_R & \\
\Gamma; \Delta_{IL} \Box \Delta_{IR} \backslash \Delta_O; \Omega_I \backslash \Omega_O \longrightarrow G & \text{ind. hyp.} \\
\Gamma; \Delta_{IL} D \Delta_{IR} \backslash \Delta_O; \Omega_I \backslash \Omega_O \longrightarrow P & \textbf{choice}_{\Delta}
\end{array}$$

case: $$\frac{\Gamma; \Delta; (\Omega_L; \Omega_R) \xrightarrow{r} D \gg P}{\Gamma; \Delta; \Omega_L D \Omega_R \xrightarrow{r} P}\,\textbf{choice}_{\Omega}$$

Then

$$\begin{array}{ll}
\Delta_I - \Delta_O = \Delta \text{ and } \Omega_I - \Omega_O = \Omega_L D \Omega_R & \text{assumptions} \\
\Omega_I = \Omega_{IL} D \Omega_{IR} \text{ and } \Omega_O = \Omega_{OL} \Box \Omega_{OR} & \\
\quad \text{where } \Omega_{IL} - \Omega_{OL} = \Omega_L \text{ and } \Omega_{IR} - \Omega_{OR} = \Omega_R & \text{defn. of -} \\
\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{IL} \backslash \Omega_{OL}\ ;\ \Omega_{IR} \backslash \Omega_{OR}) \longrightarrow D \gg P & \text{ind. hyp.} \\
\Gamma; \Delta_I \backslash \Delta_O; \Omega_{IL} D \Omega_{IR} \backslash \Omega_{OL} \Box \Omega_{OR} \longrightarrow P & \textbf{choice}_{\Omega}
\end{array}$$

case: $$\frac{\Gamma; \Delta; (\Omega_L; \Omega_R) \xrightarrow{r} D \gg P \qquad \Gamma; \Delta_G; \Omega_G \xrightarrow{r} G}{\Gamma; \Delta \bowtie \Delta_G; (\Omega_L; \Omega_G \Omega_R) \xrightarrow{r} G \twoheadrightarrow D \gg P}\twoheadrightarrow_L$$

Then

$$\begin{array}{ll}
\text{Let } \Delta_I - \Delta_O = \Delta \bowtie \Delta_G \text{ and } \Box \notin \Omega_{LI} \text{ and } \Box \notin \Omega_{RI} & \\
\quad \text{and } \Omega_{LI} - \Omega_{LO} = \Omega_L \text{ and } \Omega_{RI} - \Omega'_{RO} = \Omega_G \Omega_R & \text{assumptions} \\
\text{Let } \Delta_M \text{ be a context s.t. } \Delta_I - \Delta_M = \Delta \text{ and } \Delta_M - \Delta_O = \Delta_G & \\
\text{Let } \Omega_{GI} \Omega_{RR} = \Omega_{RI} \text{ and } \Omega_{GO} \Omega_{RO} = \Omega'_{RO} \text{ where} & \\
\quad \Omega_{GI} - \Omega_{GO} = \Omega_G \text{ and } \Omega_{RR} - \Omega_{RO} = \Omega_R \text{ and } (\Omega_{RO} = \Box\_ \text{ or } \cdot) & \\
\Gamma; \Delta_I \backslash \Delta_M; (\Omega_{LI} \backslash \Omega_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{GI} \Omega_{RO}) \longrightarrow D \gg P & \text{ind. hyp.} \\
\Gamma; \Delta_M \backslash \Delta_O; \Omega_{GI} \backslash \Omega_{GO} \longrightarrow G & \text{ind. hyp.} \\
\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{GO} \Omega_{RO}) \longrightarrow G \twoheadrightarrow D \gg P & \text{by rule } \twoheadrightarrow_L
\end{array}$$

□

The RMS derivation succeeds in removing all non-deterministic context-splits. However, this comes at the cost of introducing non-determinism into the $\top_R$ rule. This new non-determinism must also be eradicated in order to have a feasible basis for a logic programming language.

Chapter 9

Lazy Erasure

As noted in Chapter 8, we removed the non-deterministic context splits at the expense of making the $\top_R$ rule highly non-deterministic. A similar situation occurs in the development of Lolli [26] and is simply dealt with by adding a binary flag to each sequent denoting whether, or not, a $\top$ capable of consuming the linear hypotheses occurs in the derivation; if such a $\top$ does occur, then linearity can be relaxed. In that system the $\top_R$ rule does nothing except set this flag. Thus we may think of $\top$ consuming, or erasing, formulas in a lazy fashion just as the RMS system of Chapter 8 lazily splits contexts.

Unfortunately, such a simple solution is not possible for ordered linear logic. The fact that ordered contexts are split apart and later recombined prevents a solution with a single binary flag– consider that the immediate entailment sequents maintain two separate ordered context input/output pairs. However, there is a way to extend the idea of a $\top$ flag which allows us to remove the non-determinism from the $\top_R$ rule. This chapter explains how we accomplish that task.

9.1 Making $\top$ Deterministic

In this section we remove the non-determinism from the $\top_R$ rule by making it lazy– it will simply pass on its input context– and adding some new information to the sequents to keep track of which parts of the ordered context could have been consumed by a $\top$.

Rather than use a single $\top$-flag, we will use a list, τ, containing both 0 and 1 which serves as an abstraction of the output context. The 0s in τ will correspond to the $\Box$s in Ω. The 1s in τ will corresponds to regions of Ω which could have been consumed by a $\top$. For example consider the following:

$$\Omega = \Box\Omega_1\Box\Omega_2\Box \qquad\qquad \tau = 0100$$

where $\Box \notin \Omega_1\Omega_2$. The 1 in τ states that Ω_1 passed through a $\top_R$ rule and thus need not be explicitly consumed. However, Ω_2 must be explicitly consumed since there is no 1 in τ matching it. Thus the τs may also be thought of as abstractions of the output contexts which express strictness (whether formulas must be explicitly consumed) constraints. In analogy to output contexts, we will use the notation $\tau = 0$_ or $\cdot$, and its symmetric variation, to stand for $\exists\tau'.\ \tau = 0\tau'$ or $\tau = \cdot$.

In order to simplify our presentation, we shall restrict τ to never have two consecutive 1s. To cut down the number of explicit cases we must examine in our subsequent development, we define the following concatenation operators for τ which contract adjacent 1s:

$$\begin{array}{rcl} \cdot + \tau' & = & \tau' \\ \tau 0 + \tau' & = & \tau 0 \tau' \\ \tau 1 + \cdot & = & \tau 1 \\ \tau 1 + 0\tau' & = & \tau 1 0 \tau' \\ \tau 1 + 1\tau' & = & \tau 1 \tau' \end{array} \qquad\qquad \begin{array}{rcl} \cdot * \cdot & = & \cdot \\ 1 * \cdot & = & \cdot \\ \cdot * 1 & = & \cdot \\ 1 * 1 & = & 1 \end{array}$$

We will use the "disjunctive" concatenation, $+$, to form output contexts in the inference rules developed in this section. We will only use the "conjunctive" concatenation, $*$, in the definition of **mrg**, a helper function defined below.

Note that the constraint $0 \notin \tau$ implies that $\tau = \cdot$ or $\tau = 1$.

As a result of making $\top_R$ lazy, two different, yet compatible, output contexts might be computed for the two premises of the $\&_R$ rule. In order to give a correct version of the rule, we need to be able to merge compatible contexts into a form which preserves the constraints on both contexts. With this in mind, we define a helper relation, $\mathbf{mrg}(\Omega_1, \tau_1, \Omega_2, \tau_2, \Omega, \tau)$, to merge two compatible pairs of ordered output contexts and their abstractions. Note that **mrg** is a function where the first four arguments are input and the last two are output.

$$\frac{0 \notin \tau_1 \ \text{ and } \ 0 \notin \tau_2}{\mathbf{mrg}(\cdot, \tau_1, \cdot, \tau_2, \cdot, \tau_1 * \tau_2)} \qquad \frac{\mathbf{mrg}(\Omega_1, \tau_1, \Omega_2, \tau_2, \Omega, \tau)}{\mathbf{mrg}(D\Omega_1, \tau_1, D\Omega_2, \tau_2, D\Omega, \tau)}$$

$$\frac{\mathbf{mrg}(\Omega_1, 1\tau_1, \Omega_2, \tau_2, \Omega, \tau)}{\mathbf{mrg}(D\Omega_1, 1\tau_1, \Box\Omega_2, \tau_2'0\tau_2, \Box\Omega, \tau_2'0\tau)}\,(0 \notin \tau_2')$$

$$\frac{\mathbf{mrg}(\Omega_1, \tau_1, \Omega_2, 1\tau_2, \Omega, \tau)}{\mathbf{mrg}(\Box\Omega_1, \tau_1'0\tau_1, D\Omega_2, 1\tau_2, \Box\Omega, \tau_1'0\tau)}\,(0 \notin \tau_1')$$

$$\frac{\mathbf{mrg}(\Omega_1, \tau_1, \Omega_2, \tau_2, \Omega, \tau)}{\mathbf{mrg}(\Box\Omega_1, \tau_1'0\tau_1, \Box\Omega_2, \tau_2'0\tau_2, \Box\Omega, (\tau_1' * \tau_2')0\tau)}\,(0 \notin \tau_i')$$

mrg satisfies the following properties which we will rely upon in the subsequent correctness proofs.

Lemma 29

1. $\mathbf{mrg}(\Omega_1, \tau_1, \Omega_2, \tau_2, \Omega, 1\tau)$ *implies* $\tau_1 = 1\tau_1'$ *and* $\tau_2 = 1\tau_2'$.
2. $\mathbf{mrg}(\Omega_1, \tau_1, \Omega_2, \tau_2, \Omega, \tau)$ *and* $\Box \notin \Omega$ *implies* $\Omega_1 = \Omega_2 = \Omega$.
3. $\mathbf{mrg}(\Omega_1, \tau_1, \Omega_2, \tau_2, \Omega, \tau)$ *implies* $\mathbf{mrg}(\Omega_1, 1 + \tau_1, \Omega_2, 1 + \tau_2, \Omega, 1 + \tau)$.

Proof: By structural induction on the given derivation. □

We also define $\mathbf{mrgL}(\Delta_1, v_1, \Delta_2, v_2, \Delta, v)$, where $v_i \in \{T, F\}$, to merge two compatible linear contexts. We use the standard notations $\vee$ and $\wedge$ for boolean disjunction and conjunction. **mrgL** is also a function where the first four arguments are input and the last two are output.

$$\frac{}{\mathbf{mrgL}(\cdot, v_1, \cdot, v_2, \cdot, v_1 \wedge v_2)}$$

$$\frac{\mathbf{mrgL}(\Delta_1, v_1, \Delta_2, v_2, \Delta, v)}{\mathbf{mrgL}(D\Delta_1, v_1, D\Delta_2, v_2, D\Delta, v)} \qquad \frac{\mathbf{mrgL}(\Delta_1, v_1, \Delta_2, v_2, \Delta, v)}{\mathbf{mrgL}(\Box\Delta_1, v_1, \Box\Delta_2, v_2, \Box\Delta, v)}$$

$$\frac{\mathbf{mrgL}(\Delta_1, T, \Delta_2, v_2, \Delta, v)}{\mathbf{mrgL}(D\Delta_1, T, \Box\Delta_2, v_2, \Box\Delta, v)} \qquad \frac{\mathbf{mrgL}(\Delta_1, v_1, \Delta_2, T, \Delta, v)}{\mathbf{mrgL}(\Box\Delta_1, v_1, D\Delta_2, T, \Box\Delta, v)}$$

We state a few properties of **mrgL** which will be helpful in subsequent proofs.

Lemma 30
Assume $T \sqsupseteq T$ *and* $T \sqsupseteq F$ *and* $F \sqsupseteq F$ *and* $F \not\sqsupseteq T$.

1. $\mathbf{mrgL}(\Delta_1, v_1, \Delta_2, v_2, \Delta, v)$ *implies* $v = v_1 \wedge v_2$ *and* $\Delta_i \sqsupseteq \Delta$.
2. $\mathbf{mrgL}(\Delta_1, v_1, \Delta_2, v_2, \Delta, v)$ *and* $v_i = F$ *implies* $\Delta_i = \Delta$.
3. $\Delta_1 \sqsupseteq \Delta_2$ *and* $v_1 \sqsupseteq v_2$ *implies*
   there exist Δ *and* v *such that* $\mathbf{mrgL}(\Delta_1, v_1, \Delta_2, v_2, \Delta, v)$.
4. $\Delta_2 \sqsupseteq \Delta_1$ *and* $v_2 \sqsupseteq v_1$ *implies*
   there exist Δ *and* v *such that* $\mathbf{mrgL}(\Delta_1, v_1, \Delta_2, v_2, \Delta, v)$.

Proof: By structural induction on given derivation in parts 1 and 2. By structural induction on Δ_1 in part 3. By structural induction on Δ_2 in part 4. □

We now have enough machinery to write down our $\top$-flag derivation system. We have two types of sequent for our $\top$-flag derivations:

$$\begin{array}{l}\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \ \xrightarrow[v\ \tau]{}\ G \\ \Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \ ; \ \Omega_{RI} \backslash \Omega_{RO}) \ \xrightarrow[v\ (\tau_L\ ;\ \tau_R)]{}\ D \gg P\end{array}$$

which correspond exactly to their RMS counterparts with the addition of the previously described output context abstractions, τ, τ_L, τ_R, and a linear $\top$-flag, $v \in \{T, F\}$.

Here are the $\top$-flags derivation rules:

$$\frac{}{\Gamma; \Delta \backslash \Delta; \Omega \backslash \Omega \ \xrightarrow[F\ \cdot]{}\ P \doteq P}\doteq_R \qquad \frac{}{\Gamma; \Delta \backslash \Delta; \Omega \backslash \Omega \ \xrightarrow[F\ \cdot]{}\ \mathbf{1}}\mathbf{1}_R$$

$$\frac{\Gamma; \cdot \backslash \cdot; \cdot \backslash \cdot \ \xrightarrow[v\ \tau]{}\ G}{\Gamma; \Delta \backslash \Delta; \Omega \backslash \Omega \ \xrightarrow[F\ \cdot]{}\ !G}!_R \qquad \frac{\Gamma; \Delta_I \backslash \Delta_O; \cdot \backslash \cdot \ \xrightarrow[v\ \tau]{}\ G}{\Gamma; \Delta_I \backslash \Delta_O; \Omega \backslash \Omega \ \xrightarrow[v\ \cdot]{}\ \text{¡}G}\text{¡}_R$$

$$\frac{\Gamma D; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \ \xrightarrow[v\ \tau]{}\ G}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \ \xrightarrow[v\ \tau]{}\ D \rightarrow G}\rightarrow_R$$

$$\frac{\Gamma; \Delta_I D \backslash \Delta_O \square; \Omega_I \backslash \Omega_O \ \xrightarrow[v\ \tau]{}\ G}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \ \xrightarrow[v\ \tau]{}\ D \multimap G}\multimap_{RF} \qquad \frac{\Gamma; \Delta_I D \backslash \Delta_O D; \Omega_I \backslash \Omega_O \ \xrightarrow[T\ \tau]{}\ G}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \ \xrightarrow[T\ \tau]{}\ D \multimap G}\multimap_{RT}$$

$$\dfrac{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I D\backslash\Omega_O\Box \underset{v\ \ \tau 0\tau'}{\longrightarrow} G}{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \underset{v\ \ \tau}{\longrightarrow} D\twoheadrightarrow G}\twoheadrightarrow_{R0}\quad(0\notin\tau') \qquad \dfrac{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I D\backslash\Omega_O D \underset{v\ \ \tau 1}{\longrightarrow} G}{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \underset{v\ \ \tau 1}{\longrightarrow} D\twoheadrightarrow G}\twoheadrightarrow_{R1}$$

$$\dfrac{\Gamma;\Delta_I\backslash\Delta_O;D\Omega_I\backslash\Box\Omega_O \underset{v\ \ \tau' 0\tau}{\longrightarrow} G}{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \underset{v\ \ \tau}{\longrightarrow} D\rightarrowtail G}\rightarrowtail_{R0}\quad(0\notin\tau') \qquad \dfrac{\Gamma;\Delta_I\backslash\Delta_O;D\Omega_I\backslash D\Omega_O \underset{1\tau\ \ G}{\longrightarrow}}{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \underset{1\tau\ \ D\rightarrowtail G}{\longrightarrow}}\rightarrowtail_{R1}$$

$$\dfrac{\Gamma;\Delta_I\backslash\Delta_M;\Omega_I\backslash\Omega_L\Omega_2 \underset{v_1\ \ \tau_L}{\longrightarrow} G_1 \qquad \Gamma;\Delta_M\backslash\Delta_O;\Omega_2\backslash\Omega_R \underset{v_2\ \ \tau_R}{\longrightarrow} G_2}{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_L\Omega_R \underset{(v_1\vee v_2)\ \ (\tau_L+\tau_R)}{\longrightarrow} G_1\bullet G_2}\bullet_R$$

$(\Box\notin\Omega_2$ and $(\Omega_L = \_\Box$ or $\cdot))$

$$\dfrac{\Gamma;\Delta_I\backslash\Delta_M;\Omega_I\backslash\Omega_2\Omega_R \underset{v_1\ \ \tau_R}{\longrightarrow} G_1 \qquad \Gamma;\Delta_M\backslash\Delta_O;\Omega_2\backslash\Omega_L \underset{v_2\ \ \tau_L}{\longrightarrow} G_2}{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_L\Omega_R \underset{(v_1\vee v_2)\ \ (\tau_L+\tau_R)}{\longrightarrow} G_1\circ G_2}\circ_R$$

$(\Box\notin\Omega_2$ and $(\Omega_R = \Box\_$ or $\cdot))$

$$\dfrac{}{\Gamma;\Delta\backslash\Delta;\Omega\backslash\Omega \underset{T\ \ 1}{\longrightarrow} \top}\top_R$$

$$\dfrac{\Gamma;\Delta_I\backslash\Delta_1;\Omega_I\backslash\Omega_1 \underset{v_1\ \ \tau_1}{\longrightarrow} G_1 \quad \Gamma;\Delta_I\backslash\Delta_2;\Omega_I\backslash\Omega_2 \underset{v_2\ \ \tau_2}{\longrightarrow} G_2 \quad \begin{array}{c}\mathbf{mrg}(\Omega_1,\tau_1,\Omega_2,\tau_2,\Omega_O,\tau)\\ \mathbf{mrgL}(\Delta_1,v_1,\Delta_2,v_2,\Delta_O,v)\end{array}}{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \underset{v\ \ \tau}{\longrightarrow} G_1\,\&\,G_2}\&_R$$

$$\dfrac{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \underset{v\ \ \tau}{\longrightarrow} G_1}{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \underset{v\ \ \tau}{\longrightarrow} G_1\oplus G_2}\oplus_{R1} \qquad \dfrac{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \underset{v\ \ \tau}{\longrightarrow} G_2}{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \underset{v\ \ \tau}{\longrightarrow} G_1\oplus G_2}\oplus_{R2}$$

$$\dfrac{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \underset{v\ \ \tau}{\longrightarrow} G[a/x]}{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \underset{v\ \ \tau}{\longrightarrow} \forall x.\,G}\forall_R^a \qquad \dfrac{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \underset{v\ \ \tau}{\longrightarrow} G[t/x]}{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \underset{v\ \ \tau}{\longrightarrow} \exists x.\,G}\exists_R$$

(a not free in conclusion)

$$\frac{\mathbf{1}; D \gg P \setminus G \qquad \Gamma_L D G_R; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \ \underset{v\ \ \tau}{\longrightarrow}\ G}{\Gamma_L D \Gamma_R; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \ \underset{v\ \ \tau}{\longrightarrow}\ P}\,\mathbf{choice}_{\Gamma}$$

$$\frac{\mathbf{1}; D \gg P \setminus G \qquad \Gamma; \Delta_{LI} \Box \Delta_{RI} \backslash \Delta_O; \Omega_I \backslash \Omega_O \ \underset{v\ \ \tau}{\longrightarrow}\ G}{\Gamma; \Delta_{LI} D \Delta_{RI} \backslash \Delta_O; \Omega_I \backslash \Omega_O \ \underset{v\ \ \tau}{\longrightarrow}\ P}\,\mathbf{choice}_{\Delta}$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{RO}) \ \underset{v\ \ (\tau_L\ ;\ \tau_R)}{\longrightarrow}\ D \gg P}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_{LI} D \Omega_{RI} \backslash \Omega_{LO} \Box \Omega_{RO} \ \underset{v\ \ \tau_L 0 \tau_R}{\longrightarrow}\ P}\,\mathbf{choice}_{\Omega}$$

$$\frac{}{\Gamma; \Delta \backslash \Delta; (\Omega_L \backslash \Omega_L\ ;\ \Omega_R \backslash \Omega_R) \ \underset{F\ \ (\cdot\ ;\ \cdot)}{\longrightarrow}\ P \gg P}\,\mathbf{init}$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{RO}) \ \underset{v\ \ (\tau_L\ ;\ \tau_R)}{\longrightarrow}\ D_1 \gg P}{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{RO}) \ \underset{v\ \ (\tau_L\ ;\ \tau_R)}{\longrightarrow}\ D_1 \mathbin{\&} D_2 \gg P}\,\&_{L1}$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{RO}) \ \underset{v\ \ (\tau_L\ ;\ \tau_R)}{\longrightarrow}\ D_2 \gg P}{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{RO}) \ \underset{v\ \ (\tau_L\ ;\ \tau_R)}{\longrightarrow}\ D_1 \mathbin{\&} D_2 \gg P}\,\&_{L2}$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{RO}) \ \underset{v\ \ (\tau_L\ ;\ \tau_R)}{\longrightarrow}\ D[t/x] \gg P}{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{RO}) \ \underset{v\ \ (\tau_L\ ;\ \tau_R)}{\longrightarrow}\ \forall x.\, D \gg P}\,\forall_L$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{RO}) \ \underset{v\ \ (\tau_L\ ;\ \tau_R)}{\longrightarrow}\ D \gg P \qquad \Gamma; \cdot \backslash \cdot; \cdot \backslash \cdot \ \underset{v'\ \ \tau}{\longrightarrow}\ G}{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{RO}) \ \underset{v\ \ (\tau_L\ ;\ \tau_R)}{\longrightarrow}\ G \to D \gg P}\,\to_L$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_M; (\Omega_{LI} \backslash \Omega_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{RO}) \ \underset{v_1\ \ (\tau_L\ ;\ \tau_R)}{\longrightarrow}\ D \gg P \qquad \Gamma; \Delta_M \backslash \Delta_O; \cdot \backslash \cdot \ \underset{v_2\ \ \tau}{\longrightarrow}\ G}{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{RO}) \ \underset{(v_1 \vee v_2)\ \ (\tau_L\ ;\ \tau_R)}{\longrightarrow}\ G \multimap D \gg P}\,\multimap_L$$

$$\frac{\Gamma;\Delta_I\backslash\Delta_M;(\Omega_{LI}\backslash\Omega_{LO}\;;\;\Omega_{RI}\backslash\Omega_{GI}\Omega_{RO})\;\underset{v_1\;\;(\tau_L\;;\;\tau_R)}{\longrightarrow}\;D\gg P\qquad\Gamma;\Delta_M\backslash\Delta_O;\Omega_{GI}\backslash\Omega_{GO}\;\underset{v_2\;\;\tau}{\longrightarrow}\;G}{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\;;\;\Omega_{RI}\backslash\Omega_{GO}\Omega_{RO})\;\underset{(v_1\vee v_2)\;\;(\tau_L\;;\;\tau+\tau_R)}{\longrightarrow}\;G\twoheadrightarrow D\gg P}\twoheadrightarrow_L$$

$(\Box\notin\Omega_{GI}$ and $(\Omega_{RO}=\Box\_$ or $\cdot))$

$$\frac{\Gamma;\Delta_I\backslash\Delta_M;(\Omega_{LI}\backslash\Omega_{LO}\Omega_{GI}\;;\;\Omega_{RI}\backslash\Omega_{RO})\;\underset{v_1\;\;(\tau_L\;;\;\tau_R)}{\longrightarrow}\;D\gg P\qquad\Gamma;\Delta_M\backslash\Delta_O;\Omega_{GI}\backslash\Omega_{GO}\;\underset{v_1\;\;\tau}{\longrightarrow}\;G}{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\Omega_{GO}\;;\;\Omega_{RI}\backslash\Omega_{RO})\;\underset{(v_1\vee v_2)\;\;(\tau_L+\tau\;;\;\tau_R)}{\longrightarrow}\;G\twoheadrightarrow D\gg P}\rightarrowtail_L$$

$(\Box\notin\Omega_{GI}$ and $(\Omega_{LO}=\_\Box$ or $\cdot))$

We introduce the following notation:

$$\begin{aligned}\#(\Omega)&\equiv\text{ the number of placeholders in }\Omega\\ \#(\tau)&\equiv\text{ the number of 0s in }\tau\end{aligned}$$

$\top$-flags derivations satisfy the following basic properties.

Lemma 31

1. $\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O\;\underset{v\;\;\tau}{\longrightarrow}\;G$ *implies* $\Omega_I\sqsupseteq\Omega_O$ *and* $\#(\Omega_O)=\#(\tau)$.
2. $\Delta_{IL}\sqsupseteq\Delta_{OL}$ *implies*
   $\Gamma;\Delta_{IL}\Box\Delta_{IR}\backslash\Delta_{OL}\Box\Delta_{OR};\Omega_I\backslash\Omega_O\;\underset{v\;\;\tau}{\longrightarrow}\;G$ *iff*
   $\Gamma;\Delta_{IL}D\Delta_{IR}\backslash\Delta_{OL}D\Delta_{OR};\Omega_I\backslash\Omega_O\;\underset{v\;\;\tau}{\longrightarrow}\;G$.
3. $\Omega_{IL}\sqsupseteq\Omega_{OL}$ *and* $\Box\notin\Omega$ *implies*
   $\Gamma;\Delta_I\backslash\Delta_O;\Omega_{IL}\Omega\Omega_{IR}\backslash\Omega_{OL}\Omega\Omega_{OR}\;\underset{v\;\;\tau}{\longrightarrow}\;G$ *iff*
   $\Gamma;\Delta_I\backslash\Delta_O;\Omega_{IL}\Omega_{IR}\backslash\Omega_{OL}\Omega_{OR}\;\underset{v\;\;\tau}{\longrightarrow}\;G$.
4. $\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\;;\;\Omega_{RI}\backslash\Omega_{RO})\;\underset{v\;\;(\tau_L\;;\;\tau_R)}{\longrightarrow}\;D\gg P$ *implies*
   $\Omega_{LI}\sqsupseteq\Omega_{LO}$ *and* $\Omega_{RI}\sqsupseteq\Omega_{RO}$ *and*
   $\#(\Omega_{LO})=\#(\tau_L)$ *and* $\#(\Omega_{RO})=\#(\tau_R)$.
5. $\Delta_{IL}\sqsupseteq\Delta_{OL}$ *implies*
   $\Gamma;\Delta_{IL}\Box\Delta_{IR}\backslash\Delta_{OL}\Box\Delta_{OR};(\Omega_{LI}\backslash\Omega_{LO}\;;\;\Omega_{RI}\backslash\Omega_{RO})\;\underset{v\;\;(\tau_L\;;\;\tau_R)}{\longrightarrow}\;D\gg P$ *iff*
   $\Gamma;\Delta_{IL}D\Delta_{IR}\backslash\Delta_{OL}D\Delta_{OR};(\Omega_{LI}\backslash\Omega_{LO}\;;\;\Omega_{RI}\backslash\Omega_{RO})\;\underset{v\;\;(\tau_L\;;\;\tau_R)}{\longrightarrow}\;D\gg P$.

6. $\Omega_{LIL} \sqsupseteq \Omega_{LOL}$ *and* $\Box \notin \Omega$ *implies*

$$\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LIL}\Omega\Omega_{LIR}\backslash\Omega_{LOL}\Omega\Omega_{LOR}\ ;\ \Omega_{RI}\backslash\Omega_{RO}) \underset{v\ (\tau_L\ ;\ \tau_R)}{\longrightarrow} D \gg P \ \textit{ iff}$$
$$\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LIL}\Omega_{LIR}\backslash\Omega_{LOL}\Omega_{LOR}\ ;\ \Omega_{RI}\backslash\Omega_{RO}) \underset{v\ (\tau_L\ ;\ \tau_R)}{\longrightarrow} D \gg P.$$

7. $\Omega_{RIL} \sqsupseteq \Omega_{ROL}$ *and* $\Box \notin \Omega$ *implies*

$$\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RIL}\Omega\Omega_{RIR}\backslash\Omega_{ROL}\Omega\Omega_{ROR}) \underset{v\ (\tau_L\ ;\ \tau_R)}{\longrightarrow} D \gg P \ \textit{ iff}$$
$$\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RIL}\Omega_{RIR}\backslash\Omega_{ROL}\Omega_{ROR}) \underset{v\ (\tau_L\ ;\ \tau_R)}{\longrightarrow} D \gg P.$$

Proof: Each part is proved by straightforward structural induction on the given derivation. □

9.2 Correctness of $\top$-flags System

In order to prove the correctness of the $\top$-flags derivation system with respect to the resource management system of Chapter 8, we need to relate $\top$-flags output contexts, and their associated flag lists, to RMS output contexts. Due to the lazy treatment of $\top$, one $\top$-flags context can be related to many RMS contexts. We define the following relation between $\top$-flags output contexts and RMS output contexts:

$$\frac{0 \notin \tau}{\Phi(\cdot,\tau,\cdot)} \qquad \frac{\Phi(\Omega,\tau,\Omega')}{\Phi(D\Omega,\tau,D\Omega')} \qquad \frac{\Phi(\Omega,1\tau,\Omega')}{\Phi(D\Omega,1\tau,\Box\Omega')} \qquad \frac{\Phi(\Omega,\tau,\Omega')}{\Phi(\Box\Omega,\tau'0\tau,\Box\Omega')}\,(0 \notin \tau')$$

Φ satisfies the following properties which we rely upon to prove correctness of the $\top$-flags derivations.

Lemma 32

1. $\Phi(\Omega,\tau,\Omega')$ *implies* $\Omega \sqsupseteq \Omega'$.

2. $\Phi(\Omega_L,\tau_L,\Omega'_L)$ *and* $\Phi(\Omega_R,\tau_R,\Omega'_R)$ *implies* $\Phi(\Omega_L\Omega_R,\tau_L+\tau_R,\Omega'_L\Omega'_R)$.

3. $\Phi(\Omega_L\Omega_R,\tau_L\tau_R,\Omega')$ *and* $\#(\Omega_L) = \#(\tau_L)$
   and $(\Omega_R = \Box\_ \ \textit{or}\ \cdot)$ *and* $(\tau_R = 0\_ \ \textit{or}\ \cdot)$ *implies*
   there exist Ω'_L *and* Ω'_R *such that*
   $\Omega' = \Omega'_L\Omega'_R$ *and* $\Phi(\Omega_L,\tau_L,\Omega'_L)$ *and* $\Phi(\Omega_R,\tau_R,\Omega'_R)$.

4. $\Phi(\Omega_L\Omega_R, \tau_L\tau_R, \Omega')$ *and* $\#(\Omega_R) = \#(\tau_R)$
   and $(\Omega_L = \_\Box$ *or* $\cdot)$ *and* $(\tau_L = \_0$ *or* $\cdot)$ *implies*
   there exist Ω'_L *and* Ω'_R *such that*
   $\Omega' = \Omega'_L\Omega'_R$ *and* $\Phi(\Omega_L, \tau_L, \Omega'_L)$ *and* $\Phi(\Omega_R, \tau_R, \Omega'_R)$.

5. $\Phi(\Omega, \tau, \Omega'_L\Omega'_R)$ *and* $\Box \notin \Omega$ *implies*
   there exist Ω_L *and* Ω_R *such that*
   $\Omega = \Omega_L\Omega_R$ *and* $\Phi(\Omega_L, \tau, \Omega'_L)$ *and* $\Phi(\Omega_R, \tau, \Omega'_R)$.

Proof: Each part is proved by induction on the given derivation. □

Additionally, Φ and **mrg** interact in the following manner which allows our $\&_R$ rule to correctly match the corresponding RMS rule.

Lemma 33

1. $\Phi(\Omega, \tau, \Omega')$ *and* $\mathbf{mrg}(\Omega_1, \tau_1, \Omega_2, \tau_2, \Omega, \tau)$ *implies*
   $\Phi(\Omega_1, \tau_1, \Omega')$ *and* $\Phi(\Omega_2, \tau_2, \Omega')$.

2. $\Phi(\Omega_1, \tau_1, \Omega')$ *and* $\Phi(\Omega_2, \tau_2, \Omega')$ *implies*
   there exist Ω *and* τ *such that*
   $\Phi(\Omega, \tau, \Omega')$ *and* $\mathbf{mrg}(\Omega_1, \tau_1, \Omega_2, \tau_2, \Omega, \tau)$.

Proof:

Part 1: By induction on $\Phi(\Omega, \tau, \Omega')$ examining cases for $\mathbf{mrg}(\Omega_1, \tau_1, \Omega_2, \tau_2, \Omega, \tau)$ making use of lemma 29.

Part 2: By induction on $\Phi(\Omega_1, \tau_1, \Omega')$ examining cases for $\Phi(\Omega_2, \tau_2, \Omega')$.

□

We now prove the soundness of $\top$-flags derivations wrt to RMS derivations. This proof is complicated by the fact that formulas in a $\top$-flags context can be translated to □s and the deterministic context splits (in the $\twoheadrightarrow_L$, $\rightarrowtail_L$, $\bullet_R$, and $\circ_R$ rules) depend upon □s. Consider the rule

$$\frac{\Gamma; \Delta_I\backslash\Delta_M; (\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{GI}\Omega_{RO}) \underset{v_1\ (\tau_L\ ;\ \tau_R)}{\longrightarrow} D \gg P \qquad \Gamma; \Delta_M\backslash\Delta_O; \Omega_{GI}\backslash\Omega_{GO} \underset{v_2\ \ \tau}{\longrightarrow} G}{\Gamma; \Delta_I\backslash\Delta_O; (\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{GO}\Omega_{RO}) \underset{(v_1 \vee v_2)\ \ (\tau_L\ ;\ \tau + \tau_R)}{\longrightarrow} G \twoheadrightarrow D \gg P} \twoheadrightarrow_L$$

$(\Box \notin \Omega_{GI}$ and $(\Omega_{RO} = \Box\_$ or $\cdot))$

The output context, $\Omega_{GI}\Omega_{RO}$ will be translated to some context Ω'_O. We will then deterministically split Ω'_O into $\Omega'_{GI}\Omega'_{RO}$ where $\Box \notin \Omega'_{GI}$ and $\Omega'_{RO} = \Box\_$ or $\cdot$. However, if τ_R begins with a 1, Ω_{GI} might be longer than Ω'_{GI} since any formula in Ω_{GI} could have been translated to a $\Box$. In such a situation, we will have cut out of Ω_{GI}, in the second premise, the formulas which were consumed by the $\top$, corresponding to the opening 1 in τ_R, in order to construct the matching RMS rule instantiation. It turns out this is always possible to do, although the analysis is quite tedious.

We will consider four cases, two for each type of sequent depending on the value of the linear $\top$ flag.

Theorem 34 (Soundness of $\top$-flags)

1. $\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \underset{F\ \tau}{\longrightarrow} G$ *and* $\Phi(\Omega_O,\tau,\Omega'_O)$
   implies $\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega'_O \longrightarrow G$.

2. $\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \underset{T\ \tau}{\longrightarrow} G$ *and* $\Phi(\Omega_O,\tau,\Omega'_O)$ *and* $\Delta_O \sqsupseteq \Delta'_O$
   implies $\Gamma;\Delta_I\backslash\Delta'_O;\Omega_I\backslash\Omega'_O \longrightarrow G$.

3. $\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{RO}) \underset{F\ (\tau_L\ ;\ \tau_R)}{\longrightarrow} D \gg P$
   and $\Phi(\Omega_{LO},\tau_L,\Omega'_{LO})$ *and* $\Phi(\Omega_{RO},\tau_R,\Omega'_{RO})$
   implies $\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega'_{LO}\ ;\ \Omega_{RI}\backslash\Omega'_{RO}) \longrightarrow D \gg P$.

4. $\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{RO}) \underset{T\ (\tau_L\ ;\ \tau_R)}{\longrightarrow} D \gg P$
   and $\Delta_O \sqsupseteq \Delta'_O$ *and* $\Phi(\Omega_{LO},\tau_L,\Omega'_{LO})$ *and* $\Phi(\Omega_{RO},\tau_R,\Omega'_{RO})$
   implies $\Gamma;\Delta_I\backslash\Delta'_O;(\Omega_{LI}\backslash\Omega'_{LO}\ ;\ \Omega_{RI}\backslash\Omega'_{RO}) \longrightarrow D \gg P$.

Proof: By structural induction on given derivation. We give representative cases.

case: $$\frac{}{\Gamma;\Delta\backslash\Delta;\Omega\backslash\Omega \underset{T\ 1}{\longrightarrow} \top}\top_R$$
Then

$\Phi(\Omega,1,\Omega')$ and $\Delta \sqsupseteq \Delta'$ — assumptions
$\Omega \sqsupseteq \Omega'$ — Lemma 32.1
$\Gamma;\Delta\backslash\Delta';\Omega\backslash\Omega' \longrightarrow \top$ — $\top_R$

case:

$$\frac{\Gamma;\Delta_I\backslash\Delta_1;\Omega_I\backslash\Omega_1 \underset{T\ \tau_1}{\longrightarrow} G_1 \quad \Gamma;\Delta_I\backslash\Delta_2;\Omega_I\backslash\Omega_2 \underset{T\ \tau_2}{\longrightarrow} G_2 \quad \begin{array}{c}\mathbf{mrg}(\Omega_1,\tau_1,\Omega_2,\tau_2,\Omega,\tau)\\ \mathbf{mrgL}(\Delta_1,T,\Delta_2,T,\Delta_O,T)\end{array}}{\Gamma;\Delta_I\backslash\Delta_O;\Omega_I\backslash\Omega_O \underset{T\ \tau}{\longrightarrow} G_1 \,\&\, G_2}\,\&_R$$

Then

$\Phi(\Omega_O,\tau,\Omega'_O)$ and $\Delta_O \sqsupseteq \Delta'_O$	assumption
$\Phi(\Omega_1,\tau_1,\Omega'_O)$ and $\Phi(\Omega_2,\tau_2,\Omega'_O)$	Lemma 33
$\Delta_1 \sqsupseteq \Delta_O$ and $\Delta_2 \sqsupseteq \Delta_O$	Lemma 30
Note $\Delta_1 \sqsupseteq \Delta'_O$ and $\Delta_2 \sqsupseteq \Delta'_O$	
$\Gamma;\Delta_I\backslash\Delta'_O;\Omega_I\backslash\Omega'_O \longrightarrow G_1$ and $\Gamma;\Delta_I\backslash\Delta'_O;\Omega_I\backslash\Omega'_O \longrightarrow G_2$	ind. hyp.
$\Gamma;\Delta_I\backslash\Delta'_O;\Omega_I\backslash\Omega'_O \longrightarrow G_1 \,\&\, G_2$	rule $\&_R$

case:

$$\frac{}{\Gamma;\Delta\backslash\Delta;(\Omega_L\backslash\Omega_L\ ;\ \Omega_R\backslash\Omega_R) \underset{F\ (\cdot\,;\,\cdot)}{\longrightarrow} P \gg P}\,\mathbf{init}$$

Then

$\Phi(\Omega_L,\cdot,\Omega'_L)$ and $\Phi(\Omega_R,\cdot,\Omega'_R)$	assumptions
$\Omega_L = \Omega'_L$ and $\Omega_R = \Omega'_R$	inspection of Φ
$\Gamma;\Delta\backslash\Delta;(\Omega_L\backslash\Omega_L\ ;\ \Omega_R\backslash\Omega_R) \longrightarrow P \gg P$	**init**

case:

$$\frac{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{RO}) \underset{F\ (\tau_L\,;\,\tau_R)}{\longrightarrow} D \gg P}{\Gamma;\Delta_I\backslash\Delta_O;\Omega_{LI}D\Omega_{RI}\backslash\Omega_{LO}\Box\Omega_{RO} \underset{F\ \tau_L 0\tau_R}{\longrightarrow} P}\,\mathbf{choice}_\Omega$$

Then

$\Phi(\Omega_{LO}\Box\Omega_{RO},\tau_L 0\tau_R,\Omega'_O)$	assumption
$\#(\tau_L) = \#(\Omega_{LO})$ and $\#(\tau_R) = \#(\Omega_{RO})$	Lemma 31
$\Omega'_O = \Omega'_{LO}\Omega'_R$ and $\Phi(\Omega_{LO},\tau_L,\Omega'_{LO})$ and $\Phi(\Box\Omega_{RO},0\tau_R,\Omega'_R)$	Lemma 32.3
$\Omega'_R = \Box\Omega'_{RO}$ and $\Phi(\Omega_{RO},\tau_R,\Omega'_{RO})$	inversion on Φ
$\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{RO}) \longrightarrow D \gg P$	ind. hyp.
$\Gamma;\Delta_I\backslash\Delta_O;\Omega_{LI}D\Omega_{RI}\backslash\Omega_{LO}\Box\Omega_{RO} \longrightarrow P$	**choice**$_\Omega$

case:

$$\frac{\Gamma;\Delta_I\backslash\Delta_M;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{GI}\Omega_{RO})\ \underset{F\ (\tau_L\ ;\ \tau_R)}{\longrightarrow}\ D \gg P \qquad \Gamma;\Delta_M\backslash\Delta_O;\Omega_{GI}\backslash\Omega_{GO}\ \underset{F\ \tau}{\longrightarrow}\ G}{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{GO}\Omega_{RO})\ \underset{F\ (\tau_L\ ;\ \tau\tau_R)}{\longrightarrow}\ G \twoheadrightarrow D \gg P}\twoheadrightarrow_L$$

$(\Box \notin \Omega_{GI}$ and $(\Omega_{RO} = \Box\_$ or $\cdot)$ and $\tau_R \neq 1\tau')$

$\Phi(\Omega_{LO},\tau_L,\Omega'_{LO})$ and $\Phi(\Omega_{GO}\Omega_{RO},\tau\tau_R,\Omega'_O)$	assumptions
Note $\tau_R = 0\_$ or $\cdot$	
$\#(\tau_R) = \#(\Omega_{GI}\Omega_{RO}) = \#(\Omega_{RO})$ and $\#(\tau) = \#(\Omega_{GO})$	Lemma 31
$\Omega'_O = \Omega'_{GO}\Omega'_{RO}$ and $\Phi(\Omega_{GO},\tau,\Omega'_{GO})$ and $\Phi(\Omega_{RO},\tau_R,\Omega'_{RO})$ for some Ω'_{GO} and Ω'_{RO}	Lemma 32.3
$\Gamma;\Delta_M\backslash\Delta_O;\Omega_{GI}\backslash\Omega'_{GO} \ \longrightarrow\ G$	ind. hyp.(1)
$\Phi(\Omega_{GI}\Omega_{RO},\tau_R,\Omega_{GI}\Omega'_{RO})$	$\Box \notin \Omega_{GI}$
$\Gamma;\Delta_I\backslash\Delta_M;(\Omega_{LI}\backslash\Omega'_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{GI}\Omega'_{RO})\ \longrightarrow\ D \gg P$	ind. hyp.(3)
$\Omega'_{RO} = \Box\_$ or $\cdot$	inversion on Φ
$\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega'_{LO}\ ;\ \Omega_{RI}\backslash\Omega'_{GO}\Omega'_{RO})\ \longrightarrow\ G \twoheadrightarrow D \gg P$	rule $\twoheadrightarrow_L$

case:

$$\frac{\Gamma;\Delta_I\backslash\Delta_M;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{GI}\Omega_{RO})\ \underset{F\ (\tau_L\ ;\ 1\tau_R)}{\longrightarrow}\ D \gg P \qquad \Gamma;\Delta_M\backslash\Delta_O;\Omega_{GI}\backslash\Omega_{GO}\ \underset{F\ \tau}{\longrightarrow}\ G}{\Gamma;\Delta_I\backslash\Delta_O;(\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{GO}\Omega_{RO})\ \underset{F\ (\tau_L\ ;\ \tau+1\tau_R)}{\longrightarrow}\ G \twoheadrightarrow D \gg P}\twoheadrightarrow_L$$

$(\Box \notin \Omega_{GI}$ and $(\Omega_{RO} = \Box\_$ or $\cdot))$

Then

$\Phi(\Omega_{LO},\tau_L,\Omega'_{LO})$ and $\Phi(\Omega_{GO}\Omega_{RO},\tau+1\tau_R,\Omega'_O)$	assumptions
$\#(\Omega_{GO}) = \#(\tau)$ and $\#(\Omega_{GI}\Omega_{RO}) = \#(\Omega_{RO}) = \#(1\tau_R)$	lemma 31
$\Omega'_O = \Omega'_{GO}\Omega'_{RO}$ and $\Phi(\Omega_{GO},\tau+1,\Omega'_{GO})$ and $\Phi(\Omega_{RO},\tau_R,\Omega'_{RO})$	Lemma 32.3
Let $\Omega_{GOL}\Omega_{GIR} = \Omega_{GO}$ where $(\Omega_{GOL} = \_\Box$ or $\cdot)$ and $\Box \notin \Omega_{GIR}$	
Let $\tau_{GL}\tau_{GR} = \tau$ where $(\tau_{GL} = \_0$ or $\cdot)$ and $0 \notin \tau_{GR}$	
$\#(\Omega_{GO}) = \#(\Omega_{GOL}) = \#(\tau+1)$	$\Box \notin \Omega_{GIR}$
Note $\tau_{GR} + 1 = 1$	

$\Omega'_{GO} = \Omega'_{GOL}\Omega'_{GIR}$ and $\Phi(\Omega_{GOL}, \tau_{GL}, \Omega'_{GOL})$ and $\Phi(\Omega_{GIR}, 1, \Omega'_{GIR})$	
for some Ω'_{GOL} and Ω'_{GIR}	Lemma 32.4
Let $\Omega'_{GIRL}\Omega'_{GIRR} = \Omega'_{GIR}$ where $\Box \notin \Omega'_{GIRL}$ and $(\Omega'_{GIRR} = \Box\_$ or $\cdot)$	
$\Omega_{GIR} = \Omega_{GIRL}\Omega_{GIRR}$ and $\Phi(\Omega_{GIRL}, 1, \Omega'_{GIRL})$ and $\Phi(\Omega_{GIRR}, 1, \Omega'_{GIRR})$	
for some Ω_{GIRL} and Ω_{GIRR}	Lemma 32.5
$\Phi(\Omega_{GIRL}, \tau_{GR}, \Omega'_{GIRL})$ and $\Omega_{GIRL} = \Omega'_{GIRL}$	$\Box \notin \Omega'_{GIRL}, 0 \notin \tau_{GR}$
Note $\tau_{GL} + \tau_{GR} = \tau$	
$\Phi(\Omega_{GOL}\Omega_{GIRL}, \tau, \Omega'_{GOL}\Omega_{GIRL})$	Lemma 32.2
$\Omega_{GI} = \Omega_{GIL}\Omega_{GIRL}\Omega_{GIRR}$ and $\Omega_{GIL} \sqsupseteq \Omega_{GOL}$ for some Ω_{GIL}	$\Omega_{GI} \sqsupseteq \Omega_{GO}$
Note we may rewrite the second premise of the given derivation as	
$\Gamma; \Delta_M \backslash \Delta_O; \Omega_{GIL}\Omega_{GIRL}\Omega_{GIRR} \backslash \Omega_{GOL}\Omega_{GIRL}\Omega_{GIRR} \xrightarrow[F\ \tau]{} G$	
$\Gamma; \Delta_M \backslash \Delta_O; \Omega_{GIL}\Omega_{GIRL} \backslash \Omega_{GOL}\Omega_{GIRL} \xrightarrow[F\ \tau]{} G$	Lemma 31
$\Gamma; \Delta_M \backslash \Delta_O; \Omega_{GIL}\Omega_{GIRL} \backslash \Omega'_{GOL}\Omega_{GIRL} \longrightarrow G$	ind. hyp.(1)
$\Phi(\Omega_{GIL}\Omega_{GIRL}, \cdot, \Omega_{GIL}\Omega_{GIRL})$	$\Box \notin \Omega_{GIL}\Omega_{GIRL}$
Note $1 + \tau_R = 1\tau_R$	
$\Phi(\Omega_{GIL}\Omega_{GIRL}\Omega_{GIRR}\Omega_{RO}, 1\tau_R, \Omega_{GIL}\Omega_{GIRL}\Omega'_{GIRR}\Omega'_{RO})$	Lemma 32.2
$\Gamma; \Delta_I \backslash \Delta_M; (\Omega_{LI} \backslash \Omega'_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{GIL}\Omega_{GIRL}\Omega'_{GIRR}\Omega'_{RO}) \longrightarrow D \gg P$	ind. hyp.(3)
$\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega'_{LO}\ ;\ \Omega_{RI} \backslash \Omega'_{GOL}\Omega_{GIRL}\Omega'_{GIRR}\Omega'_{RO}) \longrightarrow G \twoheadrightarrow D \gg P$	rule $\twoheadrightarrow_L$

$\Box$

We now prove completeness of $\top$-flags derivations wrt RMS derivations.

Theorem 35 (Completeness of $\top$-flags)

1. $\Gamma; \Delta'_I \backslash \Delta'_O; \Omega'_I \backslash \Omega'_O \longrightarrow G$ *implies*
   there exist Δ_O, Ω_O, *and* τ *such that*
   $\Delta_O \sqsupseteq \Delta'_O$ *and* $\Phi(\Omega_O, \tau, \Omega'_O)$ *and*
   $(\Gamma; \Delta'_I \backslash \Delta'_O; \Omega'_I \backslash \Omega_O \xrightarrow[F\ \tau]{} G$ *or* $\Gamma; \Delta'_I \backslash \Delta_O; \Omega'_I \backslash \Omega_O \xrightarrow[T\ \tau]{} G)$.

2. $\Gamma; \Delta'_I \backslash \Delta'_O; (\Omega'_{LI} \backslash \Omega'_{LO}\ ;\ \Omega'_{RI} \backslash \Omega'_{RO}) \longrightarrow D \gg P$ *implies*
   there exist Δ_O, Ω_{LO}, Ω_{RO}, τ_L, *and* τ_R *such that*
   $\Delta_O \sqsupseteq \Delta'_O$ *and* $\Phi(\Omega_{LO}, \tau_L, \Omega'_{LO})$ *and* $\Phi(\Omega_{RO}, \tau_R, \Omega'_{RO})$ *and*
   $(\Gamma; \Delta'_I \backslash \Delta'_O; (\Omega'_{LI} \backslash \Omega_{LO}\ ;\ \Omega'_{RI} \backslash \Omega_{RO}) \xrightarrow[F\ (\tau_L\ ;\ \tau_R)]{} D \gg P$ *or*
   $\Gamma; \Delta'_I \backslash \Delta_O; (\Omega'_{LI} \backslash \Omega_{LO}\ ;\ \Omega'_{RI} \backslash \Omega_{RO}) \xrightarrow[T\ (\tau_L\ ;\ \tau_R)]{} D \gg P)$.

Proof: By structural induction on the given derivation. We give representative cases.

case:
$$\frac{\Gamma;\Delta'_I\backslash\Delta'_O;\Omega'_I\backslash\Omega'_O \ \longrightarrow\ G_1 \quad \Gamma;\Delta'_I\backslash\Delta'_O;\Omega'_I\backslash\Omega'_O \ \longrightarrow\ G_2}{\Gamma;\Delta'_I\backslash\Delta'_O;\Omega'_I\backslash\Omega'_O \ \longrightarrow\ G_1 \,\&\, G_2}\,\&_R$$

$\Gamma;\Delta'_I\backslash\Delta_1;\Omega'_I\backslash\Omega_1 \ \underset{v_1\ \tau_1}{\longrightarrow}\ G_1$ and $\Phi(\Omega_1,\tau_1,\Omega'_O)$ and $\Delta_1 \sqsupseteq \Delta'_O$

and (if $v_1 = F$ then $\Delta_1 = \Delta'_O$) ind. hyp.

$\Gamma;\Delta'_I\backslash\Delta_2;\Omega'_I\backslash\Omega_2 \ \underset{v_2\ \tau_2}{\longrightarrow}\ G_2$ and $\Phi(\Omega_2,\tau_2,\Omega'_O)$ and $\Delta_2 \sqsupseteq \Delta'_O$

and (if $v_2 = F$ then $\Delta_2 = \Delta'_O$) ind. hyp.

$\mathbf{mrg}(\Omega_1,\tau_1,\Omega_2,\tau_2,\Omega_O,\tau)$ and $\Phi(\Omega_O,\tau,\Omega'_O)$

for some Ω_O and τ Lemma 33

$\mathbf{mrgL}(\Delta_1,v_1,\Delta_2,v_2,\Delta_O,v)$ for some Δ_O and v Lemma 30

$\Gamma;\Delta'_I\backslash\Delta_O;\Omega'_I\backslash\Omega_O \ \underset{v\ \tau}{\longrightarrow}\ G_1 \,\&\, G_2$ rule $\&_R$

Note if $v = F$ then $\Delta_O = \Delta'_O$ Lemma 30

case:
$$\frac{}{\Gamma;\Delta'\backslash\Delta';(\Omega'_L\backslash\Omega'_L\ ;\ \Omega'_R\backslash\Omega'_R) \ \longrightarrow\ P \gg P}\,\mathbf{init}$$

Then

$\Gamma;\Delta'\backslash\Delta';(\Omega'_L\backslash\Omega'_L\ ;\ \Omega'_R\backslash\Omega'_R) \ \underset{F\ (\cdot\,;\,\cdot)}{\longrightarrow}\ P \gg P$ **init**

case:
$$\frac{\Gamma;\Delta'_I\backslash\Delta'_O;(\Omega'_{LI}\backslash\Omega'_{LO}\ ;\ \Omega'_{RI}\backslash\Omega'_{RO}) \ \longrightarrow\ D \gg P}{\Gamma;\Delta'_I\backslash\Delta'_O;\Omega'_{LI}D\Omega'_{RI}\backslash\Omega'_{LO}\Box\Omega'_{RO} \ \longrightarrow\ P}\,\mathbf{choice}_\Omega$$

Then

$\Gamma;\Delta'_I\backslash\Delta_O;(\Omega'_{LI}\backslash\Omega_{LO}\ ;\ \Omega'_{RI}\backslash\Omega_{RO}) \ \underset{v\ (\tau_L\,;\,\tau_R)}{\longrightarrow}\ D \gg P$

and $\Phi(\Omega_{LO},\tau_L,\Omega'_{LO})$ and $\Phi(\Omega_{RO},\tau_R,\Omega'_{RO})$ and $\Delta_O \sqsupseteq \Delta'_O$

and (if $v = F$ then $\Delta_O = \Delta'_O$) ind. hyp.

$\Phi(\Omega_{LO}\Box\Omega_{RO},\tau_L 0 \tau_R,\Omega'_{LO}\Box\Omega'_{RO})$ Lemma 32.2

$\Gamma;\Delta'_I\backslash\Delta_O;\Omega'_{LI}D\Omega'_{RI}\backslash\Omega_{LO}\Box\Omega_{RO} \ \underset{v\ \tau_L 0\tau_R}{\longrightarrow}\ P$ $\mathbf{choice}_\Omega$

case:

$$\frac{\Gamma; \Delta'_I \backslash \Delta'_M; (\Omega'_{LI} \backslash \Omega'_{LO} \; ; \; \Omega'_{RI} \backslash \Omega'_{GI}\Omega'_{RO}) \; \longrightarrow \; D \gg P \qquad \Gamma; \Delta'_M \backslash \Delta'_O; \Omega'_{GI} \backslash \Omega'_{GO} \; \longrightarrow \; G}{\Gamma; \Delta'_I \backslash \Delta'_O; (\Omega'_{LI} \backslash \Omega'_{LO} \; ; \; \Omega'_{RI} \backslash \Omega'_{GO}\Omega'_{RO}) \; \longrightarrow \; G \twoheadrightarrow D \gg P} \twoheadrightarrow_L$$

$(\Box \notin \Omega'_{GI}$ and $(\Omega'_{RO} = \Box\Omega'$ or $\cdot))$

$\Gamma; \Delta'_I \backslash \Delta_M; (\Omega'_{LI} \backslash \Omega_{LO} \; ; \; \Omega'_{RI} \backslash \Omega_R) \underset{v \;\; (\tau_L \; ; \; \tau_R)}{\longrightarrow} D \gg P$

and $\Phi(\Omega_{LO}, \tau_L, \Omega'_{LO})$ and $\Phi(\Omega_R, \tau_R, \Omega'_{GI}\Omega'_{RO})$

and $\Delta_M \sqsupseteq \Delta'_M$ and (if $v = F$ then $\Delta_M = \Delta'_M$) ind. hyp.

Let $\Omega_{GI}\Omega_{RO} = \Omega_R$ where $\Box \notin \Omega_{GI}$ and $(\exists \Omega. \; \Omega_{RO} = \Box\Omega$ or $\cdot)$

$\Omega_R \sqsupseteq \Omega'_{GI}\Omega'_{RO}$ Lemma 32.1

$\Omega'_{GI}\Omega'_{RO} = \Omega'_{GI}\Omega'_{ROL}\Omega'_{ROR}$ and $\Omega_{RO} \sqsupseteq \Omega'_{ROR}$ and $\Omega_{GI} \sqsupseteq \Omega'_{GI}\Omega'_{ROL}$

and $\Omega_{GI} = \Omega'_{GI}\Omega_{GIR}$ and $\Omega_{GIR} \sqsupseteq \Omega'_{ROL}$ properties of $\sqsupseteq$

$\Gamma; \Delta'_M \backslash \Delta_O; \Omega'_{GI} \backslash \Omega_{GO} \underset{v' \;\; \tau}{\longrightarrow} G$ and $\Phi(\Omega_{GO}, \tau, \Omega'_{GO})$

and $\Delta_O \sqsupseteq \Delta'_O$ and (if $v' = F$ then $\Delta_O = \Delta'_O$) ind. hyp.

Suppose $\Delta_M = D'$ and $\Delta'_M = \Box$

$\Delta_O = \Box$ $\Delta'_M \sqsupseteq \Delta_O$

$\Gamma; \Delta_M \backslash D'; \Omega'_{GI} \backslash \Omega_{GO} \underset{v' \;\; \tau}{\longrightarrow} G$ Lemma 31.1

Thus we can generalize the above argument to show iterating

$\Gamma; \Delta_M \backslash \Delta_{O1}; \Omega'_{GI} \backslash \Omega_{GO} \underset{v' \;\; \tau}{\longrightarrow} G$ and $\Delta_{O1} \sqsupseteq \Delta_O$ for some Δ_{O1} Lemma 31.1

$\Gamma; \Delta'_I \backslash \Delta_{O1}; (\Omega'_{LI} \backslash \Omega_{LO} \; ; \; \Omega'_{RI} \backslash \Omega_{GO}\Omega_{RO}) \underset{(v \vee v') \;\; (\tau_L \; ; \; \tau_R)}{\longrightarrow} D \gg P$ $\twoheadrightarrow_L$

Note if $v \vee v' = F$ then $\Delta_{O1} = \Delta'_O$.

□

We now have a derivation system for extended uniform ordered linear logic which does not require any non-deterministic splitting of resources. The non-determinism left in this system resides in the choice of formula to focus upon, the $\oplus_R$ rules, and the choice of term t in the $\forall_L$ and $\exists_R$ rules. All of this non-determinism can be handled in the standard ways– choosing an arbitrary order to choose clauses and try goals; and using unification to delay picking an actually term– to give us a derivation system whose direct implementation is reasonably efficient. Thus we can base a logic programming interpreter on the $\top$-flags system.

However, there are still inefficiencies in the $\top$-flags system which can greatly slow down proof search. Specifically, the techniques reported in [10] and [33] for improving linear logic proof search can be extended to ordered linear logic and incorporated into the $\top$-flags system. We will delay this extension until Chapter 11, and first discuss logic programming in ordered linear logic.

Chapter 10

Ordered Linear Logic Programming

Before linear logic programming languages, any state associated with logic programs had to be represented as a term (typically a list) and threaded through the program in much the same manner as state must be simulated in a purely functional language[1]. With the addition of linear hypotheses to logic programs, state could be implicitly represented by the linear context and thus moved from the term-level to the formula-level. While this situation certainly allows for more elegant programs, it also allows for more efficient programs as compiler technology matures. For example, Hodas and Tamura [28] describe a theorem prover, originally in Prolog, which becomes both more elegant and more efficient when rewritten in Lolli. Similarly, adding ordered hypotheses to a logic programming language allows some kinds of data structures (*e.g.*, stacks) to be moved from the term-level to the formula-level.

In this chapter, we introduce Olli, an ordered extension of Lolli, which is based on the $\top$-flags derivation system of chapter 9. After attending to the few details needed to turn the $\top$-flags system into a logic programming interpreter, we will present a series of example Olli programs.

[1] One could also resort to extra-logical features such as assert and retract; however, these constructs destroy the logical interpretation of a logic program, in addition to being extremely difficult to correctly use.

10.1 Olli

As mentioned at the end of chapter 9, the $\top$-flags derivation system is deterministic enough to allow a reasonable direct implementation which gives us an ordered linear logic programming interpreter. We will call the resulting ordered linear logic programming language Olli. Just as ordered linear logic is a conservative extension of intuitionistic linear logic, Olli is a conservative extension of (pure) Lolli– the formula language of Lolli is a subset of that for Olli.

In order to turn the $\top$-flag system into a logic programming interpreter, we need to deal with the two remaining sources of non-determinism in the system– the choice of a term for instantiating quantified variables; and the choices of a formula to focus on when the goal is atomic, and a sub-goal to pursue when the goal is a disjunction. We employ the standard treatment for both choices. We first alter the $\forall_L$ and $\exists_R$ rules to substitute new logic variables, rather than terms, and alter the $\doteq_R$ rule to unify the two terms rather than just check for equality. We then assume that disjunctive goals and clause formulas are tried in some arbitrary and fixed order, where backtracking allows the system to try the next choice when the current one results in failure.

At this point in time, the ideal order for trying clause formulas is not clear. The standard solution (used by λProlog and Lolli) is to try the dynamically assumed clauses in the reverse order of their assumption, and then try the static program clauses from first to last. However, this "temporal" ordering is not always optimal. For applications using the ordered context as a queue (*e.g.*, Section 10.5), it is preferable to try the ordered formulas from right to left, even though formulas are added to the left side of the ordered context. Therefore, we have chosen, for simplicity, to try dynamic clauses from right to left starting with the ordered context, then moving to the linear context, then to the unrestricted context, and finally trying the static program clauses from first to last[2]. Thus given the goal

$$D_1 \multimap D_2 \to D_3 \twoheadrightarrow D_4 \rightarrowtail P$$

After assuming all the new clause formulas, the order in which the clauses will be focussed on is: D_3, D_4, D_1, D_2.

After the initial release of Lolli, more analysis of linear logic proof search yielded

[2] This operational semantics is actually different from that of Lolli, however there is no reason (other than simplicity) for not using a more complicated search which matches Lolli for Lolli programs and still tries the ordered context from right to left.

techniques for improving the efficiency of Lolli proof search. As it turns out, these techniques can be extended to Olli, but we delay our treatment of them until chapter 11.

We assume a typed, higher-order term language[3] and implicit universal quantification over free variables (tokens beginning with an uppercase letter). We also write $B \twoheadleftarrow A$ for $A \twoheadrightarrow B$ and $B \leftarrowtail A$ for $A \rightarrowtail B$ in the manner of Prolog where backwards arrows are left-associative, and bind looser than forwards arrows and &. We use *italics* for meta-variables which stand for ground terms and `typewriter` for program code, including logic variables.

The following sections contain example logic programs which may be written in Olli. We note that all three of our system's contexts are important for logic programming. The program clauses will typically reside in the unrestricted context since their use should be unrestricted. The linear context is used to hold information which can be accessed in any order– the final examples in this chapter show how the linear context can be put to use. The ordered context can act as a logical data structure and hold information which must be accessed in a constrained fashion.

Section 10.2 contains a simple example which we will use to illustrate Olli's operational behavior in detail. In later sections, we assume familiarity with the operational semantics of Olli and concentrate mostly on the declarative interpretations of the example programs.

10.2 Simple Examples

We begin by considering various simple programs concerned with lists. `::` is the infix list constructor and `nil` is the empty list.

The following program, which does not make use of the ordered fragment of the

[3] We will assume for the sake of simplicity that the terms are dynamically within the L_λ fragment [35] which makes unification deterministic.

logic (and is therefore a Lolli program), can be used to permute a list.

```
perm (X::L) K
   ⟜ (elem X ⊸ perm L K).

perm nil (X::K)
   ⟜ elem X
   ⟜ perm nil K.

perm nil nil.
```

The program works on a query `perm` l `K` by first assuming `elem` x for every element x of l. This is achieved by the first clause. Then the assumptions are consumed one by one through uses of the second clause and added to the output list. The tail of the output list is instantiated to `nil` when there are no further linear assumptions, and the last clause can therefore succeed. Because the linear context is unordered, every possible order of linear resource consumption constitutes a valid proof where the result variable, `K`, becomes instantiated to a different permutation of the input list l. Thus, by interactively asking for solutions (or by explicitly failing), this program can be made to enumerate all possible permutations of a list.

If we replace the linear implications by right ordered implications, only one order remains possible: the one that reverses the list.

```
rev (X::L) K
   ↞ (elem X ↠ rev L K).

rev nil (X::K)
   ↞ elem X
   ↞ rev nil K.

rev nil nil.
```

After assuming `elem` x for every element x of l, we will be in the following situation, where we have elided the unrestricted context (which only contains the program clauses), the always empty linear context, and all $\top$ flags

$$\texttt{elem } x_1 \ldots \texttt{elem } x_n \backslash \Omega \;\longrightarrow\; \texttt{rev nil K}$$

where Ω has yet to be computed. At this point we must use the second `rev` clause; choosing the final clause would lead to failure, even though the head matches, since the ordered hypotheses have not been consumed. Upon choosing the second clause we are in the following situation (remember the **choice**$_\Gamma$ rule residuates the chosen formula)

$$\texttt{elem } x_1 \ldots \texttt{elem } x_n \backslash \Omega \;\longrightarrow\; (\texttt{rev nil K} \doteq \texttt{rev nil (X::K')}) \bullet (\texttt{elem X} \circ (\texttt{rev nil K'} \circ \mathbf{1}))$$

The first part of the goal succeeds, setting `K` = `(X::K')`, without consuming any resources and leaving us in the following situation

$$\texttt{elem } x_1 \ldots \texttt{elem } x_n \backslash \Omega \;\longrightarrow\; \texttt{elem X} \circ (\texttt{rev nil K'} \circ \mathbf{1})$$

We now proceed to solve the first subgoal, note that output context Ω' for the first subgoal is different than the output context Ω of the entire goal, and leave the second subgoal pending

$$\texttt{elem } x_1 \ldots \texttt{elem } x_n \backslash \Omega' \;\longrightarrow\; \texttt{elem X}$$

At this point we must choose one of the `elem` x_i to match the `elem X` in the goal. Following our operational semantics, we start with the rightmost ordered hypothesis, `elem` x_n. This choice will succeed, setting $\texttt{X} = x_n$, and cause $\Omega' = \texttt{elem } x_1 \ldots \texttt{elem } x_{n-1}\Box$. Thus we know that $\texttt{K} = x_n\texttt{::K'}$.

We next restore the pending goal and arrive, via the $\circ_R$ rule, at the following situation

$$\texttt{elem } x_1 \ldots \texttt{elem } x_{n-1} \backslash \Omega'' \;\longrightarrow\; \texttt{rev nil K'} \circ \mathbf{1}$$

where we know that $\Omega = \Omega''\Box$. The execution will proceed in the same manner, at each step consuming the rightmost ordered hypothesis, until the input ordered context is empty at which point, using the third program clause, the computation will end with $\texttt{K} = x_n\texttt{::}\ldots\texttt{::}x_1\texttt{::nil}$. Note that the goal $\mathbf{1}$ is always immediately successful and thus we may ignore all such subgoals which build up over the course of an execution.

When introducing this example, we mentioned that only one solution existed for a given input list. Not choosing the rightmost ordered hypothesis, when solving goals of the form `elem X`, always results in some ordered hypotheses (those to the right of the chosen hypothesis) not being consumed and thus a failed proof attempt. We delay a detailed explanation and analysis of this behavior until Chapter 11 where we improve the failure mechanism for our proof search.

Note that changing the outer $\twoheadrightarrow$ in the second clause will only result in changing the $\circ$ in the compiled version to a $\bullet$. Furthermore, this change does not affect provability since $\mathbf{1}$ is the unit for both conjunctions. In general the outermost ordered arrow of an unrestricted (or linear) program clause may be either $\twoheadrightarrow$ or $\rightarrowtail$ without affecting the semantics of a program.

We could also "uncurry" this example in the following way

```
rev (X::L) K
   ↞ (elem X ↠ rev L K).

rev nil (X::K)
   ↞ elem X ∘ rev nil K.

rev nil nil.
```

and maintain the exact same operational behavior. This is evident from the residuation of the second clause (supposing a goal of the form `rev nil K'`)

$$(\texttt{rev nil K'} \doteq \texttt{rev nil (X::K)}) \bullet ((\texttt{elem X} \circ \texttt{rev nil K'}) \circ \mathbf{1})$$

which is equivalent to the residuation of the "curried" form shown earlier.

We could also change the second clause to

```
rev nil (X::K)
   ↞ rev nil K • elem X.
```

and maintain the same declarative meaning as the previous versions, *i.e.*, this version will admit the same solutions as the other versions. However, the operational behaviour will now be quite different. In particular, the "recursive" subgoal of this clause, `rev nil K`, will be solved first. This will create a stack of pending goals of he form `elem X` each of which will need to consume the leftmost ordered hypothesis for the entire proof to succeed. Thus, due to the need to try ordered hypotheses from right to left, each of the pending goals, `elem X` will have to try all of the ordered hypotheses, and fail for each choice but the last, before succeeding.

We point out one further permutation. If we replace the nested $\twoheadrightarrow$ in the first program clause with $\rightarrowtail$, the program will add elements to the left of the ordered context. Then, since the rest of the program is unchanged and requires consuming ordered hypotheses from right to left, this modified program represents the identity relation.

```
id (X::L) K
   ↞ (elem X ↣ id L K).

id nil (X::K)
   ↞ elem X
   ↞ id nil K.

id nil nil.
```

10.3 Translating to deBruijn Notation

Our first serious example is a translation between lambda terms and deBruijn style terms. More details on this presentation of deBruijn terms and their connection to regular terms can be found in [47]. We use the following grammars for regular terms and deBruijn terms:

$$\begin{array}{lrcl} \text{Lambda Terms} & e & ::= & x \mid e_1\, e_2 \mid \lambda x.\ e \\ \text{deBruijn Terms} & e' & ::= & 1 \mid e' \uparrow \mid e'_1\, e'_2 \mid \Lambda e' \end{array}$$

Given a lambda term, we can construct a deBruijn term by recursively descending through the given term's structure and maintaining a stack of lambda-bound variables. Upon reaching a variable, its depth in the stack corresponds to the index in a deBruijn term.

We can translate deBruijn terms to regular lambda terms with the following judgement:

$$K \vdash e \leftrightarrow e'$$

where K is a list of regular variables, *i.e.*, $K = x_1 \ldots x_n$.

Here are the derivation rules for the translation:

$$\frac{K \vdash e_1 \leftrightarrow e'_1 \qquad K \vdash e_2 \leftrightarrow e'_2}{K \vdash e_1\, e_2 \leftrightarrow e'_1\, e'_2}\,\mathbf{tr\_app} \qquad\qquad \frac{K\, x \vdash e \leftrightarrow e'}{K \vdash \lambda x.\ e \leftrightarrow \Lambda\, e'}\,\mathbf{tr\_lam}^x$$

$$\frac{}{K\, x \vdash x \leftrightarrow 1}\,\mathbf{tr\_1} \qquad\qquad \frac{K \vdash e \leftrightarrow e'}{K\, x \vdash e \leftrightarrow e' \uparrow}\,\mathbf{tr\_}\!\uparrow$$

where x does not occur free in the conclusion of the $\mathbf{tr\_lam}^x$ rule. Note that the context, K, is a stack; and that the translation is non-deterministic, *e.g.*, $\lambda x.\ \lambda y.\ y\,(x\,x)$ can be translated to both $\Lambda\Lambda 1\,((1\uparrow)\,(1\uparrow))$ and $\Lambda\Lambda 1\,((1\,1)\uparrow)$.

We can directly transcribe the previous derivation rules to an Olli program which uses the ordered context as a stack.

We assume the following term constants (where `exp` and `exp'` are arbitrary base types for terms):

```
                                     lam'  :  exp' -> exp'.
lam  :  (exp -> exp) -> exp.         app'  :  exp' -> exp' -> exp'.
app  :  exp -> exp -> exp.          shift  :  exp' -> exp'.
                                      one  :  exp'.
```

`lam`, and `app` represent regular lambda terms (variables will be implicitly represented by meta-variables); while `lam'`, `app'`, `shift`, and `one` represent deBruijn terms.

We use the following predicates:

```
tr :  exp -> exp' -> o.    var :  exp -> o.
```

where `tr` is the translation program and `var` is a helper predicate to store named variables in the ordered context. We follow the λ-Prolog convention and use `o` to stand for the type of propositions. `tr` e `E'` expects a lambda term, e, and computes an equivalent deBruijn style term, `E'`.

We can now transcribe each rule as a program clause. We will use the following general scheme to encode the translation judgements as ordered linear logic sequents:

$$x_1 \ldots x_n \vdash e \leftrightarrow e' \qquad \rightsquigarrow \qquad \Gamma; \cdot; \texttt{var}\ x_1 \ldots \texttt{var}\ x_n \Longrightarrow \texttt{tr}\ \ulcorner e \urcorner\ \ulcorner e' \urcorner$$

where the unrestricted context Γ contains the translation program (which we develop below). $\ulcorner e \urcorner$ is the obvious representation of regular term e using the constructors for `exp` and $\ulcorner e' \urcorner$ is the representation of deBruijn term e' using the constructors for `exp'`. We will delay a formal presentation of the correctness of this encoding, and the ensuing representation of the derivation rules, until the end of Chapter 15 where we reformulate this example in an ordered logical framework.

We start with the **tr_app** rule. To translate `app E1 E2`, we simply translate both subterms using the current stack. Thus, eliding the unrestriced and linear contexts, we need to reduce proving

$$\texttt{var}\ x_1 \ldots \texttt{var}\ x_n \Longrightarrow \texttt{tr (app E1 E2) (app' E1' E2')}$$

to proving

$$\texttt{var}\ x_1 \ldots \texttt{var}\ x_n \Longrightarrow \texttt{tr E1 E1'}$$

and

$$\texttt{var}\ x_1 \ldots \texttt{var}\ x_n \Longrightarrow \texttt{tr E2 E2'}$$

where we have omitted the unrestricted and linear contexts. The x_i are term-level variables already encountered during the translation. This is achieved with the following clause:

$$\begin{array}{l}\texttt{tr (app E1 E2) (app' E1' E2')}\\ \quad \twoheadleftarrow \texttt{tr E1 E1'} \mathbin{\&} \texttt{tr E2 E2'}.\end{array}$$

Notice the & gives a copy of the stack (the ordered context) to each subterm translation. Here is the compiled version of the clause which will actually be used:

$$(\texttt{tr (app}\ e_1\ e_2\texttt{) E'} \doteq \texttt{tr (app E1 E2) (app' E1' E2')})\bullet((\texttt{tr E1 E1'}\&\texttt{tr E2 E2'})\circ\mathbf{1})$$

where tr (app e_1 e_2) E' is the goal at the time the clause is chosen.

We next consider the **tr_lam** rule. To translate a lam E, we add a variable to the stack and translate the body of the lambda. Thus we must reduce solving

$$\texttt{var}\ x_1 \ldots \texttt{var}\ x_n \Longrightarrow \texttt{tr (lam E) (lam' E')}$$

to

$$\texttt{var}\ x_1 \ldots (\texttt{var}\ x_n)\,(\texttt{var}\ x) \Longrightarrow \texttt{tr (E}\ x\texttt{) E'}$$

This is accomplished by the following clause:

$$\begin{array}{l}\texttt{tr (lam E) (lam' E')}\\ \quad \twoheadleftarrow (\forall \texttt{x. var x} \twoheadrightarrow \texttt{tr (E x) E'}).\end{array}$$

The inner ↠ forces the variable to be added to the top of the stack (the right side of the ordered context[4]). Here is the compiled version of the clause:

$$(\texttt{tr (lam}\ e\texttt{) E'} \doteq \texttt{tr (lam E) (lam' E')}) \bullet (\forall \texttt{x. var x} \twoheadrightarrow \texttt{tr (E x) E'}) \circ \mathbf{1})$$

To translate a variable, which e must be if it is not an application or lambda, we search through the stack, keeping track of how far we've gone, until we find the variable. If the target variable is at the top of the stack (the rightmost var in the context), which corresponds to the **tr_1** rule, we have the following situation:

$$\texttt{var}\ x_1 \ldots (\texttt{var}\ x_n)\,(\texttt{var}\ x) \Longrightarrow \texttt{tr}\ x\ \texttt{one}$$

[4] This is an arbitrary choice. We could rewrite the whole program to use the left side of the context as the top. However this would require changing the program clauses that read variables off the stack.

where we are done and can throw away the rest of the stack. This is accomplished by:

$$\begin{array}{l}\texttt{tr E one}\\ \quad \twoheadleftarrow \texttt{var E}\\ \quad \twoheadleftarrow \top.\end{array}$$

For this clause to be successfully used, the `var E` must come from the top of the stack (the right of the ordered context) since the $\top$ can only consume data to the left of `var E`. This situation is identical to that of the `rev` program in Section 10.2. For reference we show the compiled version of this clause:

$$(\texttt{tr}\ x\ \texttt{one} \doteq \texttt{tr E one}) \bullet (\texttt{var E} \circ (\top \circ \mathbf{1}))$$

We could alternatively write the previous program clause as:

$$\begin{array}{l}\texttt{tr E one}\\ \quad \leftarrowtail \top\\ \quad \leftarrowtail \texttt{var E}.\end{array}$$

Using this alternate formulation does not change the declarative meaning of the program; furthermore, it does not really change the operational semantics in any essential way. Consider the compiled version of the clause:

$$(\texttt{tr}\ x\ \texttt{one} \doteq \texttt{tr E one}) \bullet (\top \bullet (\texttt{var E} \bullet \mathbf{1}))$$

Upon using this clause, after the unifying E and x, we arrive at the following situation:

$$\texttt{var}\ x_1 \ldots \texttt{var}\ x_n \backslash \Omega \underset{\tau\ \ v}{\longrightarrow} \top \bullet (\texttt{var}\ x \bullet \mathbf{1})$$

We first solve the subgoal $\top$, which always succeeds. Operationally $\top$ simply passes its input context to output, and sets the $\top$-flags. We will then be in the following situation:

$$\texttt{var}\ x_1 \ldots \texttt{var}\ x_n \backslash \Omega \underset{v'\ \ \tau'}{\longrightarrow} \texttt{var}\ x \bullet \mathbf{1}$$

where v' and τ' have yet to be computed. At this point, the proof search proceeds in the same manner as before, with the rightmost ordered hypothesis always being successfully chosen. Furthermore, it is still the case that only the rightmost resource can be successfully used. After the preceding sequent is solved, we know $\tau = 1\tau'$ as mandated by the $\bullet_R$ rule. Thus the $\top$ can only consume formulas occurring to the left of all $\Box$ in Ω.

Finally, if the target variable is not at the top of the stack, we must reduce

$$\texttt{var}\ x_1, \ldots, \texttt{var}\ x_n,\ \texttt{var}\ x \Longrightarrow \texttt{tr}\ y\ \texttt{(shift E')}$$

to

$$\texttt{var}\ x_1, \ldots, \texttt{var}\ x_n \Longrightarrow \texttt{tr}\ y\ \texttt{E'}$$

which corresponds to the **tr_**↑ rule. We accomplish this by the following clause:

```
tr E (shift E')
   ↞ var F
   ↞ tr E E'.
```

Finally, we note that the deBruijn translation program can also be used to translate the other direction, from deBruijn terms to regular terms, by using a query of the form `tr E` e' where e' is a deBruijn term.

10.4 Mini-ML Abstract Machine

Our next example shows how a continuation-based abstract machine for evaluating Mini-ML can be directly encoded in Olli. The basic idea is to use the ordered context as a stack of continuations to be evaluated. We assume a standard version of Mini-ML constructed using higher-order abstract syntax [47]. Values are distinguished from terms by an asterisk; so **z** is a term while $\mathbf{z}^*$ is a value.

$$\begin{array}{rrcl}
\text{Expressions} & e & ::= & \mathbf{z} \mid \mathbf{s}\, e \mid \mathbf{lam}\, x.\ e \mid e_1\, e_2 \mid \\
 & & & (\mathbf{case}\, e_1\, \mathbf{of}\ \mathbf{z} \Rightarrow e_2 \mid \mathbf{s}\, x \Rightarrow e_3) \mid v \\
\text{Values} & v & ::= & \mathbf{z}^* \mid \mathbf{s}^*\, v \mid \mathbf{lam}^* x.\ e \mid x
\end{array}$$

We define the continuation machine as follows:

$$\begin{array}{rrcl}
\text{Instructions} & i & ::= & e \mid \mathbf{return}\, v \mid \\
 & & & (\mathbf{case}_1\, v_1\, \mathbf{of}\ \mathbf{z} \Rightarrow e_2 \mid \mathbf{s}\, x \Rightarrow e_3) \mid \\
 & & & \mathbf{app}_1 v_1 e_2 \mid \mathbf{app}_2 v_1 v_2 \\
\text{Continuations} & K & ::= & \mathbf{init} \mid K; \lambda x.\ i \\
\text{Machine States} & s & ::= & K \diamond i \mid \mathbf{answer}\, v
\end{array}$$

We use the following transition rules for machine states:

$$\begin{array}{rcl}
\textbf{st\_init} & :: & \textbf{init} \diamond \textbf{return}\, v \hookrightarrow \textbf{answer}\, v \\[1ex]
\textbf{st\_return} & :: & K; \lambda x.\, i \diamond \textbf{return}\, v \hookrightarrow K \diamond i[v/x] \\[1ex]
\textbf{st\_vl} & :: & K \diamond v \hookrightarrow K \diamond \textbf{return}\, v \\[1ex]
\textbf{st\_z} & :: & K \diamond \textbf{z} \hookrightarrow K \diamond \textbf{return}\, \textbf{z}^* \\[1ex]
\textbf{st\_s} & :: & K \diamond \textbf{s}\, e \hookrightarrow K; \lambda x.\, \textbf{return}(\textbf{s}^*\, x) \diamond e \\[1ex]
\textbf{st\_case} & :: & K \diamond \textbf{case}\, e_1\, \textbf{of}\, \textbf{z} \Rightarrow e_2 |\, \textbf{s}\, x \Rightarrow e_3 \hookrightarrow \\
& & \quad K; \lambda x_1.\, \textbf{case}_1\, x_1\, \textbf{of}\, \textbf{z} \Rightarrow e_2 |\, \textbf{s}\, x \Rightarrow e_3 \diamond e_1 \\[1ex]
\textbf{st\_case1\_z} & :: & K \diamond \textbf{case1}\, \textbf{z}^*\, \textbf{of}\, \textbf{z} \Rightarrow e_2 |\, \textbf{s}\, x \Rightarrow e_3 \hookrightarrow K \diamond e_2 \\[1ex]
\textbf{st\_case1\_s} & :: & K \diamond \textbf{case1}\, (\textbf{s}^*\, v)\, \textbf{of}\, \textbf{z} \Rightarrow e_2 |\, \textbf{s}\, x \Rightarrow e_3 \hookrightarrow K \diamond e_3[v/x] \\[1ex]
\textbf{st\_lam} & :: & K \diamond \textbf{lam}\, x.\, e \hookrightarrow K \diamond \textbf{return}\, (\textbf{lam}^* x.\, e) \\[1ex]
\textbf{st\_app} & :: & K \diamond e_1\, e_2 \hookrightarrow K; \lambda x_1.\, \textbf{app}_1\, x_1\, e_2 \diamond e_1 \\[1ex]
\textbf{st\_app1} & :: & K \diamond \textbf{app}_1\, v_1\, e_2 \hookrightarrow K; \lambda x_2.\, \textbf{app}_2\, v_1\, x_2 \diamond e_2 \\[1ex]
\textbf{st\_app2} & :: & K \diamond \textbf{app}_2\, (\textbf{lam}^* x.\, e)\, v_2 \hookrightarrow K \diamond e[v_2/x]
\end{array}$$

We now show how the continuation machine can be written as an Olli program. Rather than building an explicit stack-like structure to represent the continuation K, we will simply store instructions in the ordered context. Thus we will use the following representation to encode the machine:

$$K \diamond i \qquad \leadsto \qquad \ulcorner K \urcorner \Longrightarrow \ulcorner i \urcorner$$

where $\ulcorner K \urcorner$ is the representation, described below, of the continuation (stack) K and similarly for $\ulcorner i \urcorner$.

We use the following signature to encode the abstract machine as an Olli program:

$$\begin{array}{lcl}
\texttt{z} & : & \texttt{exp.} \\
\texttt{s} & : & \texttt{exp} \rightarrow \texttt{exp.} \\
\texttt{case} & : & \texttt{exp} \rightarrow \texttt{exp} \rightarrow (\texttt{val} \rightarrow \texttt{exp}) \rightarrow \texttt{exp.} \\
\texttt{lam} & : & (\texttt{val} \rightarrow \texttt{exp}) \rightarrow \texttt{exp.} \\
\texttt{app} & : & \texttt{exp} \rightarrow \texttt{exp} \rightarrow \texttt{exp.} \\
\texttt{vl} & : & \texttt{exp} \rightarrow \texttt{val.} \\
\texttt{z}^* & : & \texttt{val.} \\
\texttt{s}^* & : & \texttt{val} \rightarrow \texttt{val.} \\
\texttt{lam}^* & : & (\texttt{val} \rightarrow \texttt{exp}) \rightarrow \texttt{val.} \\
\texttt{eval} & : & \texttt{exp} \rightarrow \texttt{val} \rightarrow \texttt{o.} \\
\texttt{ev} & : & \texttt{exp} \rightarrow \texttt{o.} \\
\texttt{return} & : & \texttt{val} \rightarrow \texttt{o.} \\
\texttt{case1} & : & \texttt{val} \rightarrow \texttt{exp} \rightarrow (\texttt{val} \rightarrow \texttt{exp}) \rightarrow \texttt{o.} \\
\texttt{appm1} & : & \texttt{val} \rightarrow \texttt{exp} \rightarrow \texttt{o.} \\
\texttt{appm2} & : & \texttt{val} \rightarrow \texttt{val} \rightarrow \texttt{o.}
\end{array}$$

Given the goal: `return V` $\twoheadrightarrow$ `ev` e, our program will evaluate the expression e and instantiate `V` with the resulting value. The intended reading of this query is: evaluate e with the identity continuation (the continuation which just returns its value). A goal of `ev` e is intended to mean: evaluate e. A goal of `return V` is intended to mean: pass `V` to the top continuation on the stack (i.e. the rightmost element in the ordered context).

We could use term-level lambdas to represent abstractions over instructions, $\lambda x.\ i$, and explicitly encode the substitutions in the transition rules (in **st_return**, **st_case1_s**, and **st_app2**) as applications. However, we will instead make use of the left implication, $\rightarrowtail$, and let unification implicitly achieve the required substitutions. To do this, we will represent $\lambda x.\ i$ as $\forall$`V.return V` $\leftarrowtail \ulcorner i \urcorner$ where $\ulcorner i \urcorner$ is the representation of the instruction i. Thus the continuation (stack) built by an evaluation of `(s (s z))`

$$\textbf{init};\lambda x.\ \textbf{return}\,(\textsf{s}^*\ x);\lambda x.\ \textbf{return}\,(\textsf{s}^*\ x)$$

is represented as the ordered context

$$\texttt{(return V)}\quad (\forall \texttt{V. return V} \leftarrowtail \texttt{ return (s}^*\texttt{ V))}\quad (\forall \texttt{V. return V} \leftarrowtail \texttt{ return (s}^*\texttt{ V))}$$

We will then use subgoals of the form `return` v to explicitly pass a value to the top continuation on the stack. With this encoding, we will not need to explcitly represent the **st_init** and **st_return** transition rules. We have the following representations:

$$\mathbf{init} \diamond \mathbf{return}\, v \qquad \leadsto \qquad \texttt{return V} \Longrightarrow \texttt{return } v$$

where the logic variable V is the final answer;

$$K; \lambda x.\, i \diamond \mathbf{return}\, v \qquad \leadsto \qquad \ulcorner K \urcorner (\forall \texttt{V}.\, \texttt{return V} \leftarrowtail \ulcorner i \urcorner) \Longrightarrow \texttt{return } v$$

where the ordering constraints force the proof of `return` v to focus on the rightmost ordered formula.

We now show the clauses of the program. We begin with a wrapper to put queries into the correct form:

```
eval E V
  ↞ (return V ↠ ev E).
```

The rest of the program clauses directly mirror the machine transition rules:

```
ev (vl V)
  ↞ return V.

ev z
  ↞ return z*.
ev (s E)
  ↞ ((∀V. return V ↢ return (s* V)) ↠ ev E).
ev (case E1 E2 E3)
  ↞ ((∀V. return V ↢ case1 V E2 E3) ↠ ev E1).
case1 z* E2 E3
  ↞ ev E2.
case1 (s* V) E2 E3
  ↞ ev (E3 V).


ev (lam E)
  ↞ return (lam* E).
ev (app E1 E2)
  ↞ ((∀V1. return V1 ↢ app1 V1 E2) ↠ ev E1).
app1 V1 E2
  ↞ (∀V2. return V2 ↢ app2 V1 V2) ↠ ev E2).
app2 (lam* E1') V2
  ↞ ev (E1' V2).
```

The intended reading of the `ev (s E)` clause (the second program clause) is this: to evaluate `(s E)` evaluate `E` under the continuation which takes its value, `V`, and passes the value $\texttt{s}^*\ \texttt{V}$ to the next continuation on the stack. Note the use of $\rightarrowtail$ nested inside $\twoheadrightarrow$. This forces the continuations put into the ordered context to be evaluated in stack fashion. When the goal is `return V` the only choice of formula to focus on will be the rightmost formula in the ordered context.

To better illustrate this point, consider the evaluation of `(s (s z))`. The initial goal will be $\texttt{return V} \twoheadrightarrow \texttt{ev (s (s z))}$ after which `return V` will be immediately added to the previously empty ordered context. Next the `ev (s E)` clause will be chosen to focus on and will result in $\forall \texttt{V}.\,\texttt{return V} \leftarrowtail \texttt{return (s}^*\ \texttt{V)}$ being added to the right end (because of $\twoheadrightarrow$) of the ordered context. The new goal will be `ev (s z)` and the previous step will be repeated. At this point, the goal will be `ev z` which will cause the appropriate progam clause (the second in the preceding program listing) to be focused on and give rise to a new goal of $\texttt{return z}^*$.

The ordered context now consists of:

$$(\texttt{return V})\quad (\forall \texttt{V}.\,\texttt{return V} \leftarrowtail\ \texttt{return (s}^*\ \texttt{V)})\quad (\forall \texttt{V}.\,\texttt{return V} \leftarrowtail\ \texttt{return (s}^*\ \texttt{V)}).$$

Since there is no program clause whose head matches the goal, one of the clauses in the ordered context must be focused on. Although all the ordered clauses match the goal, only the rightmost one can be successfully chosen. The leftmost clause obviously cannot work since it is atomic and the other clauses are also in the ordered context. The middle clause also does not work because the $\rightarrowtail$ requires that the body of the clause be solved with resources to the left of the clause which would prevent the rightmost clause from being consumed.

10.5 Mergesort

Our next example, a merge sort, shows how the ordered context can be used as a queue. The merge sort algorithm takes an input list, breaks it up into singleton lists and merges pairs of adjacent lists into larger sorted lists. This process repeats until one sorted list is left. The algorithm can be implemented with a queue. After the initial setup– enqueing the singleton lists– we can repeatedly dequeue two lists and enqueue their merge until only one list is in the queue.

We use the following predicates:

$$\begin{array}{lcl}\texttt{mergeSort} & : & \texttt{list int} \to \texttt{list int} \to \texttt{o}.\\ \texttt{msort} & : & \texttt{list int} \to \texttt{o}.\\ \texttt{srt} & : & \texttt{list int} \to \texttt{o}.\end{array}$$

where `mergeSort` is the main program, whose first argument is the input list and whose second argument is the output; `msort` is a helper predicate which does the actual work; and `srt` is a wrapper which allows lists to be stored in the ordered context (the queue).

The computation proceeds in two phases. Assuming an input list $x_1 :: \cdots :: x_n :: \texttt{nil}$ we want to reduce solving

$$\cdot \Longrightarrow \texttt{mergeSort}\,(x_1 :: \cdots :: x_n :: \texttt{nil})\,\texttt{L}$$

to solving

$$\texttt{srt}(x_1 :: \texttt{nil}) \ldots \texttt{srt}(x_n :: \texttt{nil}) \Longrightarrow \texttt{msort L}$$

We achieve this with the two clauses

$$\begin{array}{l}\texttt{mergeSort (H::T) L}\\ \quad \twoheadleftarrow (\texttt{srt (H::nil)} \twoheadrightarrow \texttt{mergeSort T L}).\\[1ex] \texttt{mergeSort nil L} \twoheadleftarrow \texttt{msort L}.\end{array}$$

In the first clause, the embedded $\twoheadrightarrow$ causes the new `srt` hypothesis to be added to the right of all the other ordered hypotheses (the top of our queue).

In the second phase we assume a general situation of the form

$$(\texttt{srt}\,l_n)\ldots(\texttt{srt}\,l_2)(\texttt{srt}\,l_1) \Longrightarrow \texttt{msort L}$$

where the l_i are already sorted and `L` is still to be computed. Starting from the right, we merge l_1 and l_2 and add the result to the *left* end of the ordered context, in effect using it as a work queue. The resulting situation will be

$$(\texttt{srt}\,l_{12})(\texttt{srt}\,l_n)\ldots(\texttt{srt}\,l_4)(\texttt{srt}\,l_3) \Longrightarrow \texttt{msort L}$$

which is then treated the same way, merging l_3 and l_4. We finish when there is only one element $\texttt{srt}\,k$ in the ordered context and unify `L` with k. This is expressed by the following two clauses.

```
msort L
   ↞ srt L1
   ↞ srt L2
   ← merge L1 L2 L12
   ↞ (srt L12 ↣ msort L).

msort L ↞ srt L.
```

In the first clause, similar to the previous example, the three $\twoheadleftarrow$ require that the two `srt` hypotheses be taken from the right of the ordered clauses used to solve the rest of the body. The unrestricted implication, $\leftarrow$, is used for the call to `merge` since the merge operation does not make use of the ordered (or linear) context. The embedded $\rightarrowtail$ causes the new `srt` clause to be inserted at the left end of the ordered consequences available to the recursive computation of `msort`. Since all ordered assumptions must be used, the final clause above can succeed only if the complete list has indeed been sorted, that is, there is only one ordered hypothesis `srt` l.

The standard Prolog-style `merge` predicate which, given two sorted lists l_1 and l_2, returns a sorted merge l_{12} is as follows

```
merge (H1::T1) (H2::T2) (H2::T3)
   ← H1 > H2
   ← merge (H1::T1) T2 T3.

merge (H1::T1) (H2::T2) (H1::T3)
   ← H1 =< H2
   ← merge T1 (H2::T2) T3.

merge L nil L.

merge nil L L.
```

If we change the first `msort` clause to assume `L12` on the right, by writing (`srt L12` $\twoheadrightarrow$ `msort L`) instead of (`srt L12` $\rightarrowtail$ `msort L`), then we obtain an insertion sort because after one step we arrive at

$$(\mathtt{srt}\, l_n) \ldots (\mathtt{srt}\, l_3)(\mathtt{srt}\, l_{12})$$

which will next merge l_3 into l_{12}, etc.

10.6 Breadth-First Tree Numbering

Our next example also employs the ordered context as a queue. We show a program for computing the breadth-first numbering of a given tree. Given a tree as input, this program will compute a tree with the same structure whose nodes are labeled with integers corresponding to their order in a breadth-first traversal. This example directly encodes the algorithm proposed by Okasaki in [41].

The program essentially executes a breadth-first traversal of the given tree by using the ordered context as a work queue. The right end of the context will be the top of the queue; thus we will always use $\rightarrowtail$ to enqueue data. The root of the tree at the top of the work queue is always the next node to visit in the traversal. Thus, upon dequeing a tree, we enqueue its children and continue on. We build up the answer tree while "traversing" the input tree. This aspect of the computation requires the use of a double queue in the (eager) functional setting. However, we may entirely sidestep this issue with Olli by using logic variables to implicitly reconstruct our answer tree.

We assume the following tree constructors

$$\texttt{node} : \alpha \to \texttt{tree}\ \alpha \to \texttt{tree}\ \alpha \to \texttt{tree}\ \alpha. \qquad \texttt{empty} : \texttt{tree}\ \alpha.$$

for a parameterized type `tree`.

We make use of the following predicates in our program:

$$\begin{array}{lcl} \texttt{nd} & : & (\texttt{tree}\,\alpha) \to (\texttt{tree int}) \to \texttt{o}. \\ \texttt{bf\_num} & : & (\texttt{tree}\,\alpha) \to (\texttt{tree int}) \to \texttt{o}. \\ \texttt{bf} & : & \texttt{int} \to \texttt{o}. \end{array}$$

where `int` is the type of Peano numbers with the usual constructors `z` and `s`. `nd` stores an input (sub)tree and the equivalent breadth-first numbered (sub)tree. `bf_num` is our main predicate and `bf` is a helper predicate which simply cycles through the work queue implicitly represented by the ordered context.

We begin by enqueuing the whole tree and setting the counter to 0.

$$\begin{array}{l} \texttt{bf\_num T T'} \\ \quad \twoheadleftarrow (\texttt{nd T T'} \rightarrowtail \texttt{bf z}). \end{array}$$

We will now be in the following (generalized) situation:

$$(\texttt{nd}\ t_1\ \texttt{T1}) \ldots (\texttt{nd}\ t_m\ \texttt{Tn}) \Longrightarrow \texttt{bf}\ n$$

If the queue is empty, we are done.

```
bf N.
```

No further computation is required since unification will have already reconstructed the complete answer tree. Note that this clause will only succeed if the ordered context is empty.

If there is an empty tree at the top of the queue

$$(\texttt{nd}\ t_1\ \texttt{T1})\ldots(\texttt{nd empty T'}) \Longrightarrow \texttt{bf}\ n$$

then we just ignore it since there are no leaves to include in the breadth-first numbering.

```
bf N
   ↞ nd empty empty
   ↞ bf N.
```

Note that this clause, like similar clauses in the previous examples, can only succeed when the first subgoal matches the rightmost ordered clause.

If there is a non-empty tree at the top of the queue

$$(\texttt{nd}\ t_1\ \texttt{T1})\ldots(\texttt{nd (node}\ v\ t_l\ t_r\texttt{) T'}) \Longrightarrow \texttt{bf}\ n$$

mark the answer subtree's root node with the current node number (accomplished by unification), enqueue the two children and continue on with the current node number incremented.

```
bf N
   ↞ nd (node _ L R) (node N L' R')
   ↞ (nd L L' ↣ nd R R' ↣ bf (s N)).
```

Note that the two children must be enqueued in the order written above to achieve the correct ordering

$$(\texttt{nd}\ t_r\ \texttt{R'})(\texttt{nd}\ t_l\ \texttt{L'})(\texttt{nd}\ t_1\ \texttt{T1})\ldots \Longrightarrow \texttt{bf (s}\ n\texttt{)}$$

10.7 Breadth-First Search Graph Numbering

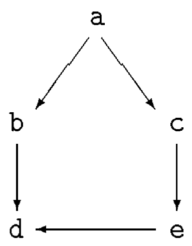
Our next example continues the theme of breadth-first search. However, we will now show a more involved program which computes the shortest distance from a given start node to every other node in a graph (assuming equal length edges). This example will make use of all three contexts.

This program will be given a graph[5] and a start node as input. It will return a list of distances from the start node to each other graph node. The computation proceeds by executing a breadth-first traversal of the graph during which each node is marked with the its distance from the start node. The shortest distance is guaranteed since the traversal is breadth-first.

This program will assume graphs are lists of nodes with their associated edge lists. We make the following type definition:

$$\texttt{graph}\ \alpha\ \equiv\ (\texttt{list}\ (\alpha \times (\texttt{list}\ \alpha)))$$

Thus we have the following representation:



≡ [<a,[b,c]>, <b,[d]>, <c,[e]>, <d,[]>, <e,[d]>]

and the result of running our program on this graph with a given as the start node would be:

[<a,0>, <b,1>, <c,1>, <d,2>, <e,2>]

We will use the following predicates for our program (where opt is the standard parametrized option type):

$$\begin{aligned}
\texttt{bfs\_main} &: (\texttt{graph}\ \alpha) \rightarrow \alpha \rightarrow (\texttt{list}\ (\alpha \times (\texttt{opt}\,\texttt{int}))) \rightarrow \texttt{o}.\\
\texttt{bfs} &: (\texttt{graph}\ \alpha) \rightarrow \alpha \rightarrow \texttt{o}.\\
\texttt{bfs'} &: \alpha \rightarrow \texttt{int} \rightarrow \texttt{o}.\\
\texttt{bfs''} &: (\texttt{list}\ \alpha) \rightarrow \texttt{int} \rightarrow \texttt{o}.
\end{aligned}$$

$$\begin{aligned}
\texttt{finish} &: (\texttt{graph}\ \alpha) \rightarrow (\texttt{list}\,(\alpha \times (\texttt{opt}\,\texttt{int}))) \rightarrow \texttt{o}.\\
\texttt{nodes} &: (\texttt{graph}\ \alpha) \rightarrow (\texttt{list}\,(\alpha \times (\texttt{opt}\,\texttt{int}))) \rightarrow \texttt{o}.\\
\texttt{node} &: (\alpha \times (\texttt{list}\ \alpha)) \rightarrow \texttt{o}.\\
\texttt{used} &: (\alpha \times \texttt{int}) \rightarrow \texttt{o}.\\
\texttt{next} &: (\alpha \times \texttt{int}) \rightarrow \texttt{o}.
\end{aligned}$$

[5] We assume that each node is uniquely labelled in the graph.

where the arguments to bfs_main are the input graph, start node and result list respectively; bfs, bfs' and bfs'' are helper predicates; finish is a predicate to package up the results of the computation into the output list; and the final four predicates allow various terms to be stored in the various contexts.

The program begins by placing a copy of the entire input graph into the context for later use. We will need this at the end of the computation to gracefully deal with disconnected graphs.

```
bfs_main G S R
   ← (nodes G R → bfs G S).
```

After storing away a copy of the input graph, we begin by placing each node of the graph, with its edge list, into the linear context. When done placing the entire graph into the linear context, we call bfs' to begin the traversal at the specified start node, S, with current distance 0.

```
bfs (N::G) S
   ∘— (node N ⊸ bfs G S).
bfs nil S
   ∘— bfs' S z.
```

We are now ready to begin the breadth-first traversal, each step of which is broken into two stages. In the first stage, we will be in the following situation:

$$\Gamma(\texttt{nodes}\ g\ \texttt{R})\,;\,\Delta\,;\,(\texttt{next}\ \texttt{<}v_1,d_1\texttt{>})\ldots(\texttt{next}\ \texttt{<}v_n,d_n\texttt{>})\ \Longrightarrow\ \texttt{bfs'}\ v\ d$$

where Γ contains the program clauses, g is the original input graph, Δ contains either node <v_i, e_i> or used <v_i, d'_i> for each node label v_i in g, and $v = v_i$ for some i.

First we take the node we are going to start with, v, out of the linear context. We then mark v as used, also recording the current distance, and pass v's edge list to the second stage of the search process. If v was already encountered earlier in the search, we continue to the second stage without passing it's edge list.

```
bfs' V D
   ∘— node <V,E>
   ↞ (used <V,D> ⊸ bfs'' E D).
bfs' V D
   ∘— used <V,D'>
   ↞ (used <V,D'> ⊸ bfs'' nil D).
```

At the beginning of the second stage, we will be in the following situation:

$$\Gamma(\texttt{nodes}\ g\ \texttt{R})\,;\,\Delta\,;\,(\texttt{next <}v_1, d_1\texttt{>})\ldots(\texttt{next <}v_n, d_n\texttt{>}) \Longrightarrow \texttt{bfs''}\ e\ d$$

We now add new edges to the work queue and then pick a node off the top of the queue to continue the search from (return to the first stage with). If there are no nodes left in the queue, the computation is finished and we just need to package up the results. We accomplish this by passing our copy of the input graph and the final result list to `finish`.

```
bfs'' (V::Vs) D
   ↞ (next <V,D> ↣ bfs'' Vs D).
bfs'' nil D
   ↞ next <V,D'>
   ↞ bfs' V (s D').
bfs'' nil D
   ← nodes G R
   ⟜ finish G R.
```

Once the computation is finished, the ordered context will be empty and the linear context will contain each node's distance from start node (if the node was reachable).

$$\Gamma(\texttt{nodes}\ g\ \texttt{R})\,;\,\Delta\,;\,\cdot \Longrightarrow \texttt{finish}\ g\ \texttt{R}$$

The program finishes by placing all of the graph nodes and their distances into the output list. Unreachable nodes, left in the context from the initialization process, will be marked with `none`.

```
finish nil nil.
finish (<V,E>::G) (<V,some D>::R)
   ⟜ used <V,D>
   ⟜ finish G R.
finish (<V,E>::G) (<V,none>::R)
   ⟜ node <V,E>
   ⟜ finish G R.
```

10.8 Parsing

The following example is a fragment of a parsing program from [50]. This example shows how Olli can be used to directly parse grammatical constructions with unbounded dependencies such as relative clauses.

1 : $\mathtt{snt} \twoheadleftarrow \mathtt{vp} \twoheadleftarrow \mathtt{np}$.
2 : $\mathtt{vp} \twoheadleftarrow \mathtt{np} \twoheadleftarrow \mathtt{tv}$.
3 : $\mathtt{rel} \leftarrowtail \mathtt{whom} \twoheadleftarrow (\mathtt{np} \multimap \mathtt{snt})$.
4 : $\mathtt{np} \twoheadleftarrow \mathtt{jill}$.
5 : $\mathtt{tv} \twoheadleftarrow \mathtt{married}$.

We may intuitively read the formulas in the following manner: $\mathtt{snt} \twoheadleftarrow \mathtt{vp} \twoheadleftarrow \mathtt{np}$ states that a sentence is a verb phrase to the right of a noun phrase. We can use these formulas to parse a phrase by putting each word of the sentence into the ordered context and trying to derive the atomic formula corresponding to the phrase type. This method of parsing was used by Lambek in his paper introducing the Lambek calculus [32].

We may interpret clause 3 as: a relative clause is `whom` to the left of a sentence missing a noun phrase. As explained in [26] this is a standard interpretation of relative clauses. By putting a `np` into the linear context, the sentence after `whom` will only be successfully parsed if it is indeed missing a noun phrase.

We now show a trace of the above formulas parsing a relative clause, showing at each step the resource sequent (including the current goal), the pending goals, and the the rule applied. The unrestricted context containing the above program is left implicit.

Action	Active hypotheses and goal	Goals pending
	$\cdot\,;\, \mathtt{whom\ jill\ married} \longrightarrow \mathtt{rel}$	none
reduce by 3	$\cdot\,;\, \mathtt{whom\ jill\ married} \longrightarrow \mathtt{whom}$	$\mathtt{np} \multimap \mathtt{snt}$
solved, restore pending goal	$\cdot\,;\, \mathtt{jill\ married} \longrightarrow \mathtt{np} \multimap \mathtt{snt}$	none
assume	$\mathtt{np}\,;\, \mathtt{jill\ married} \longrightarrow \mathtt{snt}$	none
reduce by 1	$\mathtt{np}\,;\, \mathtt{jill\ married} \longrightarrow \mathtt{vp}$	$\mathtt{np}$
reduce by 2	$\mathtt{np}\,;\, \mathtt{jill\ married} \longrightarrow \mathtt{np}$	$\mathtt{tv}\,,\, \mathtt{np}$
solved, restore pending goal	$\cdot\,;\, \mathtt{jill\ married} \longrightarrow \mathtt{tv}$	$\mathtt{np}$
reduce by 5	$\cdot\,;\, \mathtt{jill\ married} \longrightarrow \mathtt{married}$	$\mathtt{np}$
solved, restore pending goal	$\cdot\,;\, \mathtt{jill} \longrightarrow \mathtt{np}$	none
reduce by 4	$\cdot\,;\, \mathtt{jill} \longrightarrow \mathtt{jill}$	none
solved		

Using the linear context in manner described above has some limitations which should be pointed out. The correct parsing of dependent clauses typically constrains where the relative pronoun may fill in for a missing noun phrase. Most relative clauses, rather than being sentences missing noun phrases are really sentences whose verb phrase is missing a noun phrase.

If we changed the relative clause to be **whom married jill** we would not have a grammatically correct relative noun. However the parser given above will be able to parse the modified phrase since the location of the missing noun phrase is not constrained. There are a variety of simple ways to fix this problem for the small parser given above. For instance, we could define a new type of sentence in which the verb phrase is missing a noun phrase. This basically amounts to using gap-locator rules as described in [42] [26].

The parsers given in [26], which logically handle some constraints on the placement of dependencies, are constructed quite differently from the Olli parser we have presented. Rather than placing the input to be parsed into the context, they pass it around as a list. They do however use the linear context to store fillers— empty noun phrase predicates which can be used when an actual noun phrase is missing in the sentence— in the same manner as the above parser does. We point out that all of the (pure) Lolli parsers are also valid Olli programs since (pure) Lolli is a subset of Olli.

In fact with this threaded style of parser, Olli is able to correctly handle at least one natural language phenomenom which could not be done in pure Lolli. When a relative clause occurs inside a relative clause, correct parses of the sentence should associate the inner relative pronoun with the first missing noun phrase in the clause and the second with the second. In other words, dependencies should not cross inside a nested relative clause. Consider the phrase: **the book that the man whom Jane likes GAP wrote GAP** where **GAP** denotes a missing noun phrase. The first **GAP** should correspond to **the man** and not to **the book**.

Since the Lolli parser (as well as the Olli parser given above) introduces fillers into the linear context, there is no way to force the parser to use one or the other of two suitable fillers. Thus the Lolli parser would parse the preceding sentence in two ways, only one of which would be correct. Hodas' solution was to rely on a non-logical control construct and the operational semantics of Lolli to prevent the bad parse. One can easily see (as Hodas remarked) that such a situation can be directly handled in Olli by putting the fillers into the ordered context.

Chapter 11

Eager Failure

While the $\top$-flags derivation system of chapter 9 provides a reasonably efficient proof search procedure for Olli, it does not take full advantage of the ordering constraints to fail as soon as possible. Since the Olli's operational semantics specify a depth first search, it is advantageous to fail as early as possible in order to minimize program execution time. A $\top$-flags derivation can only fail when an atom does not match the head of the focus formula in the **init** rule, or when an ordered, or linear, hypothesis is not consumed. This second condition is only tested after the output context of the $\twoheadrightarrow_R$, $\rightarrowtail_R$, and $\multimap_R$ rules are computed. However, by adding a little bit of information to the sequents, it is possible to *eagerly* detect such a failure much earlier during proof search.

In [10], Cervesato *et al.* introduced RM3, a derivation system for Lolli which better utilized the linearity constraints and failed earlier than previous systems. Lopez and Pimentel then refined this approach in [33], taking advantage of the operational semantics of derivation search, and produced an equivalent system, the frame system, better suited to implementation than RM3. In this chapter, we extend Lopez and Pimentel's approach to OLL and produce a derivation system which fails much sooner than the $\top$-flags system.

11.1 Failing Earlier in Linear Logic

In order to fail earlier, we keep track of which parts of the linear context must be consumed and which parts need not be consumed in a derivation. Then, at the end of a derivation branch (in the **init** rule), we require that the whole linear context need not be consumed. For this reason we will call our new derivation system strict– we do not allow hypotheses into output contexts if we know they will never be consumed. We determine a formula will never be consumed if there is no possibility of it being passed into a future derivation branch (assuming bottom-up, left-to-right derivation construction).

Consider the following example of a failed derivation (for a purely linear system):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P_1P_2\backslash P_1P_2 \longrightarrow P \gg P}\,\textbf{init} \qquad \dfrac{\dfrac{}{P_2\backslash P_2 \longrightarrow P_1 \gg P_1}\,\textbf{init}}{P_1P_2\backslash\Box P_2 \longrightarrow P_1}\,\textbf{choice}_\Delta}{P_1P_2\backslash\Box P_2 \longrightarrow P_1 \multimap P \gg P}\,{\multimap_L}}{P_1P_2(P_1 \multimap P)\backslash\Box P_2\Box \longrightarrow P}\,\textbf{choice}_\Delta}{P_1P_2\backslash\Box P_2 \longrightarrow (P_1 \multimap P) \multimap P}\,{\multimap_R}}{P_1\backslash\textbf{failure} \longrightarrow P_2 \multimap (P_1 \multimap P) \multimap P}\,{\multimap_R}}{\cdot\backslash - \longrightarrow P_1 \multimap P_2 \multimap (P_1 \multimap P) \multimap P}\,{\multimap_R}$$

This derivation fails because of the unused hypothesis P_2. As written, the failure is not detected until the output of the $\multimap_R$ rule is checked. However, it is possible to tell that the derivation must fail at the $P_2\backslash P_2 \longrightarrow P_1 \gg P_1$ sequent, the rightmost leaf of the tree. At that point, all the proof branches have been completed, yet there is still an unused hypothesis. Therefore the proof must fail since this hypothesis will never be consumed.

To fail earlier, we must simply collect enough information to let the **init** rules determine whether there are any pending branches in the search tree into which the ordered and linear hypotheses will be passed. The RM3 system achieves this by maintaining separate contexts to distinguish hypotheses which will get passed on from those which will not. The frame system achieves the same end without separate contexts by taking advantage of the fact that hypotheses are always added to the right side of the linear context. Thus the linear context forms a stack-like structure where the top stack frame contains the strict formulas, *i.e.*, the ones which will not be passed on.

Rather than forming explicit stacks, one may achieve the same result as the frame system by allowing a special symbol in the context which essentially starts a new stack frame. We will use the symbol $\triangleleft$ and call it a frame pointer. A strict derivation system[1], for linear logic, using frame pointers can be characterized by the following inference rules:

$$\dfrac{}{\delta(\cdot)} \qquad \dfrac{}{\delta(\Delta\lhd)} \qquad \dfrac{\delta(\Delta)}{\delta(\Delta\Box)}$$

$$\dfrac{\delta(\Delta)}{\Delta\backslash\Delta \longrightarrow P \gg P}\,\textbf{init}$$

$$\dfrac{\Delta_I \lhd \backslash \Delta_M \lhd \longrightarrow D \gg P \qquad \Delta_M\backslash\Delta_O \longrightarrow G}{\Delta_I\backslash\Delta_O \longrightarrow G \multimap D \gg P}\,{\multimap_L}$$

$$\dfrac{\Delta_I\backslash\Delta_O \longrightarrow D \gg P \qquad \cdot\backslash\cdot \longrightarrow G}{\Delta_I\backslash\Delta_O \longrightarrow G \multimap D \gg P}\,{\rightarrow_L}$$

where all the other inference rules are unchanged. The check in the **init** rule causes eager failures. The new frame pointer is added in the $\multimap_L$ rule because all linear hypotheses in the output of the first premise will be passed into the second premise. However, in the $\rightarrow_L$ rule we do not add a frame pointer because no linear hypotheses will be passed into the second premise.

We rewrite the previous example, using frame pointers as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\delta(P_1P_2\lhd)}}{P_1P_2 \lhd \backslash P_1P_2\lhd \longrightarrow P \gg P}\,\textbf{init} \qquad \dfrac{\dfrac{\textbf{failure}}{P_2\backslash - \longrightarrow P_1 \gg P_1}\,\textbf{init}}{P_1P_2\backslash - \longrightarrow P_1}\,\textbf{choice}_\Delta}{P_1P_2\backslash - \longrightarrow P_1 \multimap P \gg P}\,{\multimap_L}}{P_1P_2(P_1 \multimap P)\backslash - \longrightarrow P}\,\textbf{choice}_\Delta}{P_1P_2\backslash - \longrightarrow (P_1 \multimap P) \multimap P}\,{\multimap_R}}{P_1\backslash - \longrightarrow P_2 \multimap (P_1 \multimap P) \multimap P}\,{\multimap_R}}{\cdot\backslash - \longrightarrow P_1 \multimap P_2 \multimap (P_1 \multimap P) \multimap P}\,{\multimap_R}$$

where the proof search fails in the **init** rule since $\delta(P_2)$ does not hold.

[1] We ignore the lazy treatment of $\top$ which is an orthogonal issue.

11.2 Extension to Ordered Contexts

Although the frame pointer machinery works well for linear contexts, it is not directly applicable to ordered contexts. There are two basic differences between linear and ordered contexts which stand in our way. Firstly, formulas are added to both sides of the ordered context, unlike the linear context in which formulas are always added on one side. Secondly, the ordered context is split between premises in the multiplicative rules, this stands in contrast to the linear context which is never syntactically split apart.

The main idea behind the extension of frames to ordered contexts is simply to add another type of frame pointer, $\triangleright$, which points to the right. The ability to start a frame from either end of the context is exactly what is needed for dealing with the deterministic ordered context splitting in our resource management system (and its extension by $\top$-flags). To see this more clearly, consider the left rules for the ordered implications (ignoring the unrestricted and linear contexts, and ignoring $\top$-flags).

$$\frac{\Omega_{LI}\backslash\Omega_{LO};\Omega_{RI}\backslash\Omega_{GI}\Omega_{RO} \longrightarrow D \gg P \qquad \Omega_{GI}\backslash\Omega_{GO} \longrightarrow G}{\Omega_{LI}\backslash\Omega_{LO};\Omega_{RI}\backslash\Omega_{GO}\Omega_{RO} \longrightarrow G \twoheadrightarrow D \gg P}$$

$(\square \notin \Omega_{GI}$ and $(\Omega_{RO} = \square\_$ or $\cdot))$

In the rule for $\twoheadrightarrow$, only the formulas (Ω_{GI}) on the left-hand side of the right output context will get passed into the pending proof branch for G.

However, in the rule for $\rightarrowtail$

$$\frac{\Omega_{LI}\backslash\Omega_{LO}\Omega_{GI};\Omega_{RI}\backslash\Omega_{RO} \longrightarrow D \gg P \qquad \Omega_{GI}\backslash\Omega_{GO} \longrightarrow G}{\Omega_{LI}\backslash\Omega_{LO}\Omega_{GO};\Omega_{RI}\backslash\Omega_{RO} \longrightarrow G \rightarrowtail D \gg P}$$

$(\square \notin \Omega_{GI}$ and $(\Omega_{LO} = \_\square$ or $\cdot))$

only formulas on the right-hand side of the left output context (Ω_{GI}) get passed on. Thus we need the ability to start frames on the left-hand side of the ordered context to correctly deal with $\twoheadrightarrow$, and on the right-hand side to deal with $\rightarrowtail$.

Following this reasoning, we will end up with rules of the form:

$$\frac{(\Omega_L = \triangleright\Omega \;\text{ or }\; \Omega_L = \Omega\triangleleft \;\text{ or }\; \Omega_L = \cdot) \;\text{ and }\; (\Omega_R = \triangleright\Omega' \;\text{ or }\; \Omega_R = \Omega'\triangleleft \;\text{ or }\; \Omega_R = \cdot)}{\Omega_L\backslash\Omega_L;\Omega_R\backslash\Omega_R \longrightarrow P \gg P}$$

$$\frac{\Omega_{LI}\triangleleft\backslash\Omega_{LO}\Omega_{GI}\triangleleft;\Omega_{RI}\backslash\Omega_{RO} \longrightarrow D \gg P \qquad \Omega_{GI}\backslash\Omega_{GO} \longrightarrow G}{\Omega_{LI}\backslash\Omega_{LO}\Omega_{GO};\Omega_{RI}\backslash\Omega_{RO} \longrightarrow G \rightarrowtail D \gg P}$$

$(\square \notin \Omega_{GI}$ and $(\Omega_{LO} = \_\square$ or $\cdot))$

$$\frac{\Omega_{LI}\backslash\Omega_{LO}\Omega_{GI};\triangleright\Omega_{RI}\backslash\triangleright\Omega_{RO}\longrightarrow D\gg P \qquad \Omega_{GI}\backslash\Omega_{GO}\longrightarrow G}{\Omega_{LI}\backslash\Omega_{LO}\Omega_{GO};\Omega_{RI}\backslash\Omega_{RO}\longrightarrow G\rightarrowtail D\gg P}$$
$$(\Box\notin\Omega_{GI} \text{ and } (\Omega_{RO}=\Box\_ \text{ or } \cdot))$$

The frame pointers are simply stating that some the portions of the ordered context will get passed on to some pending proof branch. That is why the **init** rule will accept a context as long as there is at least one pointer on the whole context. When we add a $\triangleleft$ to the left context in the rule for $\rightarrowtail$, we are stating that the rightmost portion, which does not contain $\Box$, of the left output context will be passed on to the proof branch for G.

The last detail to consider is that formulas are added on both the left and right side of the context. The system outlined above does not fully account for this as shown by the following example:

$$\cfrac{\cfrac{\cfrac{}{D\triangleleft\triangleleft\backslash D\triangleleft\triangleleft;\cdot\backslash\cdot\longrightarrow C\gg C}\textbf{init} \quad \cfrac{\cfrac{\cfrac{\cfrac{}{\cdot\backslash\cdot;BD\triangleleft\backslash BD\triangleleft\longrightarrow A\gg A}\textbf{init}}{ABD\triangleleft\backslash\Box BD\triangleleft\longrightarrow A}\textbf{choice}_{\Omega}}{BD\triangleleft\backslash BD\triangleleft\longrightarrow A\rightarrowtail A}\rightarrowtail_R}{D\triangleleft\backslash\textbf{failure}\longrightarrow B\rightarrowtail A\rightarrowtail A}\rightarrowtail_R}{D\triangleleft\backslash -;\cdot\backslash -\longrightarrow(B\rightarrowtail A\rightarrowtail A)\rightarrowtail C\gg C}\rightarrowtail_L \quad \text{not done}}{\cfrac{D\backslash -;\cdot\backslash -\longrightarrow D\rightarrowtail(B\rightarrowtail A\rightarrowtail A)\rightarrowtail C\gg C}{D(D\rightarrowtail(B\rightarrowtail A\rightarrowtail A)\rightarrowtail C)\backslash -\longrightarrow C}\textbf{choice}_{\Omega}}\rightarrowtail_L$$

The failure does not occur until the $\rightarrowtail_R$ rule explicitly checks that the B was consumed. We can allow the the failure occur in the **init** rule by constraining the scope of the frame pointer to not include hypotheses added to the context after the frame pointer. We will achieve this scoping by tagging hypotheses and frame pointers with an integer, or "level", and carrying around the current level on the sequent arrow.

We make the following definition:

$$\overline{\Omega}=\max\{n|D^n\in\Omega\}$$

and define the following relation which plays a similar role for ordered contexts as δ plays for linear contexts:

$$\frac{}{o(\cdot)} \qquad \frac{\overline{\Omega}\leq m}{o(\overset{m}{\triangleright}\Omega)} \qquad \frac{\overline{\Omega}\leq m}{o(\Omega\overset{m}{\triangleleft})}$$

With the above considerations and definitions, our rules take the following form:

$$\frac{o(\Omega_L) \qquad o(\Omega_R)}{\Omega_L \backslash \Omega_L ; \Omega_R \backslash \Omega_R \xrightarrow{n} P \gg P}$$

$$\frac{\Omega_{LI} \overset{n}{\triangleleft} \backslash \Omega_{LO} \Omega_{GI} \overset{n}{\triangleleft} ; \Omega_{RI} \backslash \Omega_{RO} \xrightarrow{n} D \gg P \qquad \Omega_{GI} \backslash \Omega_{GO} \xrightarrow{n+1} G}{\Omega_{LI} \backslash \Omega_{LO} \Omega_{GO} ; \Omega_{RI} \backslash \Omega_{RO} \xrightarrow{n} G \rightarrowtail D \gg P}$$

$(\Box \notin \Omega_{GI}$ and $(\Omega_{LO} = \_\Box$ or $\cdot))$

$$\frac{\Omega_{LI} \backslash \Omega_{LO} \Omega_{GI} ; \overset{n}{\triangleright} \Omega_{RI} \backslash \overset{n}{\triangleright} \Omega_{RO} \xrightarrow{n} D \gg P \qquad \Omega_{GI} \backslash \Omega_{GO} \xrightarrow{n+1} G}{\Omega_{LI} \backslash \Omega_{LO} \Omega_{GO} ; \Omega_{RI} \backslash \Omega_{RO} \xrightarrow{n} G \rightarrowtail D \gg P}$$

$(\Box \notin \Omega_{GI}$ and $(\Omega_{RO} = \Box\_$ or $\cdot))$

$$\frac{\Omega_I D^n \backslash \Omega_O \Box \xrightarrow{n} G}{\Omega_I \backslash \Omega_O \xrightarrow{n} D \twoheadrightarrow G} \twoheadrightarrow_R \qquad \frac{D^n \Omega_I \backslash \Box \Omega_O \xrightarrow{n} G}{\Omega_I \backslash \Omega_O \xrightarrow{n} D \rightarrowtail G} \rightarrowtail_R$$

Note that the derivation rules maintain the following invariant. Every unconsumed portion of an output context will be flanked by a frame pointer, whose tag is at least as big as all the formula tags in the portion, explicitly denoting that the portion will be passed into a pending proof branch. This invariant will also hold for the full strict derivation system presented in Section 11.3.

11.3 Strict Derivation System

This section presents the complete strict derivation system which extends the $\top$-flags system of Chapter 9.

As usual we will have two types of sequents

$$\Gamma ; \Delta_I \backslash \Delta_O ; \Omega_I \backslash \Omega_O \underset{v \ \tau}{\xrightarrow{n}} G$$
$$\Gamma ; \Delta_I \backslash \Delta_O ; (\Omega_{LI} \backslash \Omega_{LO} \ ; \ \Omega_{RI} \backslash \Omega_{RO}) \underset{v \ (\tau_L \ ; \ \tau_R)}{\xrightarrow{n}} D \gg P$$

where n is a natural number; input contexts, Ω_{xI}, are lists of tagged clause formulas, G^n, and pointers, $\overset{n}{\triangleleft}$ and $\overset{n}{\triangleright}$; and output contexts, Ω_{xO}, are lists of tagged clause formulas, placeholders and pointers; Γ, Δ_x and τ_x are unchanged from the $\top$-flags system.

Before showing the strict inference rules, we need to extend all of the machinery used in ⊤-flags derivations to work on strict contexts. We begin by extending $\sqsupseteq$ to work on strict linear and ordered contexts:

$$\frac{}{\cdot \sqsupseteq \cdot} \qquad \frac{\Psi \sqsupseteq \Psi'}{\Psi D^m \sqsupseteq \Psi' D^m} \qquad \frac{\Psi \sqsupseteq \Psi'}{\Psi D^m \sqsupseteq \Psi' \square} \qquad \frac{\Psi \sqsupseteq \Psi'}{\Psi \overset{n}{\triangleleft} \sqsupseteq \Psi' \overset{n}{\triangleleft}} \qquad \frac{\Psi \sqsupseteq \Psi'}{\Psi \overset{n}{\triangleright} \sqsupseteq \Psi' \overset{n}{\triangleright}} \qquad \frac{\Psi \sqsupseteq \Psi'}{\Psi \triangleleft \sqsupseteq \Psi' \triangleleft}$$

We next consider **mrg** for strict ordered contexts:

$$\frac{}{\mathbf{mrg}(\cdot, \tau_1, \cdot, \tau_2, \cdot, \tau_1 * \tau_2)} \qquad \frac{\mathbf{mrg}(\Omega_1, \tau_1, \Omega_2, \tau_2, \Omega, \tau)}{\mathbf{mrg}(D^m\Omega_1, \tau_1, D^m\Omega_2, \tau_2, D^m\Omega, \tau)}$$

$$\frac{\mathbf{mrg}(\Omega_1, 1\tau_1, \Omega_2, \tau_2, \Omega, \tau)}{\mathbf{mrg}(D^m\Omega_1, 1\tau_1, \square\Omega_2, \tau_2'0\tau_2, \square\Omega, \tau_2'0\tau)} (0 \notin \tau_2')$$

$$\frac{\mathbf{mrg}(\Omega_1, \tau_1, \Omega_2, 1\tau_2, \Omega, \tau)}{\mathbf{mrg}(\square\Omega_1, \tau_1'0\tau_1, D^m\Omega_2, 1\tau_2, \square\Omega, \tau_1'0\tau)} (0 \notin \tau_1')$$

$$\frac{\mathbf{mrg}(\Omega_1, \tau_1, \Omega_2, \tau_2, \Omega, \tau)}{\mathbf{mrg}(\square\Omega_1, \tau_1'0\tau_1, \square\Omega_2, \tau_2'0\tau_2, \square\Omega, (\tau_1' * \tau_2')0\tau)} (0 \notin \tau_i')$$

$$\frac{\mathbf{mrg}(\Omega_1, \tau_1, \Omega_2, \tau_2, \Omega, \tau)}{\mathbf{mrg}(\overset{n}{\triangleright}\Omega_1, \tau_1, \overset{n}{\triangleright}\Omega_2, \tau_2, \overset{n}{\triangleright}\Omega, \tau)} \qquad \frac{\mathbf{mrg}(\Omega_1, \tau_1, \Omega_2, \tau_2, \Omega, \tau)}{\mathbf{mrg}(\overset{n}{\triangleleft}\Omega_1, \tau_1, \overset{n}{\triangleleft}\Omega_2, \tau_2, \overset{n}{\triangleleft}\Omega, \tau)}$$

Finally we extend **mrgL** to strict linear contexts.

$$\frac{}{\mathbf{mrgL}(\cdot, v_1, \cdot, v_2, \cdot, v_1 \wedge v_2)} \qquad \frac{\mathbf{mrgL}(\Delta_1, v_1, \Delta_2, v_2, \Delta, v)}{\mathbf{mrgL}(\triangleleft\Delta_1, v_1, \triangleleft\Delta_2, v_2, \triangleleft\Delta, v)}$$

$$\frac{\mathbf{mrgL}(\Delta_1, v_1, \Delta_2, v_2, \Delta, v)}{\mathbf{mrgL}(D\Delta_1, v_1, D\Delta_2, v_2, D\Delta, v)} \qquad \frac{\mathbf{mrgL}(\Delta_1, v_1, \Delta_2, v_2, \Delta, v)}{\mathbf{mrgL}(\square\Delta_1, v_1, \square\Delta_2, v_2, \square\Delta, v)}$$

$$\frac{\mathbf{mrgL}(\Delta_1, T, \Delta_2, v_2, \Delta, v)}{\mathbf{mrgL}(D\Delta_1, T, \square\Delta_2, v_2, \square\Delta, v)} \qquad \frac{\mathbf{mrgL}(\Delta_1, v_1, \Delta_2, T, \Delta, v)}{\mathbf{mrgL}(\square\Delta_1, v_1, D\Delta_2, T, \square\Delta, v)}$$

We state the following lemma which we will use in the proof of Lemma 42.

Lemma 36 $\mathbf{mrgL}(\Delta_1, v_1, \Delta_2, v_2, \Delta, v)$ *and* $\delta(\Delta_1)$ *and* $\delta(\Delta_2)$ *implies* $\delta(\Delta)$.

Proof: By structural induction on the given derivation. □

We may now write the inference rules for strict derivations:

$$\frac{\delta(\Delta) \qquad o(\Omega)}{\Gamma; \Delta\backslash\Delta; \Omega\backslash\Omega \ \xrightarrow[F\ \cdot]{n} \ P \doteq P} \doteq_R$$

$$\frac{\delta(\Delta) \qquad o(\Omega) \qquad \Gamma; \cdot\backslash\cdot; \cdot\backslash\cdot \ \xrightarrow[v\ \tau]{n} \ G}{\Gamma; \Delta\backslash\Delta; \Omega\backslash\Omega \ \xrightarrow[F\ \cdot]{n} \ !G} !_R$$

$$\frac{o(\Omega) \qquad \Gamma; \Delta_I\backslash\Delta_O; \cdot\backslash\cdot \ \xrightarrow[v\ \tau]{n} \ G}{\Gamma; \Delta_I\backslash\Delta_O; \Omega\backslash\Omega \ \xrightarrow[v\ \cdot]{n} \ !G} ¡_R$$

$$\frac{\Gamma D; \Delta_I\backslash\Delta_O; \Omega_I\backslash\Omega_O \ \xrightarrow[v\ \tau]{n} \ G}{\Gamma; \Delta_I\backslash\Delta_O; \Omega_I\backslash\Omega_O \ \xrightarrow[v\ \tau]{n} \ D \to G} \to_R$$

$$\frac{\Gamma; \Delta_I D\backslash\Delta_O\Box; \Omega_I\backslash\Omega_O \ \xrightarrow[F\ \tau]{n} \ G}{\Gamma; \Delta_I\backslash\Delta_O; \Omega_I\backslash\Omega_O \ \xrightarrow[F\ \tau]{n} \ D \to G} \multimap_{RF}$$

$$\frac{\Gamma; \Delta_I D\backslash\Delta_O X; \Omega_I\backslash\Omega_O \ \xrightarrow[T\ \tau]{n} \ G}{\Gamma; \Delta_I\backslash\Delta_O; \Omega_I\backslash\Omega_O \ \xrightarrow[T\ \tau]{n} \ D \to G} \multimap_{RT}$$

$(X = \Box \text{ or } X = D)$

$$\frac{\Gamma; \Delta_I\backslash\Delta_O; \Omega_I D^n\backslash\Omega_O\Box \ \xrightarrow[v\ \tau 0\tau']{n} \ G}{\Gamma; \Delta_I\backslash\Delta_O; \Omega_I\backslash\Omega_O \ \xrightarrow[v\ \tau]{n} \ D \twoheadrightarrow G} \twoheadrightarrow_{R0} \quad (0 \notin \tau')$$

$$\frac{\Gamma; \Delta_I\backslash\Delta_O; \Omega_I D^n\backslash\Omega_O D^n \ \xrightarrow[v\ \tau 1]{n} \ G}{\Gamma; \Delta_I\backslash\Delta_O; \Omega_I\backslash\Omega_O \ \xrightarrow[v\ \tau 1]{n} \ D \twoheadrightarrow G} \twoheadrightarrow_{R1}$$

$$\frac{\Gamma; \Delta_I\backslash\Delta_O; D^n\Omega_I\backslash\Box\Omega_O \ \xrightarrow[v\ \tau' 0\tau]{n} \ G}{\Gamma; \Delta_I\backslash\Delta_O; \Omega_I\backslash\Omega_O \ \xrightarrow[v\ \tau]{n} \ D \rightarrowtail G} \rightarrowtail_{R0} \quad (0 \notin \tau')$$

$$\frac{\Gamma; \Delta_I\backslash\Delta_O; D^n\Omega_I\backslash D^n\Omega_O \ \xrightarrow[v\ 1\tau]{n} \ G}{\Gamma; \Delta_I\backslash\Delta_O; \Omega_I\backslash\Omega_O \ \xrightarrow[v\ 1\tau]{n} \ D \rightarrowtail G} \rightarrowtail_{R1}$$

$$\dfrac{\Gamma; \Delta_I \triangleleft \backslash \Delta_M \triangleleft; \Omega_I \overset{n}{\triangleleft} \backslash \Omega_L \Omega_2 \overset{n}{\triangleleft} \xrightarrow[v_1 \ \tau_L]{(n+1)} G_1 \qquad \Gamma; \Delta_M \backslash \Delta_O; \Omega_2 \backslash \Omega_R \xrightarrow[v_2 \ \tau_R]{(n+1)} G_2}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_L \Omega_R \xrightarrow[(v_1 \vee v_2) \ (\tau_L + \tau_R)]{n} G_1 \bullet G_2} \bullet_{R0}$$

($\Box \notin \Omega_2$ and ($\Omega_L = \_\Box$ or $\cdot$) and $\tau_L \neq \tau 1$)

$$\dfrac{\Gamma; \Delta_I \triangleleft \backslash \Delta_M \triangleleft; \Omega_I \overset{n}{\triangleleft} \backslash \Omega_L \Omega_2 \overset{n}{\triangleleft} \xrightarrow[v_1 \ \tau_L 1]{(n+1)} G_1 \qquad \Gamma; \Delta_M \backslash \Delta_O; \overset{n}{\triangleright} \Omega_2 \backslash \overset{n}{\triangleright} \Omega_R \xrightarrow[v_2 \ \tau_R]{(n+1)} G_2}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_L \Omega_R \xrightarrow[(v_1 \vee v_2) \ (\tau_L + \tau_R)]{n} G_1 \bullet G_2} \bullet_{R1}$$

($\Box \notin \Omega_2$ and ($\Omega_L = \_\Box$ or $\cdot$))

$$\dfrac{\Gamma; \Delta_I \triangleleft \backslash \Delta_M \triangleleft; \overset{n}{\triangleright} \Omega_I \backslash \overset{n}{\triangleright} \Omega_2 \Omega_R \xrightarrow[v_1 \ \tau_R]{(n+1)} G_1 \qquad \Gamma; \Delta_M \backslash \Delta_O; \Omega_2 \backslash \Omega_L \xrightarrow[v_2 \ \tau_L]{(n+1)} G_2}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_L \Omega_R \xrightarrow[(v_1 \vee v_2) \ (\tau_L + \tau_R)]{n} G_1 \circ G_2} \circ_{R0}$$

($\Box \notin \Omega_2$ and ($\Omega_R = \Box\_$ or $\cdot$) and $\tau_R \neq 1\tau$)

$$\dfrac{\Gamma; \Delta_I \triangleleft \backslash \Delta_M \triangleleft; \overset{n}{\triangleright} \Omega_I \backslash \overset{n}{\triangleright} \Omega_2 \Omega_R \xrightarrow[v_1 \ 1\tau_R]{(n+1)} G_1 \qquad \Gamma; \Delta_M \backslash \Delta_O; \Omega_2 \overset{n}{\triangleleft} \backslash \Omega_L \overset{n}{\triangleleft} \xrightarrow[v_2 \ \tau_L]{(n+1)} G_2}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_L \Omega_R \xrightarrow[(v_1 \vee v_2) \ (\tau_L + 1\tau_R)]{n} G_1 \circ G_2} \circ_{R1}$$

($\Box \notin \Omega_2$ and ($\Omega_R = \Box\_$ or $\cdot$))

$$\dfrac{\delta(\Delta) \qquad o(\Omega)}{\Gamma; \Delta \backslash \Delta; \Omega \backslash \Omega \xrightarrow[F \ \cdot]{n} 1} 1_R \qquad\qquad \dfrac{}{\Gamma; \Delta \backslash \Delta; \Omega \backslash \Omega \xrightarrow[T \ 1]{n} \top} \top_R$$

$$\dfrac{\Gamma; \Delta_I \backslash \Delta_1; \Omega_I \backslash \Omega_1 \xrightarrow[v_1 \ \tau_1]{n} G_1 \quad \Gamma; \Delta_I \backslash \Delta_2; \Omega_I \backslash \Omega_2 \xrightarrow[v_2 \ \tau_2]{n} G_2 \quad \begin{array}{l} \mathbf{mrg}(\Omega_1, \tau_1, \Omega_2, \tau_2, \Omega_O, \tau) \\ \mathbf{mrgL}(\Delta_1, v_1, \Delta_2, v_2, \Delta_O, v) \end{array}}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \xrightarrow[v \ \tau]{n} G_1 \,\&\, G_2} \&_R$$

$$\dfrac{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \xrightarrow[v \ \tau]{n} G_1}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \xrightarrow[v \ \tau]{n} G_1 \oplus G_2} \oplus_{R1} \qquad\qquad \dfrac{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \xrightarrow[v \ \tau]{n} G_2}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \xrightarrow[v \ \tau]{n} G_1 \oplus G_2} \oplus_{R2}$$

$$\frac{1; D \gg P \setminus G \qquad \Gamma_L D G_R; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \;\underset{v\;\;\tau}{\xrightarrow{n}}\; G}{\Gamma_L D \Gamma_R; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \;\underset{v\;\;\tau}{\xrightarrow{n}}\; P}\,\text{choice}_\Gamma$$

$$\frac{1; D \gg P \setminus G \qquad \Gamma; \Delta_{LI} \Box \Delta_{RI} \backslash \Delta_O; \Omega_I \backslash \Omega_O \;\underset{v\;\;\tau}{\xrightarrow{n}}\; G}{\Gamma; \Delta_{LI} D \Delta_{RI} \backslash \Delta_O; \Omega_I \backslash \Omega_O \;\underset{v\;\;\tau}{\xrightarrow{n}}\; P}\,\text{choice}_\Delta$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \;;\; \Omega_{RI} \backslash \Omega_{RO}) \;\underset{v\;\;(\tau_L\;;\;\tau_R)}{\xrightarrow{n}}\; D \gg P}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_{LI} D^m \Omega_{RI} \backslash \Omega_{LO} \Box \Omega_{RO} \;\underset{v\;\;\tau_L 0 \tau_R}{\xrightarrow{n}}\; P}\,\text{choice}_\Omega$$

$$\frac{\delta(\Delta) \qquad o(\Omega_L) \qquad o(\Omega_R)}{\Gamma; \Delta \backslash \Delta; (\Omega_L \backslash \Omega_L \;;\; \Omega_R \backslash \Omega_R) \;\underset{F\;\;(\cdot\;;\;\cdot)}{\xrightarrow{n}}\; P \gg P}\,\textbf{init}$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \;;\; \Omega_{RI} \backslash \Omega_{RO}) \;\underset{v\;\;(\tau_L\;;\;\tau_R)}{\xrightarrow{n}}\; D_1 \gg P}{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \;;\; \Omega_{RI} \backslash \Omega_{RO}) \;\underset{v\;\;(\tau_L\;;\;\tau_R)}{\xrightarrow{n}}\; D_1 \mathbin{\&} D_2 \gg P}\,\&_{L1}$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \;;\; \Omega_{RI} \backslash \Omega_{RO}) \;\underset{v\;\;(\tau_L\;;\;\tau_R)}{\xrightarrow{n}}\; D_2 \gg P}{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \;;\; \Omega_{RI} \backslash \Omega_{RO}) \;\underset{v\;\;(\tau_L\;;\;\tau_R)}{\xrightarrow{n}}\; D_1 \mathbin{\&} D_2 \gg P}\,\&_{L2}$$

$$\frac{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \;;\; \Omega_{RI} \backslash \Omega_{RO}) \;\underset{v\;\;(\tau_L\;;\;\tau_R)}{\xrightarrow{n}}\; D \gg P \qquad \Gamma; \cdot \backslash \cdot; \cdot \backslash \cdot \;\underset{v'\;\;\tau'}{\xrightarrow{n}}\; G}{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \;;\; \Omega_{RI} \backslash \Omega_{RO}) \;\underset{v\;\;(\tau_L\;;\;\tau_R)}{\xrightarrow{n}}\; G \to D \gg P}\,\to_L$$

$$\frac{\Gamma; \Delta_I \lhd \backslash \Delta_M \lhd; (\Omega_{LI} \backslash \Omega_{LO} \;;\; \Omega_{RI} \backslash \Omega_{RO}) \;\underset{v_1\;\;(\tau_L\;;\;\tau_R)}{\xrightarrow{n}}\; D \gg P \qquad \Gamma; \Delta_M \backslash \Delta_O; \cdot \backslash \cdot \;\underset{v_2\;\;\tau}{\xrightarrow{n}}\; G}{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \;;\; \Omega_{RI} \backslash \Omega_{RO}) \;\underset{(v_1 \vee v_2)\;\;(\tau_L\;;\;\tau_R)}{\xrightarrow{n}}\; G \multimap D \gg P}\,\multimap_L$$

$$\dfrac{\begin{array}{l}\Gamma; \Delta_I \triangleleft \backslash \Delta_M \triangleleft; (\Omega_{LI} \backslash \Omega_{LO} \,;\, \overset{n}{\triangleright}\Omega_{RI} \backslash \overset{n}{\triangleright} \Omega_{GI}\Omega_{RO}) \quad {}_{v_1}\xrightarrow[(\tau_L\,;\,\tau_R)]{n} \quad D \gg P \\ \Gamma; \Delta_M \backslash \Delta_O; \Omega_{GI} \backslash \Omega_{GO} \;\; {}_{v_2}\xrightarrow[\tau]{(n+1)} \; G\end{array}}{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \,;\, \Omega_{RI} \backslash \Omega_{GO}\Omega_{RO}) \quad {}_{(v_1 \vee v_2)}\xrightarrow[(\tau_L\,;\,\tau+\tau_R)]{n} \quad G \twoheadrightarrow D \gg P} \; {\twoheadrightarrow}_{L0}$$

$(\square \notin \Omega_{GI}$ and $(\Omega_{RO} = \square\_$ or $\cdot)$ and $\tau_R \neq 1\tau)$

$$\dfrac{\begin{array}{l}\Gamma; \Delta_I \triangleleft \backslash \Delta_M \triangleleft; (\Omega_{LI} \backslash \Omega_{LO} \,;\, \overset{n}{\triangleright}\Omega_{RI} \backslash \overset{n}{\triangleright} \Omega_{GI}\Omega_{RO}) \quad {}_{v_1}\xrightarrow[(\tau_L\,;\,1\tau_R)]{n} \quad D \gg P \\ \Gamma; \Delta_M \backslash \Delta_O; \Omega_{GI} \overset{n}{\triangleleft} \backslash \Omega_{GO} \overset{n}{\triangleleft} \;\; {}_{v_2}\xrightarrow[\tau]{(n+1)} \; G\end{array}}{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \,;\, \Omega_{RI} \backslash \Omega_{GO}\Omega_{RO}) \quad {}_{(v_1 \vee v_2)}\xrightarrow[(\tau_L\,;\,\tau+1\tau_R)]{n} \quad G \twoheadrightarrow D \gg P} \; {\twoheadrightarrow}_{L1}$$

$(\square \notin \Omega_{GI}$ and $(\Omega_{RO} = \square\_$ or $\cdot))$

$$\dfrac{\begin{array}{l}\Gamma; \Delta_I \triangleleft \backslash \Delta_M \triangleleft; (\Omega_{LI} \overset{n}{\triangleleft} \backslash \Omega_{LO}\Omega_{GI} \overset{n}{\triangleleft} ;\, \Omega_{RI} \backslash \Omega_{RO}) \quad {}_{v_1}\xrightarrow[(\tau_L\,;\,\tau_R)]{n} \quad D \gg P \\ \Gamma; \Delta_M \backslash \Delta_O; \Omega_{GI} \backslash \Omega_{GO} \;\; {}_{v_2}\xrightarrow[\tau]{(n+1)} \; G\end{array}}{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \,;\, \Omega_{RI} \backslash \Omega_{GO}\Omega_{RO}) \quad {}_{(v_1 \vee v_2)}\xrightarrow[(\tau_L+\tau\,;\,\tau_R)]{n} \quad G \rightarrowtail D \gg P} \; {\rightarrowtail}_{L0}$$

$(\square \notin \Omega_{GI}$ and $(\Omega_{LO} = \_\square$ or $\cdot)$ and $\tau_L \neq \tau 1)$

$$\dfrac{\begin{array}{l}\Gamma; \Delta_I \triangleleft \backslash \Delta_M \triangleleft; (\Omega_{LI} \overset{n}{\triangleleft} \backslash \Omega_{LO}\Omega_{GI} \overset{n}{\triangleleft} ;\, \Omega_{RI} \backslash \Omega_{RO}) \quad {}_{v_1}\xrightarrow[(\tau_L 1\,;\,\tau_R)]{n} \quad D \gg P \\ \Gamma; \Delta_M \backslash \Delta_O; \overset{n}{\triangleright}\Omega_{GI} \backslash \overset{n}{\triangleright} \Omega_{GO} \;\; {}_{v_2}\xrightarrow[\tau]{(n+1)} \; G\end{array}}{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO} \,;\, \Omega_{RI} \backslash \Omega_{GO}\Omega_{RO}) \quad {}_{(v_1 \vee v_2)}\xrightarrow[(\tau_L 1+\tau\,;\,\tau_R)]{n} \quad G \rightarrowtail D \gg P} \; {\rightarrowtail}_{L1}$$

$(\square \notin \Omega_{GI}$ and $(\Omega_{LO} = \_\square$ or $\cdot))$

11.4 Correctness of Strict Derivations

Proving the soundness of strict derivations wrt $\top$-flags derivations is trivial since each strict derivation, when stripped of tags and pointers, is a $\top$-flags derivation.

Theorem 37

1. $\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \xrightarrow[v\ \tau]{n} G$ *implies* $\exists \Delta'_I, \Delta'_O, \Omega'_I, \Omega'_O.$
   $\Gamma; \Delta'_I \backslash \Delta'_O; \Omega'_I \backslash \Omega'_O \xrightarrow[v\ \tau]{} G$

2. $\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{RO}) \xrightarrow[v\ (\tau_L\ ;\ \tau_R)]{n} D \gg P$ *implies*
   $\exists \Delta'_I, \Delta'_O, \Omega'_{LI}, \Omega'_{LO}, \Omega'_{RI}, \Omega'_{RO}.$
   $\Gamma; \Delta'_I \backslash \Delta'_O; (\Omega'_{LI} \backslash \Omega'_{LO}\ ;\ \Omega'_{RI} \backslash \Omega'_{RO}) \xrightarrow[v\ (\tau_L\ ;\ \tau_R)]{} D \gg P$

Proof: Structural induction on given derivation. □

In order to prove the other direction, we introduce some machinery to relate ⊤-flags contexts to strict contexts.

We start with Θ which relates a ⊤-flags context, not containing □, to a strict context.

$$\frac{}{\Theta^n(\cdot,\cdot)} \qquad \frac{\Theta^n(\Omega,\Omega')}{\Theta^n(\Omega,\overset{m}{\triangleright}\Omega')}(m \le n) \qquad \frac{\Theta^n(\Omega,\Omega')}{\Theta^n(\Omega,\overset{m}{\triangleleft}\Omega')}(m \le n) \qquad \frac{\Theta^n(\Omega,\Omega')}{\Theta^n(D\Omega,D^m\Omega')}(m \le n)$$

We will make use of the following properties of Θ.

Lemma 38

1. $\Theta^n(\Omega_L\Omega_R,\Omega')$ *implies*
   $\exists \Omega'_L, \Omega'_R.\ \Omega'_L\Omega'_R = \Omega'$ *and* $\Theta^n(\Omega_L,\Omega'_L)$ *and* $\Theta^n(\Omega_R,\Omega'_R).$

2. $\Theta^n(\Omega_L,\Omega'_L)$ *and* $\Theta^n(\Omega_R,\Omega'_R)$ *implies* $\Theta^n(\Omega_L\Omega_R,\Omega'_L\Omega'_R).$

3. $\Theta^n(\Omega,\Omega'_L\Omega'_R)$ *and* $m \le n$ *implies* $\Theta^n(\Omega,\Omega'_L \overset{m}{\triangleright} \Omega'_R)$ *and* $\Theta^n(\Omega,\Omega'_L \overset{m}{\triangleleft}\ \Omega'_R).$

4. $\Theta^m(\Omega,\Omega')$ *and* $m \le n$ *implies* $\Theta^n(\Omega,\Omega')$.

5. $\Theta^n(\Omega,\Omega')$ *implies* $\overline{\Omega'} \le n.$

Proof: By structural induction on given derivation. □

We now relate ⊤-flags input/output context pairs to strict input/output context pairs.

$$\frac{}{\Xi^n(\cdot\backslash\cdot,\cdot\backslash\cdot)} \qquad \frac{\Xi^n(\Omega\backslash\Omega,\Omega'\backslash\Omega') \qquad \Xi^n(\Omega_I\backslash\Omega_O,\Omega'_I\backslash\Omega'_O)}{\Xi^n(\Omega D\Omega_I\backslash\Omega\Box\Omega_O,\Omega' D^m\Omega'_I\backslash\Omega'\Box\Omega'_O)}\ (\Box\notin\Omega, m\le n)$$

$$\frac{\Theta^m(\Omega,\Omega')}{\Xi^n(\Omega\backslash\Omega,\Omega' \overset{m}{\triangleleft} \backslash\Omega'\overset{m}{\triangleleft})}\ (\Box\notin\Omega, m\le n) \qquad \frac{\Theta^m(\Omega,\Omega')}{\Xi^n(\Omega\backslash\Omega,\overset{m}{\triangleright}\Omega'\backslash \overset{m}{\triangleright} \Omega')}\ (\Box\notin\Omega, m\le n)$$

Since we will only translate successful ⊤-flags derivations to strict derivations, we must ensure that the translated output contexts satisfy the invariant on strictness derivations stated at the end of Section 11.2.

We make use of the following properties of Ξ.

Lemma 39

1. $\Xi^n(\Omega\backslash\Omega,\Omega'_I\backslash\Omega'_O)$ *implies* $\Omega'_I=\Omega'_O$.

2. $\Xi^n(\Omega_I\backslash\Omega_O,\Omega'_I\backslash\Omega'_O)$ *implies* $\Theta^n(\Omega_I,\Omega'_I)$.

3. $\Xi^n(\Omega_{LI}D\Omega_{RI}\backslash\Omega_{LO}\Box\Omega_{RO},\Omega'_I\backslash\Omega'_O)$ *and* $\Omega_{LI}\sqsupseteq\Omega_{LO}$ *and* $m\le n$
   implies $\exists\Omega'_{LI},\Omega'_{RI},\Omega'_{LO},\Omega'_{RO}$.
   $\Omega'_I=\Omega'_{LI}D^m\Omega'_{RI}$ *and* $\Omega'_O=\Omega'_{LO}\Box\Omega'_{RO}$ *and*
   $\Xi^n(\Omega_{LI}\backslash\Omega_{LO},\Omega'_{LI}\backslash\Omega'_{LO})$ *and* $\Xi^n(\Omega_{RI}\backslash\Omega_{RO},\Omega'_{RI}\backslash\Omega'_{RO})$.

4. $\Xi^n(\Omega_{LI}\backslash\Omega_{LO},\Omega'_{LI}\backslash\Omega'_{LO})$ *and* $\Xi^n(\Omega_{RI}\backslash\Omega_{RO},\Omega'_{RI}\backslash\Omega'_{RO})$ *and* $m\le n$ *implies*
   $\Xi^n(\Omega_{LI}D\Omega_{RI}\backslash\Omega_{LO}\Box\Omega_{RO},\Omega'_{LI}D^m\Omega'_{RI}\backslash\Omega'_{LO}\Box\Omega'_{RO})$.

5. $\Xi^n(\Omega_I\backslash\Omega_O,\Omega'_{LI}\Omega'_{RI}\backslash\Omega'_{LO}\Omega'_{RO})$ *implies*
   $\Xi^n(\Omega_I\backslash\Omega_O,\Omega'_{LI}\overset{n}{\triangleright}\Omega_{RI}\backslash\Omega'_{LO}\overset{n}{\triangleright}\Omega'_{RO})$ *and* $\Xi^n(\Omega_I\backslash\Omega_O,\Omega'_{LI}\overset{n}{\triangleright}\Omega_{RI}\backslash\Omega'_{LO}\overset{n}{\triangleleft}\Omega'_{RO})$.

6. $\Xi^m(\Omega_I\backslash\Omega_O,\Omega'_I\backslash\Omega'_O)$ *and* $m\le n$ *implies* $\Xi^n(\Omega_I\backslash\Omega_O,\Omega'_I\backslash\Omega'_O)$.

7. $\Xi^n(\Omega_I\backslash\Omega_O,\Omega'_I\backslash\Omega'_O)$ *implies* $\overline{\Omega'_I}\le n$ *and* $\overline{\Omega'_O}\le n$.

Proof: By induction on the given derivation with appeals to Lemma 38.
For part 4, induct on the derivation for $\Xi^n(\Omega_{LI}\backslash\Omega_{LO},\Omega'_{LI}\backslash\Omega'_{LO})$ and consider cases for $\Xi^n(\Omega_{RI}\backslash\Omega_{RO},\Omega'_{RI}\backslash\Omega'_{RO})$. □

We now turn our attention to linear contexts.

$$\frac{}{\epsilon(\cdot\backslash\cdot, \cdot\backslash\cdot)} \qquad \frac{\epsilon(\Delta_I\backslash\Delta_O, \Delta'_I\backslash\Delta'_O)}{\epsilon(\Delta_I\backslash\Delta_O, \triangleleft\Delta'_I\backslash \triangleleft \Delta'_O)}$$

$$\frac{\epsilon(\Delta_I\backslash\Delta_O, \Delta'_I\backslash\Delta'_O)}{\epsilon(D\Delta_I\backslash D\Delta_O, D\Delta'_I\backslash D\Delta'_O)} \quad \frac{\epsilon(\Delta_I\backslash\Delta_O, \Delta'_I\backslash\Delta'_O)}{\epsilon(D\Delta_I\backslash\Box\Delta_O, D\Delta'_I\backslash\Box\Delta'_O)} \quad \frac{\epsilon(\Delta_I\backslash\Delta_O, \Delta'_I\backslash\Delta'_O)}{\epsilon(\Box\Delta_I\backslash\Box\Delta_O, \Box\Delta'_I\backslash\Box\Delta'_O)}$$

Lemma 40

1. $\epsilon(\Delta\backslash\Delta, \Delta'_I\backslash\Delta'_O)$ *implies* $\Delta'_I = \Delta'_O$.

2. $\epsilon(\Delta_I\backslash\Delta_O, \Delta'_I\backslash\Delta'_O)$ *and* $\Delta_I\backslash\Delta_M$ *and* $\Delta_M \sqsupseteq \Delta_O$
implies there exists Δ'_M *such that*
$\epsilon(\Delta_I\backslash\Delta_M, \Delta'_I\backslash\Delta'_M)$ *and* $\epsilon(\Delta_M\backslash\Delta_O, \Delta'_M\backslash\Delta'_O)$.

Proof: By induction on the structure of the given ϵ derivation. □

We can now show the completeness of strict derivations wrt to $\top$-flags derivations.

Theorem 41

1. $\Gamma; \Delta_I\backslash\Delta_O; \Omega_I\backslash\Omega_O \underset{v\ \tau}{\longrightarrow} G$ *and* $\delta(\Delta'_O)$ *and* $\epsilon(\Delta_I\backslash\Delta_O, \Delta'_I\backslash\Delta'_O)$ *and* $\Xi^n(\Omega_I\backslash\Omega_O, \Omega_O\backslash\Omega'_O)$
implies $\Gamma; \Delta'_I\backslash\Delta'_O; \Omega'_I\backslash\Omega'_O \xrightarrow[v\ \tau]{n} G$

2. $\Gamma; \Delta_I\backslash\Delta_O; (\Omega_{LI}\backslash\Omega_{LO}; \Omega_{RI}\backslash\Omega_{RO}) \underset{v\ (\tau_L\ ;\ \tau_R)}{\longrightarrow} D \gg P$ *and* $\delta(\Delta'_O)$ *and* $\epsilon(\Delta_I\backslash\Delta_O, \Delta'_I\backslash\Delta'_O)$
and $\Xi^n(\Omega_{LI}\backslash\Omega_{LO}, \Omega'_{LI}\backslash\Omega'_{LO})$ *and* $\Xi^n(\Omega_{RI}\backslash\Omega_{RO}, \Omega'_{RI}\backslash\Omega'_{RO})$
implies $\Gamma; \Delta'_I\backslash\Delta'_O; (\Omega'_{LI}\backslash\Omega'_{LO}\ ;\ \Omega'_{RI}\backslash\Omega'_{RO}) \xrightarrow[v\ (\tau_L\ ;\ \tau_R)]{n} D \gg P$

Proof: Structural induction on given strict derivation making use of lemmas 38 and 39. We give representative cases.

case:

$$\frac{}{\Gamma; \Delta\backslash\Delta; (\Omega_L\backslash\Omega_L\ ;\ \Omega_R\backslash\Omega_R) \underset{F\ (\cdot\ ;\ \cdot)}{\longrightarrow} P \gg P}\,\textbf{init}$$

Then

$\Xi^n(\Omega_L \backslash \Omega_L, \Omega'_{LI} \backslash \Omega'_{LO})$ and $\Xi^n(\Omega_R \backslash \Omega_R, \Omega'_{RI} \backslash \Omega'_{RO})$ assumptions

$\delta(\Delta'_O)$ and $\epsilon(\Delta \backslash \Delta, \Delta'_I \backslash \Delta'_O)$ assumptions

$\Delta'_I = \Delta'_O$ Lemma 40

$\Omega'_{LI} = \Omega'_{LO}$ and $\Omega'_{RI} = \Omega'_{RO}$ Lemma 39.1

$o(\Omega'_{LI})$ and $o(\Omega'_{RI})$ defn. of Ξ

$\Gamma; \Delta' \backslash \Delta'; (\Omega'_{LI} \backslash \Omega'_{LI}\ ;\ \Omega'_{RI} \backslash \Omega'_{RI}) \underset{F\ (\cdot\,;\,\cdot)}{\xrightarrow{n}} P \gg P$ **init**

case:

$$\frac{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{RO}) \underset{v\ (\tau_L\,;\,\tau_R)}{\longrightarrow} D \gg P}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_{LI} D \Omega_{RI} \backslash \Omega_{LO} \Box \Omega_{RO} \underset{v\ \tau_L 0 \tau_R}{\longrightarrow} P}\ \textbf{choice}_\Omega$$

Then

$\Xi^n(\Omega_{LI} D \Omega_{RI} \backslash \Omega_{LO} \Box \Omega_{RO}, \Omega'_I \backslash \Omega'_O)$ assumptions

$\delta(\Delta'_O)$ and $\epsilon(\Delta_I \backslash \Delta_O, \Delta'_I \Delta'_O)$ assumptions

$\Omega'_I = \Omega'_{LI} D^m \Omega'_{RI}$ and $\Omega'_O = \Omega'_{LO} \Box \Omega'_{RO}$ and
$\Xi(\Omega_{LI} \backslash \Omega_{LO}, \Omega'_{LI} \backslash \Omega'_{LO})$ and $\Xi(\Omega_{RI} \backslash \Omega_{RO}, \Omega'_{RI} \backslash \Omega'_{RO})$ Lemma 39.3

$\Gamma; \Delta'_I \backslash \Delta'_O; (\Omega'_{LI} \backslash \Omega'_{LO}\ ;\ \Omega'_{RI} \backslash \Omega'_{RO}) \underset{v\ (\tau_L\,;\,\tau_R)}{\xrightarrow{n}} D \gg P$ ind. hyp.

$\Gamma; \Delta'_I \backslash \Delta'_O; \Omega'_{LI} D^m \Omega'_{RI} \backslash \Omega'_{LO} \Box \Omega'_{RO} \underset{v\ \tau_L 0 \tau_R}{\xrightarrow{n}} P$ **choice**$_\Omega$

case:

$$\frac{\Gamma; \Delta_I \backslash \Delta_M; (\Omega_{LI} \backslash \Omega_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{GI} \Omega_{RO}) \underset{v_1\ (\tau_L\,;\,1\tau_R)}{\longrightarrow} D \gg P \qquad \Gamma; \Delta_M \backslash \Delta_O; \Omega_{GI} \backslash \Omega_{GO} \underset{v_2\ \tau}{\longrightarrow} G}{\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{GO} \Omega_{RO}) \underset{(v_1 \vee v_2)\ (\tau_L\,;\,\tau+1\tau_R)}{\longrightarrow} G \twoheadrightarrow D \gg P}\ {\twoheadrightarrow}_L$$

$(\Box \notin \Omega_{GI}$ and $(\Omega_{RO} = \Box\Omega$ or $\cdot))$

Then

$\Xi^n(\Omega_{LI} \backslash \Omega_{LO}, \Omega'_{LI} \backslash \Omega'_{LO})$ and $\Xi^n(\Omega_{RI} \backslash \Omega_{GO} \Omega_{RO}, \Omega'_{RI} \backslash \Omega'_R)$ assumptions

$\delta(\Delta'_O)$ and $\epsilon(\Delta_I \backslash \Delta_O, \Delta'_I \backslash \Delta'_O)$ assumptions

There exists Δ'_M such that
$\epsilon(\Delta_I \backslash \Delta_M, \Delta'_I \backslash \Delta'_M)$ and $\epsilon(\Delta_M \backslash \Delta_O, \Delta'_M \backslash \Delta'_I)$ Lemma 40

$\delta(\Delta'_M \triangleleft)$ defn. of δ.

Suppose $\Omega_{RO} = \Box \Omega_O$ for some Ω_O (case for $\Omega_{RO} = \cdot$ is similar)

$\Omega_{RI} = \Omega_{GI} D \Omega_I$ where $\Omega_{GI} \sqsupseteq \Omega_{GO}$ and $\Omega_I \sqsupseteq \Omega_O$ $\Omega_{RI} \sqsupseteq \Omega_{GO} \Omega_{RO}$

Note that $\Xi^n(\Omega_{GI}D\Omega_I\backslash\Omega_{GO}\Box\Omega_O, \Omega'_{RI}\backslash\Omega'_R)$

$\Omega'_{RI} = \Omega'_{GI}D^m\Omega'_I$ and $\Omega'_R = \Omega'_{GO}\Box\Omega'_O$ and

$\Xi^n(\Omega_{GI}\backslash\Omega_{GO}, \Omega'_{GI}\backslash\Omega'_{GO})$ and $\Xi^n(\Omega_I\backslash\Omega_O, \Omega'_I\backslash\Omega'_O)$ Lemma 39.3

$\Theta^n(\Omega_{GI}, \Omega'_{GI})$ Lemma 39.2

$\Xi^n(\Omega_{GI}\backslash\Omega_{GI}, \overset{n}{\triangleright}\Omega'_{GI}\backslash\overset{n}{\triangleright}\Omega'_{GI})$ defn. of Ξ

$\Xi^n(\Omega_{GI}D\Omega_I\backslash\Omega_{GI}\Box\Omega_O, \overset{n}{\triangleright}\Omega'_{GI}D^m\Omega'_I\backslash\overset{n}{\triangleright}\Omega'_{GI}\Box\Omega'_O)$ Lemma 39.4

$\Gamma; \Delta'_I\lhd\backslash\Delta'_M\lhd; (\Omega'_{LI}\backslash\Omega'_{LO}\ ;\ \overset{n}{\triangleright}\Omega'_{GI}D^m\Omega'_I\backslash\overset{n}{\triangleright}\Omega'_{GI}\Box\Omega'_O)\ \xrightarrow[v\ (\tau_L\ ;\ \tau_R)]{n}\ D \gg P$

ind. hyp.

$\Xi^{(n+1)}(\Omega_{GI}\backslash\Omega_{GO}, \Omega'_{GI}\overset{n}{\triangleleft}\backslash\Omega'_{GO}\overset{n}{\triangleleft})$ Lemmas 39.5 and 39.6

$\Gamma; \Delta'_M\backslash\Delta'_O; \Omega'_{GI}\overset{n}{\triangleleft}\backslash\Omega'_{GO}\overset{n}{\triangleleft}\ \xrightarrow[v\ \tau]{(n+1)}\ G$ ind. hyp.

$\Gamma; \Delta'_I\backslash\Delta'_O; (\Omega'_{LI}\backslash\Omega'_{LO}\ ;\ \Omega'_{GI}D^m\Omega'_I\backslash\Omega'_{GO}\Box\Omega'_O)\ \xrightarrow[v\ (\tau_L\ ;\ \tau+\tau_R)]{n}\ D \gg P$ by rule $\twoheadrightarrow_L$

□

11.5 Strictness Properties

We show that the system will never fail at the $\multimap_R$ rule.

Lemma 42 (Linear Strictness)

1. $\Gamma; \Delta_I\backslash\Delta_O; \Omega_I\backslash\Omega_O\ \xrightarrow[F\ \tau]{n}\ G$ *implies* $\delta(\Delta_O)$.
2. $\Gamma; \Delta_I\backslash\Delta_O; (\Omega_{LI}\backslash\Omega_{LO}\ ;\ \Omega_{RI}\backslash\Omega_{RO})\ \xrightarrow[F\ (\tau_L\ ;\ \tau_R)]{n}\ G \gg P$ *implies* $\delta(\Delta_O)$.

Proof: Induction on given derivation using Lemma 36. □

Thus we may collapse the two $\multimap_R$ rules into one rule:

$$\frac{\Gamma; \Delta_I D\backslash\Delta_O X; \Omega_I\backslash\Omega_O\ \xrightarrow[v\ \tau]{n}\ G}{\Gamma; \Delta_I\backslash\Delta_O; \Omega_I\backslash\Omega_O\ \xrightarrow[v\ \tau]{n}\ D \multimap G}\multimap_R$$

where X is either $\Box$ or D.

Additionally, the strict system will never fail at the $\twoheadrightarrow_R$ or $\rightarrowtail_R$ rules. To show this, we introduce the following notation:

$$
\begin{array}{rcl}
\rho(\Omega) & \equiv & \max\{n \mid \overset{n}{\triangleleft} \in \Omega \text{ or } \overset{n}{\triangleright} \in \Omega\} \\[2ex]
\mu(\Omega_I, \Omega_O) & \equiv & \Omega_I \sqsupseteq \Omega_O \quad \text{and} \quad \overline{\Omega_O} \leq \rho(\Omega_I) \quad \text{and} \quad \forall \Omega_{IL}, \Omega_{IR}, D^m, \Omega_{OL}, \Omega_{OR}. \\
& & \qquad (\Omega_{IL} D^m \Omega_{IR} = \Omega_I \text{ and } \Omega_{OL} \Box \Omega_{OR} = \Omega_O \text{ and } \|\Omega_{IL}\| = \|\Omega_{OL}\|) \\
& & \qquad\qquad \text{implies} \quad \overline{\Omega_{OL}} \leq \rho(\Omega_{IL}) \text{ and } \overline{\Omega_{OR}} \leq \rho(\Omega_{IR})
\end{array}
$$

Lemma 43 (Ordered Strictness)

1. $\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \underset{v \quad \tau}{\xrightarrow{n}} G$ *implies* $\mu(\Omega_I, \Omega_O)$

2. $\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{RO}) \underset{v \quad (\tau_L\ ;\ \tau_R)}{\xrightarrow{n}} D \gg P$ *implies* $\mu(\Omega_{LI}, \Omega_{LO})$ *and* $\mu(\Omega_{RI}, \Omega_{RO})$

Proof: Structural induction on given derivation. We show a representative case.

case:

$$
\frac{\Gamma; \Delta_I \triangleleft \backslash \Delta_M \triangleleft; \Omega_I \overset{n}{\triangleleft} \backslash \Omega_L \Omega_2 \overset{n}{\triangleleft} \underset{v_1 \quad \tau_L}{\xrightarrow{(n+1)}} G_1 \qquad \Gamma; \Delta_M \backslash \Delta_O; \Omega_2 \backslash \Omega_R \underset{v_2 \quad \tau_R}{\xrightarrow{(n+1)}} G_2}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_L \Omega_R \underset{(v_1 \vee v_2) \quad (\tau_L + \tau_R)}{\xrightarrow{n}} G_1 \bullet G_2} \bullet_{R0}
$$

$(\Box \notin \Omega_2$ and $(\Omega_L = \Omega\Box$ or $\cdot)$ and $\tau_L \neq \tau 1)$

Then

$$
\begin{array}{lr}
\mu(\Omega_I \overset{n}{\triangleleft}, \Omega_L \Omega_2 \overset{n}{\triangleleft}) \text{ and } \mu(\Omega_2, \Omega_R) & \text{ind. hyp.} \\
\Omega_I = \Omega_{IL} \Omega_2 \text{ and } \Omega_{IL} \sqsupseteq \Omega_L & \Omega_I \overset{n}{\triangleleft} \sqsupseteq \Omega_L \Omega_2 \overset{n}{\triangleleft} \\
\Omega_I \sqsupseteq \Omega_L \Omega_R & \\
\rho(\Omega_I) \geq \rho(\Omega_{IL}) \text{ and } \rho(\Omega_I) \geq \rho(\Omega_2) & \Omega_I = \Omega_{IL} \Omega_2 \\
\overline{\Omega_L} \leq \rho(\Omega_{IL}) \leq \rho(\Omega_I) & \Omega_L = \Omega\Box \text{ or } \cdot \text{ and } \mu(\Omega_I \overset{n}{\triangleleft}, \Omega_L \Omega_2 \overset{n}{\triangleleft}) \\
\overline{\Omega_R} \leq \rho(\Omega_2) \leq \rho(\Omega_I) & \mu(\Omega_2, \Omega_R) \\
\overline{\Omega_L \Omega_R} \leq \rho(\Omega_I) & \\
\text{Let } \Omega_{LI} D^m \Delta_{RI} = \Delta_I \text{ and } \Omega_{LO} \Box \Omega_{RO} = \Omega_L \Omega_R \text{ where } \|\Omega_{LI}\| = \|\Omega_{LO}\| & \\
D^m \in \Omega_{IL} \text{ or } D^m \in \Omega_2 & \\
\text{In either case } \overline{\Omega_{xO}} \leq \rho(\Omega_{xI}) & \text{ind. hyp.}
\end{array}
$$

□

We will call a sequent $\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \xrightarrow[v\ \tau]{n} G$ valid if $\overline{\Omega_I} \leq n$ and $\rho(\Omega_I) < n$. We will call a sequent $\Gamma; \Delta_I \backslash \Delta_O; (\Omega_{LI} \backslash \Omega_{LO}\ ;\ \Omega_{RI} \backslash \Omega_{RO}) \xrightarrow[v\ (\tau_L\ ;\ \tau_R)]{n} D \gg P$ valid if $\overline{\Omega_{LI}\Omega_{RI}} \leq n$ and $\rho(\Omega_{LI}\Omega_{RI}) \leq n$. Note that the derivation rules preserve this validity. It is a corollary of Lemma 43 that bottom-up, left-to-right derivations of valid sequents will never fail at the $\twoheadrightarrow_R$ or $\rightarrowtail_R$ rules.

11.6 Implementation Issues

After choosing a formula D, computation of its residual formula, $G'; D \gg P \setminus G$, is parametric in P. Thus, we can compile program clauses once, abstracting over P, and then use them in their compiled form by giving them the P we are trying to solve.

The strict system has the nice property that the ordered context checks in the **init**, $\doteq_R$, 1_R, and $\rightarrow_L$ rules may be done in constant time. One need only check each end of the ordered context to determine if $\overline{\Omega} \leq m$. One can easily see that this is indeed the case by noticing that the tags only increase from conclusion to premise(s) and that no inference rules change the tags on any ordered hypotheses or frame pointers. Then, since everything is added to the right or left side of the ordered context, the highest tag in a context must occur at either end of the context.

The linear context checks, *i.e.*, $\delta(\Delta)$, are possibly linear in the length of Δ. However, we can further optimize the linear context management as follows. Since we know the strictness property holds, we may actually remove linear hypotheses from the linear context once they are consumed. We may then change the definition of δ as follows:

$$\frac{}{\delta(\cdot)} \qquad \frac{}{\delta(\Delta \triangleleft)}$$

We then change the **choice**$_\Delta$, $\top_R$ and $\multimap_R$ rules as follow:

$$\frac{\mathbf{1}; D \gg P \setminus G \qquad \Gamma; \Delta_L\Delta_R \backslash \Delta_O; \Omega_I \backslash \Omega_O \xrightarrow[v\ \tau]{n} G}{\Gamma; \Delta_L D \Delta_R \backslash \Delta_O; \Omega_I \backslash \Omega_O \xrightarrow[v\ \tau]{n} P}\,\mathbf{choice}_\Omega$$

$$\frac{\triangleleft \notin \Delta'}{\Gamma; \Delta \triangleleft \Delta' \backslash \Delta; \Omega_I \backslash \Omega_O \xrightarrow[T\ 1]{n} \top}\,\top_{R\triangleleft} \qquad \frac{}{\Gamma; \cdot \backslash \cdot; \Omega_I \backslash \Omega_O \xrightarrow[T\ 1]{n} \top}\,\top_{R\cdot}$$

$$\frac{\Gamma; \Delta_I D \backslash \Delta_O; \Omega_I \backslash \Omega_O \ \xrightarrow[\upsilon\ \ \tau]{n}\ G}{\Gamma; \Delta_I \backslash \Delta_O; \Omega_I \backslash \Omega_O \ \xrightarrow[\upsilon\ \ \tau]{n}\ D \multimap G} \multimap_R$$

While failing much sooner than previous versions, the system is still not quite equivalent to RM3. In the $\&_R$ rule, no attempt is made to restrict the second premise to only using formulas consumed by the first premise. Thus a strict derivation can fail in the $\&_R$ rule. We could of course adapt the scheme used in the frame system [33], which is linear in the size of the linear context, to prevent failures due to linear hypotheses. However the extension of this scheme to the ordered case is not very satisfying– it seems to prohibit constant time ordered context strictness checks at the proof leaves. Thus, it is not clear if "fixing" this is worth the trouble since it would seem to require scanning through the ordered context at every terminal rule.

There is a prototype implementation of Olli using the strict derivation system, with the just-mentioned modifications. It is written in the Teyjus [38] implementation of λProlog and is available at:

`http://www.cs.cmu.edu/~jpolakow.`

Part III

Ordered Logical Framework

Chapter 12

Ordered Types

In this chapter we add proof terms to ordered linear logic. This essentially turns the logic into a type system for an ordered lambda calculus following the Curry-Howard isomorphism. Furthermore, as hinted at in Chapter 2, the local reductions and expansions become β-reductions and η-expansions for the ordered lambda calculus terms.

After re-introducing the logic as a type theory, we will show that there is a canonical fragment of the type theory, *i.e.*, a fragment for which *canonical* (or long $\beta\eta$-normal) forms exist, which corresponds to the uniform fragment of the logic examined in Chapter 6. The existence of canonical forms is critical in logical framework applications of our calculus, since it is the canonical forms which are in bijective correspondence with the objects to be represented. This property is inherited both from the logical framework LF [23] and its linear refinement LLF [11].

In Chapter 13 we will show that we can add dependent types to the theory to produce a LF style logical framework. We will then proceed to show that typechecking remains decidable and that canonical forms still exist[1]. Finally, in Chapter 16 we will show how such an ordered logical framework can be used to analyze some syntactic properties of the CPS transform.

[1] We will not show actual canonical forms, rather we will show that a slightly weaker notion exists which is still suitable for logical framework representations– see Section 15.3.

12.1 Proof Terms

In this section we present the basic natural deduction system for ordered linear logic (Chapter 2.10) annotated with proof terms. The proof terms will form an ordered lambda calculus.

We start with the types we shall use:

Types	A	$::=$	a	atomic types
		$\mid$	$A_1 \rightarrow A_2$	unrestricted implication
		$\mid$	$A_1 \multimap A_2$	linear implication
		$\mid$	$A_1 \twoheadrightarrow A_2$	ordered right implication
		$\mid$	$A_1 \rightarrowtail A_2$	ordered left implication
		$\mid$	$A_1 \,\&\, A_2$	additive conjunction
		$\mid$	$\top$	additive truth
		$\mid$	$A_1 \bullet A_2$	multiplicative conjunction
		$\mid$	$\mathbf{1}$	multiplicative truth
		$\mid$	$A_1 \oplus A_2$	additive disjunction
		$\mid$	$\mathbf{0}$	additive falsehood
		$\mid$	$!A$	unrestricted modality
		$\mid$	¡A	linear modality

We do not consider the quantifiers here. Instead we will consider a generalization of the universal quantifier (dependent types) in Chapter 13. We will not treat the existential quantifier at all, since it is not necessary for the applications of ordered lambda terms considered in this thesis. Furthermore, we only include one multiplicative pair type ($\bullet$) since the other one ($\circ$) is trivially definable, $A_1 \circ A_2 \;\equiv\; A_2 \bullet A_1$.

We will use the following proof terms:

$$
\begin{array}{llll}
\textit{Terms} & M & ::= & x \mid y \mid z & \text{variables} \\
& & \mid & \lambda x{:}A.\ M \mid M_1\, M_2 & \text{unrestricted functions } (\rightarrow) \\
& & \mid & \hat{\lambda} y{:}A.\ M \mid M_1 \,\hat{}\, M_2 & \text{linear functions } (\multimap) \\
& & \mid & \overset{>}{\lambda} z{:}A.\ M \mid M_1 {}^{>} M_2 & \text{right ordered functions } (\twoheadrightarrow) \\
& & \mid & \overset{<}{\lambda} z{:}A.\ M \mid M_1 {}^{<} M_2 & \text{left ordered functions } (\rightarrowtail) \\
& & \mid & \langle M_1\,,\, M_2\rangle \mid \textbf{fst}\, M \mid \textbf{snd}\, M & \text{additive pairs } (\&) \\
& & \mid & \langle\rangle & \text{additive unit } (\top) \\
& & \mid & [M_1\,,\, M_2] \mid \textbf{let}\,[z\,,\, z'] = M\ \textbf{in}\ N & \text{multiplicative pairs } (\bullet) \\
& & \mid & \star \mid \textbf{let}\,\star = M\ \textbf{in}\ N & \text{multiplicative unit } (\mathbf{1}) \\
& & \mid & \textbf{inl}^B\, M \mid \textbf{inr}^A\, M & \\
& & \mid & \textbf{case}\, M\ \textbf{of}\ \textbf{inl}\, z \Rightarrow N \| \textbf{inr}\, z' \Rightarrow N' & \text{additive disjunction } (\oplus) \\
& & \mid & \textbf{abort}^A\, M & \text{additive falsehood } (\mathbf{0}) \\
& & \mid & !M \mid \textbf{let}\, !x = M\ \textbf{in}\ N & \text{unrestricted modality } (!) \\
& & \mid & \text{¡}M \mid \textbf{let}\, \text{¡}y = M\ \textbf{in}\ N & \text{linear modality } (\text{¡})
\end{array}
$$

We use the following judgement to type our terms:

$$\Gamma; \Delta; \Omega \vdash M : A$$

where Γ, Δ, and Ω have the same form as in the natural deduction system in Chapter 2[2].

We can now state all of the typing rules. These are the same rules given in Chapter 2.10, annotated with proof terms. We also restate the local reductions and expansions for each term as β-reductions and η-expansions.

Variable Rules.

$$\frac{}{\Gamma_1(x{:}A)\Gamma_2; \cdot; \cdot \vdash x : A}\,\textbf{uvar}$$

$$\frac{}{\Gamma; y{:}A; \cdot \vdash y : A}\,\textbf{lvar}$$

$$\frac{}{\Gamma; \cdot; z{:}A \vdash z : A}\,\textbf{ovar}$$

[2] However, we now interpret the labels on hypotheses as variables.

Unrestricted Functions $A \rightarrow B$**.**

$$\frac{\Gamma(x{:}A); \Delta; \Omega \vdash M : B}{\Gamma; \Delta; \Omega \vdash \lambda x{:}A.\ M : A \rightarrow B} \rightarrow I$$

$$\frac{\Gamma; \Delta; \Omega \vdash M : A \rightarrow B \qquad \Gamma; \cdot; \cdot \vdash N : A}{\Gamma; \Delta; \Omega \vdash M\, N : B} \rightarrow E$$

We have the following reduction rule:

$$(\lambda x{:}A.\ M)\, N \quad \Longrightarrow_\beta \quad [N/x]M$$

We have the following expansion for terms M of type $A \rightarrow B$:

$$M \quad \Longrightarrow_\eta \quad \lambda x{:}A.\ M\, x$$

where x is not free in M.

Linear Functions $A \multimap B$**.**

$$\frac{\Gamma; \Delta(y{:}A); \Omega \vdash M : B}{\Gamma; \Delta; \Omega \vdash \hat{\lambda} y{:}A.\ M : A \multimap B} \multimap I$$

$$\frac{\Gamma; \Delta_1; \Omega \vdash M : A \multimap B \qquad \Gamma; \Delta_2; \cdot \vdash N : A}{\Gamma; \Delta_1 \bowtie \Delta_2; \Omega \vdash M\,\hat{}\,N : B} \multimap E$$

We have the following reduction rule:

$$(\hat{\lambda} y{:}A.\ M)\,\hat{}\,N \quad \Longrightarrow_\beta \quad [N/y]M$$

We have the following expansion for terms M of type $A \multimap B$:

$$M \quad \Longrightarrow_\eta \quad \hat{\lambda} y{:}A.\ M\,\hat{}\,y$$

where y is not free in M.

Right Ordered Functions $A \twoheadrightarrow B$.

$$\frac{\Gamma; \Delta; \Omega(z{:}A) \vdash M : B}{\Gamma; \Delta; \Omega \vdash \overset{>}{\lambda} z{:}A.\ M : A \twoheadrightarrow B} \twoheadrightarrow I$$

$$\frac{\Gamma; \Delta_1; \Omega_1 \vdash M : A \twoheadrightarrow B \qquad \Gamma; \Delta_2; \Omega_2 \vdash N : A}{\Gamma; \Delta_1 \bowtie \Delta_2; \Omega_1\Omega_2 \vdash M^{>} N : B} \twoheadrightarrow E$$

We have the following reduction rule:

$$(\overset{>}{\lambda} z{:}A.\ M)^{>} N \quad \Longrightarrow_{\beta} \quad [N/z]M$$

We have the following expansion for terms M of type $A \twoheadrightarrow B$:

$$M \quad \Longrightarrow_{\eta} \quad \overset{>}{\lambda} z{:}A.\ M^{>} z$$

where z is not free in M.

Left Ordered Functions $A \rightarrowtail B$.

$$\frac{\Gamma; \Delta; (z{:}A)\Omega \vdash M : B}{\Gamma; \Delta; \Omega \vdash \overset{<}{\lambda} z{:}A.\ M : A \rightarrowtail B} \rightarrowtail I$$

$$\frac{\Gamma; \Delta_2; \Omega_2 \vdash M : A \rightarrowtail B \qquad \Gamma; \Delta_1; \Omega_1 \vdash N : A}{\Gamma; \Delta_1 \bowtie \Delta_2; \Omega_1\Omega_2 \vdash M^{<} N : B} \rightarrowtail E$$

We have the following reduction rule:

$$(\overset{<}{\lambda} z{:}A.\ M)^{<} N \quad \Longrightarrow_{\beta} \quad [N/z]M$$

We have the following expansion for terms M of type $A \rightarrowtail B$:

$$M \quad \Longrightarrow_{\eta} \quad \overset{<}{\lambda} z{:}A.\ M^{<} z$$

where z is not free in M.

Multiplicative Conjunction $A{\bullet}B$.

$$\frac{\Gamma; \Delta_1; \Omega_1 \vdash M{:}A \qquad \Gamma; \Delta_2; \Omega_2 \vdash N{:}B}{\Gamma; \Delta_1 \bowtie \Delta_2; \Omega_1\Omega_2 \vdash [M\,,\,N] : A \bullet B}\bullet I$$

$$\frac{\Gamma; \Delta_2; \Omega_2 \vdash M : A \bullet B \qquad \Gamma; \Delta_1; \Omega_1(z{:}A)(z'{:}B)\Omega_3 \vdash N : C}{\Gamma; \Delta_1\Delta_2; \Omega_1\Omega_2\Omega_3 \vdash \mathbf{let}\,[z\,,\,z'] = M\ \mathbf{in}\ N : C}\bullet E$$

We have the following reduction rule:

$$\mathbf{let}\,[z\,,\,z'] = [M\,,\,M']\ \mathbf{in}\ N \Longrightarrow_\beta N[M/z, M'/z']$$

We have the following expansion for terms M of type $A \bullet B$:

$$M \quad \Longrightarrow_\eta \quad \mathbf{let}\,[z\,,\,z'] = M\ \mathbf{in}\ [z\,,\,z']$$

We will not bother with assigning a term to the other ordered conjunction, $\circ$, since it simply reverses z and z' in the term displayed above.

Multiplicative Unit $\mathbf{1}$.

$$\frac{}{\Gamma; \cdot; \cdot \vdash \star : \mathbf{1}}\mathbf{1}I$$

$$\frac{\Gamma; \Delta_2; \Omega_2 \vdash M : \mathbf{1} \qquad \Gamma; \Delta_1; \Omega_1\Omega_3 \vdash N : C}{\Gamma; \Delta_1 \bowtie \Delta_2; \Omega_1\Omega_2\Omega_3 \vdash \mathbf{let}\,\star = M\ \mathbf{in}\ N : C}\mathbf{1}E$$

We have the following reduction rule:

$$\mathbf{let}\,\star = \star\ \mathbf{in}\ N \Longrightarrow_\beta N$$

We have the following expansion for terms M of type $\mathbf{1}$:

$$M \quad \Longrightarrow_\eta \quad \mathbf{let}\,\star = M\ \mathbf{in}\ \star$$

Additive Conjunction $A\&B$**.**

$$\frac{\Gamma;\Delta;\Omega \vdash M : A \qquad \Gamma;\Delta;\Omega \vdash N : B}{\Gamma;\Delta;\Omega \vdash \langle M\,,\,N\rangle : A \,\&\, B}\,\&I$$

$$\frac{\Gamma;\Delta;\Omega \vdash M : A \,\&\, B}{\Gamma;\Delta;\Omega \vdash \mathbf{fst}\,M : A}\,\&E_1 \qquad \frac{\Gamma;\Delta;\Omega \vdash M : A \,\&\, B}{\Gamma;\Delta;\Omega \vdash \mathbf{snd}\,M : B}\,\&E_2$$

We have the following reduction rules:

$$\begin{array}{rcl} \mathbf{fst}\,\langle M\,,\,N\rangle & \Longrightarrow_\beta & M \\ \mathbf{snd}\,\langle M\,,\,N\rangle & \Longrightarrow_\beta & N \end{array}$$

We have the following expansion for terms M of type $A \,\&\, B$:

$$M \quad \Longrightarrow_\eta \quad \langle \mathbf{fst}\,M\,,\,\mathbf{snd}\,M\rangle$$

Additive Unit $\top$**.**

$$\frac{}{\Gamma;\Delta;\Omega \vdash \langle\rangle : \top}\,\top I$$

Since there is no elimination rule, there are no reductions for the additive unit. However we do the following expansion for terms M of type $\top$:

$$M \quad \Longrightarrow_\eta \quad \langle\rangle$$

Additive Disjunction $\oplus$**.**

$$\frac{\Gamma;\Delta;\Omega \vdash M : A}{\Gamma;\Delta;\Omega \vdash \mathbf{inl}^B\,M : A \oplus B}\,\oplus I_1 \qquad \frac{\Gamma;\Delta;\Omega \vdash M : B}{\Gamma;\Delta;\Omega \vdash \mathbf{inr}^A\,M : A \oplus B}\,\oplus I_2$$

$$\frac{\Gamma;\Delta_2;\Omega_2 \vdash M : A \oplus B \qquad \Gamma;\Delta_1;\Omega_1(z{:}A)\Omega_3 \vdash N : C \quad \Gamma;\Delta_1;\Omega_1(z'{:}B)\Omega_3 \vdash N' : C}{\Gamma;\Delta_1 \bowtie \Delta_2;\Omega_1\Omega_2\Omega_3 \vdash \mathbf{case}\,M\,\mathbf{of}\,\mathbf{inl}\,z \Rightarrow N \| \mathbf{inr}\,z' \Rightarrow N' : C}\,\oplus E$$

We have the following reduction rules:

$$\begin{array}{rcl} \mathbf{case}\,\mathbf{inl}^B\,M\,\mathbf{of}\,\mathbf{inl}\,z \Rightarrow N \| \mathbf{inr}\,z' \Rightarrow N' & \Longrightarrow_\beta & [M/z]N \\ \mathbf{case}\,\mathbf{inr}^A\,M'\,\mathbf{of}\,\mathbf{inl}\,z \Rightarrow N \| \mathbf{inr}\,z' \Rightarrow N' & \Longrightarrow_\beta & [M'/z']N' \end{array}$$

We have the following expansion for terms M of type $A \oplus B$:

$$M \quad \Longrightarrow_\eta \quad \mathbf{case}\,M\,\mathbf{of}\,\mathbf{inl}\,z \Rightarrow \mathbf{inl}\,z \| \mathbf{inr}\,z' \Rightarrow \mathbf{inr}\,z'$$

Additive Falsehood $\mathbf{0}$.

$$\frac{\Gamma;\Delta_2;\Omega_2 \vdash M : \mathbf{0}}{\Gamma;\Delta_1 \bowtie \Delta_2;\Omega_1\Omega_2\Omega_3 \vdash \mathbf{abort}^C\, M : C}\,\mathbf{0}E$$

Since there is no introduction rule for $\mathbf{0}$, there are no new reductions. We do have the following expansion for terms M of type $\mathbf{0}$:

$$M \quad \Longrightarrow_\eta \quad \mathbf{abort}^{\mathbf{0}}\, M$$

Unrestricted Modality $!A$.

$$\frac{\Gamma;\cdot;\cdot \vdash M : A}{\Gamma;\cdot;\cdot \vdash\, !M : !A}\,!I$$

$$\frac{\Gamma;\Delta_2;\Omega_2 \vdash M : !A \qquad \Gamma(x{:}A);\Delta_1;\Omega_1\Omega_3 \vdash N : C}{\Gamma;\Delta_1 \bowtie \Delta_2;\Omega_1\Omega_2\Omega_3 \vdash \mathbf{let}\, !x = M\ \mathbf{in}\ N : C}\,!E$$

We have the following reduction rule:

$$\mathbf{let}\, !x = !M\ \mathbf{in}\ N \quad \Longrightarrow_\beta \quad [M/x]N$$

We have the following expansion for terms M of type $!A$:

$$M \quad \Longrightarrow_\eta \quad \mathbf{let}\, !x = M\ \mathbf{in}\ !x$$

Linear Modality $\text{¡}A$.

$$\frac{\Gamma;\Delta;\cdot \vdash M : A}{\Gamma;\Delta;\cdot \vdash \text{¡}M : \text{¡}A}\,\text{¡}I$$

$$\frac{\Gamma;\Delta_2;\Omega_2 \vdash M : \text{¡}A \qquad \Gamma;\Delta_1(y{:}A);\Omega_1\Omega_3 \vdash N : C}{\Gamma;\Delta_1 \bowtie \Delta_2;\Omega_1\Omega_2\Omega_3 \vdash \mathbf{let}\, \text{¡}y = M\ \mathbf{in}\ N : C}\,\text{¡}E$$

We have the following reduction rule:

$$\mathbf{let}\, \text{¡}y = \text{¡}M\ \mathbf{in}\ N \quad \Longrightarrow_\beta \quad [M/y]N$$

We have the following expansion for terms M of type $\text{¡}A$:

$$M \quad \Longrightarrow_\eta \quad \mathbf{let}\, \text{¡}y = M\ \mathbf{in}\ \text{¡}y$$

In order to prove subject reduction we proceed to establish the expected structural properties for contexts and substitution lemmas.

Lemma 44 (Structural Properties)

1. $\Gamma_1(x{:}A)(x'{:}A')\Gamma_2; \Delta; \Omega \vdash M : B$ *implies* $\Gamma_1(x'{:}A')(x{:}A)\Gamma_2; \Delta; \Omega \vdash M : B$.
2. $\Gamma_1\Gamma_2; \Delta; \Omega \vdash M : B$ *implies* $\Gamma_1(x{:}A)\Gamma_2; \Delta; \Omega \vdash M : B$.
3. $\Gamma_1(x{:}A)(x'{:}A)\Gamma_2; \Delta; \Omega \vdash M : B$ *implies* $\Gamma_1(x{:}A)\Gamma_2; \Delta; \Omega \vdash [x/x']M : B$.
4. $\Gamma; \Delta_1(y{:}A)(y'{:}A')\Delta_2; \Omega \vdash M : B$ *implies* $\Gamma; \Delta_1(y'{:}A')(y{:}A)\Delta_2; \Omega \vdash M : B$.

Proof: By induction on the structure of the given derivations. □

Lemma 45 (Substitution Properties)

1. $\Gamma_1(x{:}A)\Gamma_2; \Delta; \Omega \vdash M : B$ *and* $\Gamma_1; \cdot; \cdot \vdash N : A$
implies $\Gamma_1\Gamma_2; \Delta; \Omega \vdash [N/x]M : B$.
2. $\Gamma; \Delta_1(y{:}A)\Delta_2; \Omega \vdash M : B$ *and* $\Gamma; \Delta'; \cdot \vdash N : A$
implies $\Gamma; (\Delta_1 \bowtie \Delta')\Delta_2; \Omega \vdash [N/y]M : B$.
3. $\Gamma; \Delta; \Omega_1(z{:}A)\Omega_2 \vdash M : B$ *and* $\Gamma; \Delta'; \Omega' \vdash N : A$
implies $\Gamma; \Delta \bowtie \Delta'; \Omega_1\Omega'\Omega_2 \vdash [N/z]M : B$.

Proof: By induction over the structure of the given typing derivation for M in each case, using Lemma 44. □

Subject reduction now follows immediately.

Theorem 46 (Subject Reduction)

$\Gamma; \Delta; \Omega \vdash M : A$ *and* $M \Longrightarrow_\beta M'$ *implies* $\Gamma; \Delta; \Omega \vdash M' : A$.

Proof: For each reduction, we apply inversion to the given typing derivation and then use the substitution lemma 45 to obtain the typing derivation for the conclusion. □

Subject expansion also holds.

Theorem 47 (Subject Expansion)

$\Gamma; \Delta; \Omega \vdash M : A$ *and* $M \Longrightarrow_\eta M'$ *implies* $\Gamma; \Delta; \Omega \vdash M' : A$.

Proof: By a direct derivation in each case[3], using weakening (Lemma 44.(2)) for unrestricted functions. □

Finally, we state one further property of ordered typing derivations which is crucial for our logical-relations applications in Chapter 16.

Theorem 48 (Demotion)

1. $\Gamma; \Delta_1(y{:}A)\Delta_2; \Omega \vdash M : B$ *implies* $\Gamma(x{:}A); \Delta_1\Delta_2; \Omega \vdash [x/y]M : B$.
2. $\Gamma; \Delta; \Omega_1(z{:}A)\Omega_2 \vdash M : B$ *implies* $\Gamma; \Delta(y{:}A); \Omega_1\Omega_2 \vdash [y/z]M : B$.

Proof: In both cases by induction on the structure of the given derivation. □

12.2 Canonical Forms

In this section, we show that canonical (or long $\beta\eta$-normal) forms exists for a fragment of the ordered lambda calculus. This fragment consists of the types which correspond to the uniform fragment of ordered linear logic. We give a proof by logical relations which foreshadows some of the techniques we will use in Chapters 13, 14, and 15 to prove decidability of type-checking in an ordered logical framework[4]. We further speculate that this proof could likely be adapted to a direct proof of normalization for ordered linear logic (without using the sequent calculus).

For the remainder of this chapter, we only consider types, A, within the canonical fragment:

Canonical Types	A	$::=$	a	atomic types
		$\mid$	$A_1 \to A_2$	unrestricted implication
		$\mid$	$A_1 \multimap A_2$	linear implication
		$\mid$	$A_1 \twoheadrightarrow A_2$	ordered right implication
		$\mid$	$A_1 \rightarrowtail A_2$	ordered left implication
		$\mid$	$A_1 \,\&\, A_2$	additive conjunction
		$\mid$	$\top$	additive truth

[3] In fact the expansions shown in Chapter 2.

[4] The type system of this chapter extended with dependent types.

In order to show the existence of canonical forms, we will give a canonicalization procedure, and then show that this procedure always succeeds on terms in the canonical fragment of the calculus (*i.e.*, terms whose types are in the uniform fragment of ordered linear logic). In order to simplify our argument, and to show a more realistic canonicalization procedure, we ignore linearity and ordering constraints when transforming a term into canonical form. This omission is sound as long as we know beforehand that the term being transformed is well-typed.

Our proof proceeds as follows. We first formalize the property that a term can be converted to canonical form via a deductive system which can easily be related to the usual notion of long $\beta\eta$-normal form. This deductive system can also be read as an algorithm for converting a term to canonical form.

We then prove that any well-typed term (within the canonical fragment) can indeed be converted to canonical form. Our proof will be an argument by Kripke logical relations (also called Tait's method) consisting of two parts: (1) If M is a well-typed term of type A then M is in the logical relation represented by A, and (2) if M is in the logical relation represented by A then there is some canonical term N convertible to M. Our reduction strategy is based on weak head reduction defined below.

$$\frac{}{(\lambda x{:}A.\ M)\, N \ \xrightarrow{\texttt{whr}}\ M[N/x]}\,\beta_{\rightarrow} \qquad \frac{M \ \xrightarrow{\texttt{whr}}\ M'}{M\, N \ \xrightarrow{\texttt{whr}}\ M'\, N}\,\mathrm{whr}_{\rightarrow}$$

$$\frac{}{(\hat{\lambda} y{:}A.\ M)\,\hat{}\,N \ \xrightarrow{\texttt{whr}}\ M[N/y]}\,\beta_{\multimap} \qquad \frac{M \ \xrightarrow{\texttt{whr}}\ M'}{M\,\hat{}\,N \ \xrightarrow{\texttt{whr}}\ M'\,\hat{}\,N}\,\mathrm{whr}_{\multimap}$$

$$\frac{}{(\overset{<}{\lambda} z{:}A.\ M)^{<}\, N \ \xrightarrow{\texttt{whr}}\ M[N/z]}\,\beta_{\rightarrowtail} \qquad \frac{M \ \xrightarrow{\texttt{whr}}\ M'}{M^{<}\, N \ \xrightarrow{\texttt{whr}}\ M'^{<}\, N}\,\mathrm{whr}_{\rightarrowtail}$$

$$\frac{}{(\overset{>}{\lambda} z{:}A.\ M)^{>}\, N \ \xrightarrow{\texttt{whr}}\ M[N/z]}\,\beta_{\twoheadrightarrow} \qquad \frac{M \ \xrightarrow{\texttt{whr}}\ M'}{M^{>}\, N \ \xrightarrow{\texttt{whr}}\ M'^{>}\, N}\,\mathrm{whr}_{\twoheadrightarrow}$$

$$\frac{}{\mathbf{fst}\,\langle M\,,\,N\rangle \xrightarrow{\mathtt{whr}} M}\beta_{\&1} \qquad \frac{M \xrightarrow{\mathtt{whr}} M'}{\langle M\,,\,N\rangle \xrightarrow{\mathtt{whr}} \langle M'\,,\,N\rangle}\mathrm{whr}_{\&1}$$

$$\frac{}{\mathbf{snd}\,\langle M\,,\,N\rangle \xrightarrow{\mathtt{whr}} N}\beta_{\&2} \qquad \frac{N \xrightarrow{\mathtt{whr}} N'}{\langle M\,,\,N\rangle \xrightarrow{\mathtt{whr}} \langle M\,,\,N'\rangle}\mathrm{whr}_{\&2}$$

Intuitively, canonical terms are atomic terms of atomic type, or λ-abstractions of canonical terms, or pairs of canonical terms. Atomic terms are variables, or **fst** applied to atomic terms, or **snd** applied to atomic terms, or applications of atomic terms to canonical terms. This is formalized in the judgments $\Psi \vdash M \Uparrow M' : A$, which denotes that M has canonical form M' at type A, and $\Psi \vdash M \downarrow M' : A$, which denotes that M has atomic form M' at type A. Ψ stands for a context composed of unrestricted, linear and ordered variables. Note that these inference rules are essentially the normal derivation rules of Chapter 3 without the linearity and ordering constraints on hypotheses.

Variables

$$\frac{}{\Psi_1(x{:}A)\Psi_2 \vdash x \downarrow x : A}\mathbf{uvar}$$

$$\frac{}{\Psi_1(y{:}A)\Psi_2 \vdash y \downarrow y : A}\mathbf{lvar}$$

$$\frac{}{\Psi_1(z{:}A)\Psi_2 \vdash z \downarrow z : A}\mathbf{ovar}$$

Atomic Types.

$$\frac{\Psi \vdash M \downarrow M' : a}{\Psi \vdash M \Uparrow M' : a}\mathbf{coercion}$$

$$\frac{M \xrightarrow{\mathtt{whr}} M' \qquad \Psi \vdash M' \Uparrow M'' : a}{\Psi \vdash M \Uparrow M'' : a}\mathbf{reduction}$$

Unrestricted Functions.

$$\frac{\Psi(x{:}A) \vdash M\,x \Uparrow M' : B}{\Psi \vdash M \Uparrow \lambda x{:}A.\ M' : A \to B}\,{\to}I$$

$$\frac{\Psi \vdash M \downarrow M' : A \to B \qquad \Psi \vdash N \Uparrow N' : A}{\Psi \vdash M\,N \downarrow M'\,N' : B}\,{\to}E$$

Linear Functions.

$$\frac{\Psi(y{:}A) \vdash M\,\hat{}\,y \Uparrow M' : B}{\Psi \vdash M \Uparrow \hat{\lambda} y{:}A.\ M' : A \multimap B}\,{\multimap}I$$

$$\frac{\Psi \vdash M \downarrow M' : A \multimap B \qquad \Psi \vdash N \Uparrow N' : A}{\Psi \vdash M\,\hat{}\,N \downarrow M'\,\hat{}\,N' : B}\,{\multimap}E$$

Right Ordered Functions.

$$\frac{\Psi(z{:}A) \vdash M\,^{>}z \Uparrow M' : B}{\Psi \vdash M \Uparrow \overset{>}{\lambda} z{:}A.\ M' : A \twoheadrightarrow B}\,{\twoheadrightarrow}I$$

$$\frac{\Psi \vdash M \downarrow M' : A \twoheadrightarrow B \qquad \Psi \vdash N \Uparrow N' : A}{\Psi \vdash M\,^{>}N \downarrow M'\,^{>}N' : B}\,{\twoheadrightarrow}E$$

Left Ordered Functions.

$$\frac{\Psi(z{:}A) \vdash M^{<}\,z \Uparrow M' : B}{\Psi \vdash M \Uparrow \overset{<}{\lambda} z{:}A.\ M' : A \rightarrowtail B}\,{\rightarrowtail}I$$

$$\frac{\Psi \vdash M \downarrow M' : A \rightarrowtail B \qquad \Psi \vdash N \Uparrow N' : A}{\Psi \vdash M^{<}\,N \downarrow M'^{<}\,N' : B}\,{\rightarrowtail}E$$

Additive Pairs.

$$\frac{\Psi \vdash \mathsf{fst}\,M \Uparrow M_1' : A \qquad \Psi \vdash \mathsf{snd}\,M \Uparrow M_2' : B}{\Psi \vdash M \Uparrow \langle M_1',\, M_2' \rangle : A \,\&\, B}\,\&I$$

$$\frac{\Psi \vdash M \downarrow M' : A \,\&\, B}{\Psi \vdash \mathsf{fst}\,M \downarrow \mathsf{fst}\,M' : A}\,\&E1 \qquad \frac{\Psi \vdash M \downarrow M' : A \,\&\, B}{\Psi \vdash \mathsf{snd}\,M \downarrow \mathsf{snd}\,M' : B}\,\&E2$$

Additive Unit.

$$\frac{}{\Psi \vdash M \Uparrow \langle\rangle : \top}\,\top I$$

We first state the following soundness lemma for the transformation process. Note that since the transformation judgements ignore linearity and ordering constraints, it is only meant to be applied to well-typed terms.

Lemma 49
$\Gamma; \Delta; \Omega \vdash M : A$ *and* $\Gamma\Delta\Omega \vdash M \Uparrow M' : A$ *implies*
 $\Gamma; \Delta; \Omega \vdash M' : A$ *and* M' *is in long* $\beta\eta$*-normal form.*

Proof: By structural inductions on the given derivations. □

We use the notation $\Psi' \geq \Psi$ to denote that Ψ' contains all declarations in Ψ and possibly more. We then have the following weakening lemma.

Lemma 50

1. $\Psi \vdash M \Uparrow M' : A$ *and* $\Psi' \geq \Psi$ *implies* $\Psi' \vdash M \Uparrow M' : A$.

2. $\Psi \vdash M \downarrow M' : A$ *and* $\Psi' \geq \Psi$ *implies* $\Psi' \vdash M \downarrow M' : A$.

Proof: By structural induction on the given derivation. □

The following unary Kripke logical relation is the crux of our argument. It is defined by induction on the type A.

$\Psi \vdash M \in [\![a]\!]$ iff $\Psi \vdash M \Uparrow N : a$ for some N.

$\Psi \vdash M \in [\![A_1 \rightarrow A_2]\!]$ iff for all N and $\Psi' \geq \Psi$,
 $\Psi' \vdash N \in [\![A_1]\!]$ implies $\Psi' \vdash M\,N \in [\![A_2]\!]$.

$\Psi \vdash M \in [\![A_1 \multimap A_2]\!]$ iff for all N and $\Psi' \geq \Psi$,
 $\Psi' \vdash N \in [\![A_1]\!]$ implies $\Psi' \vdash M\,\hat{}\,N \in [\![A_2]\!]$.

$\Psi \vdash M \in [\![A_1 \twoheadrightarrow A_2]\!]$ iff for all N and $\Psi' \geq \Psi$,
 $\Psi' \vdash N \in [\![A_1]\!]$ implies $\Psi' \vdash M\,{}^{>}N \in [\![A_2]\!]$.

$\Psi \vdash M \in [\![A_1 \rightarrowtail A_2]\!]$ iff for all N and $\Psi' \geq \Psi$,

$\Psi' \vdash N \in \llbracket A_1 \rrbracket$ implies $\Psi' \vdash M^{<} N \in \llbracket A_2 \rrbracket$.

$\Psi \vdash M \in \llbracket A_1 \mathbin{\&} A_2 \rrbracket$ iff $\Psi \vdash \mathbf{fst}\, M \in \llbracket A_1 \rrbracket$ and $\Psi \vdash \mathbf{snd}\, M \in \llbracket A_2 \rrbracket$.

$\Psi \vdash M \in \llbracket \top \rrbracket$.

We can now formally state and prove the second part of our proof— that well-typed terms in the logical relation at all types have canonical forms. We can prove this only simultaneously with the reverse statement for terms with an atomic form.

Lemma 51 (Logical Relations and Canonical Forms)

1. $\Psi \vdash M \in \llbracket A \rrbracket$ *implies* $\Psi \vdash M \Uparrow N : A$ *for some* N.

2. $\Psi \vdash M \downarrow N : A$ *implies* $\Psi \vdash M \in \llbracket A \rrbracket$.

Proof: By induction on A using structural properties of contexts. We show the case for ordered right implication. All other cases are similar or simpler.

Case: $A = A_1 \twoheadrightarrow A_2$. For each of the two properties (1) and (2) we need two appeals to the induction hypothesis, one on part (1) and one on part (2). First we consider property (1).

$\Psi \vdash M \in \llbracket A_1 \twoheadrightarrow A_2 \rrbracket$	by assumption
$\Psi(x{:}A_1) \geq \Psi$	defn. of $\geq$
$\Psi(z{:}A_1) \vdash z \downarrow z : A_1$	by rule **ovar**
$\Psi(z{:}A_1) \vdash z \in \llbracket A_1 \rrbracket$	by ind. hyp. (2) on A_1
$\Psi(z{:}A_1) \vdash M^{>} z \in \llbracket A_2 \rrbracket$	by defn. of $\llbracket A_1 \twoheadrightarrow A_2 \rrbracket$
$\Psi(z{:}A_1) \vdash M^{>} z \Uparrow M_2 : A_2$	by ind. hyp. (1) on A_2
$\Psi \vdash M \Uparrow \overset{>}{\lambda} z{:}A_1.\ M_2 : A_1 \twoheadrightarrow A_2$	by rule $\twoheadrightarrow I$

Next we show property (2).

$\Psi \vdash M \downarrow M' : A_1 \twoheadrightarrow A_2$	by assumption
$\Psi' \geq \Psi$ and $\Psi' \vdash N \in \llbracket A_1 \rrbracket$	new assumptions
$\Psi' \vdash N \Uparrow N' : A_1$	by ind. hyp. (1) on A_1
$\Psi' \vdash M \downarrow M' : A_1 \twoheadrightarrow A_2$	Lemma 50
$\Psi' \vdash M^{>} N \downarrow M'^{>} N' : A_2$	by rule $\twoheadrightarrow E$
$\Psi' \vdash M^{>} N \in \llbracket A_2 \rrbracket$	by ind. hyp. (2) on A_2
$\Psi \vdash M \in \llbracket A_1 \twoheadrightarrow A_2 \rrbracket$	by defn. of $\llbracket A_1 \twoheadrightarrow A_2 \rrbracket$

□

To prove that every well-typed term is in the logical relation we need closure under head expansion.

Lemma 52 (Closure Under Head Expansion)

$M \xrightarrow{\mathbf{whr}} M'$ *and* $\Psi \vdash M' \in \llbracket A \rrbracket$ *implies* $\Psi \vdash M \in \llbracket A \rrbracket$.

Proof: By induction on A. We show two representative cases.

Case: $A = a$. Then

$\Psi \vdash M' \in \llbracket a \rrbracket$ by assumption

$\Psi \vdash M' \Uparrow M'' : a$ by defn. of $\llbracket a \rrbracket$

$M \xrightarrow{\mathbf{whr}} M'$ by assumption

$\Psi \vdash M \Uparrow M'' : a$ by rule **reduction**

$\Psi \vdash M \in \llbracket a \rrbracket$ by defn. of $\llbracket a \rrbracket$

Case: $A = A_1 \twoheadrightarrow A_2$. Then

$\Psi \vdash M' \in \llbracket A_1 \twoheadrightarrow A_2 \rrbracket$ by assumption

$\Psi' \geq \Psi$ and $\Psi \vdash N \in \llbracket A_1 \rrbracket$ new assumptions

$\Psi' \vdash M'^{>} N \in \llbracket A_2 \rrbracket$ by defn. of $\llbracket A_1 \twoheadrightarrow A_2 \rrbracket$

$M \xrightarrow{\mathbf{whr}} M'$ by assumption

$M^{>} N \xrightarrow{\mathbf{whr}} M'^{>} N$ by rule $\mathbf{whr}_{\twoheadrightarrow}$

$\Psi' \vdash M^{>} N \in \llbracket A_2 \rrbracket$ by ind. hyp. on A_2

$\Psi \vdash M \in \llbracket A_1 \twoheadrightarrow A_2 \rrbracket$ by defn. of $\llbracket A_1 \twoheadrightarrow A_2 \rrbracket$

□

In order to show $\Gamma; \Delta; \Omega \vdash M : A$ implies $\Gamma\Delta\Omega \vdash M \in \llbracket A \rrbracket$, we need to explicitly manipulate substitutions. We shall define a substitution in the usual way as a list of variable assignments:

$$\sigma \quad ::= \quad \cdot \mid \sigma, M/x \mid \sigma, M/y \mid \sigma, M/z$$

where there is at most one assignment for any variable. Substitutions compose in the obvious way. We write id_Ψ for the identity substitution on the variables declared in Ψ. We define logical relations on substitutions by induction on the structure of contexts.

$$
\begin{array}{l}
\Psi' \vdash \sigma \in [\![\cdot]\!] \quad \text{iff} \quad \sigma = \cdot \\[1ex]
\Psi' \vdash (\sigma, M/x) \in [\![\Psi(x{:}A)]\!] \quad \text{iff} \\
\quad \Psi' \vdash \sigma \in [\![\Psi]\!] \quad \text{and} \quad \Psi' \vdash M \in [\![A]\!] \\[1ex]
\Psi' \vdash (\sigma, M/y) \in [\![\Psi(y{:}A)]\!] \quad \text{iff} \\
\quad \Psi' \vdash \sigma \in [\![\Psi]\!] \quad \text{and} \quad \Psi' \vdash M \in [\![A]\!] \\[1ex]
\Psi' \vdash (\sigma, M/z) \in [\![\Psi(z{:}A)]\!] \quad \text{iff} \\
\quad \Psi' \vdash \sigma \in [\![\Psi]\!] \quad \text{and} \quad \Psi' \vdash M \in [\![A]\!]
\end{array}
$$

Lemma 53 (Weakening for $[\![-]\!]$)

1. $\Psi \vdash M \in [\![A]\!]$ *and* $\Psi' \geq \Psi$ *implies* $\Psi' \vdash M \in [\![A]\!]$.

2. $\Psi' \vdash \sigma \in [\![\Psi]\!]$ *and* $\Psi'' \geq \Psi'$ *implies* $\Psi'' \vdash \sigma \in [\![\Psi]\!]$.

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

Lemma 54 (Identity) $\Psi \vdash id_\Psi \in [\![\Psi]\!]$

Proof: Immediate by definition and lemma 51. □

Lemma 55 (Typing and Logical Relations)
$\Gamma; \Delta; \Omega \vdash M : A$ *and* $\Psi' \geq \Gamma\Delta\Omega$ *and* $\Psi \vdash \sigma \in [\![\Psi']\!]$
implies $\Psi \vdash M[\sigma] \in [\![A]\!]$
where $M[\sigma]$ *is the result of applying substitution* σ *to* M*.*

Proof: By induction on the structure of the given typing derivation $\mathcal{D}$ using Lemma 52 and elementary inversion properties of the logical relations for substitutions. We show three representative cases.

Case:

$$\mathcal{D} \ = \ \frac{}{\Gamma; \cdot; z{:}A \vdash z{:}A}\,\textbf{ovar}$$

This case follows directly from the assumption that the substitution is in the logical relation.

$\Psi' \geq \Gamma(z{:}A)$ and $\Psi \vdash \sigma \in [\![\Psi']\!]$	assumptions
$\Psi \vdash M' \in [\![A]\!]$ where $M'/z \in \sigma$	by inversion properties of $[\![-]\!]$
$\Psi \vdash z[\sigma] \in [\![A]\!]$	by defn. of substitution

Case:

$$\mathcal{D} \ = \ \frac{\begin{array}{c}\mathcal{D}_2 \\ \Gamma; \Delta; \Omega(z{:}A_1) \vdash M_2 : A_2\end{array}}{\Gamma; \Delta; \Omega \vdash \overset{>}{\lambda} z{:}A_1.\ M_2 : A_1 \twoheadrightarrow A_2}\twoheadrightarrow I$$

In this case the critical step uses closure under head expansion (Lemma 52).

$\Psi' \geq \Gamma\Delta\Omega$ and $\Psi \vdash \sigma \in [\![\Psi']\!]$	assumptions
$\Psi'' \geq \Psi$ and $\Psi'' \vdash N \in [\![A_1]\!]$	new assumptions
$\Psi'' \vdash \sigma \in [\![\Psi']\!]$	Lemma 53
$\Psi'' \vdash (\sigma, N/z) \in [\![\Psi'(z{:}A_1)]\!]$	by defn. of $[\![-]\!]$
$\Psi'' \vdash M_2[\sigma, N/z] \in [\![A_2]\!]$	by ind. hyp. on $\mathcal{D}_2$
$\Psi'' \vdash (\overset{>}{\lambda} z{:}A_1.\ M_2[\sigma, z/z])^{>} N \in [\![A_2]\!]$	by Lemma 52
$\Psi \vdash \overset{>}{\lambda} z{:}A_1.\ M_2[\sigma, z/z] \in [\![A_1 \twoheadrightarrow A_2]\!]$	by defn. of $[\![A_1 \twoheadrightarrow A_2]\!]$
$\Psi \vdash (\overset{>}{\lambda} z{:}A_1.\ M_2)[\sigma] \in [\![A_1 \twoheadrightarrow A_2]\!]$	by defn. of substitution

Case:

$$\mathcal{D} \ = \ \frac{\begin{array}{c}\mathcal{D}_1 \\ \Gamma; \Delta_1; \Omega_1 \vdash M_1 : A_2 \twoheadrightarrow A_1\end{array} \qquad \begin{array}{c}\mathcal{D}_2 \\ \Gamma; \Delta_2; \Omega_2 \vdash M_2 : A_2\end{array}}{\Gamma; \Delta_1 \bowtie \Delta_2; \Omega_1\Omega_2 \vdash M_1{}^{>} M_2 : A_1}\twoheadrightarrow E$$

In this case the result follows by elementary inversion properties for the logical relation on substitutions.

$\Psi' \geq \Gamma(\Delta_1 \bowtie \Delta_2)\Omega_1\Omega_2$ and $\Psi \vdash \sigma \in \llbracket \Psi' \rrbracket$ by assumption

$\Psi \vdash M_1[\sigma] \in \llbracket A_2 \twoheadrightarrow A_1 \rrbracket$ by ind. hyp. on $\mathcal{D}_1$

$\Psi \vdash M_2[\sigma] \in \llbracket A_2 \rrbracket$ by ind. hyp. on $\mathcal{D}_2$

$\Psi \vdash (M_1[\sigma])^{>}(M_2[\sigma]) \in \llbracket A_1 \rrbracket$ by defn. of $\llbracket A_2 \twoheadrightarrow A_1 \rrbracket$

$\Psi \vdash (M_1{}^{>}M_2)[\sigma] \in \llbracket A_1 \rrbracket$ by defn. of substitution

□

Theorem 56 (Canonical Forms)
$\Gamma; \Delta; \Omega \vdash M : A$ *implies for some* N, $\Gamma\Delta\Omega \vdash M \Uparrow N : A$.

Proof: Immediate from lemmas 55, 51, and 54. □

Now that we know the ordered type system has a canonical fragment, it is natural to start thinking about basing a logical framework on the system. However, most interesting logical framework applications require the ability to quantify over terms. Towards this end, we shall extend the type system with dependent types in Chapter 13, and then proceed to show, in Chapters 14 and 15 that type-checking remains decidable and canonical forms still exist. The proof techniques we use, which are essentially generalizations of the method employed in this chapter, follow the development in [24] and [59] extended to the ordered case. We show an extended application, concerning syntactic properties of the CPS transform, of the resulting ordered logical framework in Chapter 16.

Chapter 13

Ordered Logical Framework

Logical frameworks are formal systems (meta-languages) suitable for encoding and reasoning about deductive systems (object languages). For a general overview of logical frameworks and some of their uses, see [46]. The Edinburgh logical framework (LF), introduced in [23], is a particular logical framework based on dependent types; it may be intuitively thought of as λP in the lambda cube [6][1].

LF representations typically follow the slogan *judgements as types*, with a new type declared for each judgement to be represented; derivations then become LF terms. Dependent types allow for a smooth encoding of higher-level properties of the object system in the same manner. For example, we can encode Mini-ML, evaluation for Mini-ML, and a proof of value soundness all with the judgements-as-types methodology by using dependent types– see [47] for details. This technique essentially reduces checking the validity of object-level deductions to meta-level type-checking.

In order for such encodings to really be useful, LF must satisfy two key properties. Every LF term, of a type representing an object-level judgement, should correspond to some object-level derivation of that judgement. In order to establish a bijection, we require that LF have a notion of canonical forms to represent all LF terms for any particular type. Furthermore, we should be able to decide whether a given object-level deduction is valid. Thus, since type-checking a term corresponds to proving a deduction valid, LF type-checking should be decidable.

An additional aspect of LF representations is the use of higher-order abstract syntax, an idea going back to Church [12], which employs meta-level variables to

[1] This intuition is not exact since LF is actually a proper subset of λP as noted in [31].

represent object-level variables. With this technique, we can often avoid explicitly encoding the machinery of object-level substitution; instead, we may rely upon the underlying LF substitution mechanism. Thus, higher-order abstract syntax allows for particularly elegant encodings of systems with variable bindings.

However, higher-order abstract syntax only works when the object-level variables behave in the same manner as the meta-level variables. Thus one could not use the technique to accurately represent linear functions, since LF variables are unrestricted. In order to allow the use of LF-style representation techniques on a wider class of systems, Cervesato and Pfenning investigated the extension of LF with linear types [11]. The resulting linear logical framework (LLF), which conservatively extends LF, permits LF-style representation of linear systems such as those involving state. For technical reasons, only non-dependent linear types were considered and additive conjunctions including the additive unit were added to the framework.

In this chapter, we add dependent types to the basic ordered type system of Chapter 12 with the intention of forming an LF-style logical framework. Following LLF, we only allow unrestricted dependencies– types can only depend upon unrestricted variables. We will prove the decidability of type-checking for the system, in Chapter 15, with a simple extension of the techniques of [24] and [59] to the ordered case. This proof can be thought of as a generalization of the basic techniques used to prove the existence of canonical forms in the non-dependent type-theory (Section 12.2). The existence of canonical forms for the dependent system will be a by-product of a type-checking algorithm.

Our development precisely follows that of [24] and [59]. The addition of ordered types does not affect the development in any significant manner (except to add more cases) since, for the purposes of the equality checking, ordered types are identical to linear types. The details of all proofs in this and the subsequent two chapters, may be found in [24] and [59].

13.1 Ordered Logical Framework

We introduce the type system which we will refer to as the ordered logical framework (OLF). It is essentially the ordered type system of Chapter 12 augmented with dependent types, *i.e.*, types which may depend upon terms. In order to facilitate a clear exposition, we will shift our nomenclature to *objects*, *families*, and *kinds*. Objects correspond to terms, families correspond to types, while kinds are new constructs which classify families (*i.e.*, kinds are "types" for families). Here is the syntax for OLF.

Syntax

$$
\begin{array}{rcll}
\text{Kinds} & K & ::= & \texttt{type} \mid \Pi x{:}A.\ K \\
\text{Families} & A & ::= & a \mid A\,M \mid \\
 & & & \Pi x{:}A_1.\ A_2 \mid A_1 \multimap A_2 \mid \\
 & & & A_1 \twoheadrightarrow A_2 \mid A_1 \rightarrowtail A_2 \mid \\
 & & & A_1 \,\&\, A_2 \mid \top \\
\text{Objects} & M & ::= & c \mid x \mid y \mid z \mid \\
 & & & \lambda x{:}A.\ M \mid M_1\,M_2 \mid \\
 & & & \hat{\lambda} y{:}A.\ M \mid M_1\,{}^{\wedge} M_2 \mid \\
 & & & \overset{>}{\lambda} z{:}A.\ M \mid M_1\,{}^{>} M_2 \mid \\
 & & & \overset{<}{\lambda} z{:}A.\ M \mid M_1\,{}^{<} M_2 \mid \\
 & & & \langle M_1\,,\, M_2 \rangle \mid \mathbf{fst}\ M \mid \mathbf{snd}\ M \mid \langle\rangle \\
\text{Signatures} & \Sigma & ::= & \cdot \mid \Sigma(a{:}K) \mid \Sigma(c{:}A) \\
\text{Unrestricted Contexts} & \Gamma & ::= & \cdot \mid \Gamma(x{:}A) \\
\text{Linear Contexts} & \Delta & ::= & \cdot \mid \Delta(y{:}A) \\
\text{Ordered Contexts} & \Omega & ::= & \cdot \mid \Omega(z{:}A)
\end{array}
$$

We also use N for objects and B for families. We continue our convention of syntactically distinguishing unrestricted, linear and ordered assumptions. Thus, like LLF, only unrestricted assumptions may appear in families and kinds since Πs only bind unrestricted variables, x.

We will employ the following judgements to define the OLF type theory.

Judgements

$\vdash \Sigma\ \texttt{sig}$	Σ is a valid signature
$\vdash_\Sigma \Gamma\ \texttt{uctx}$	Γ is a valid unrestricted context
$\Gamma \vdash_\Sigma \Delta\ \texttt{lctx}$	Δ is a valid linear context
$\Gamma \vdash_\Sigma \Omega\ \texttt{octx}$	Ω is a valid ordered context

$\Gamma \vdash_\Sigma K : \mathtt{kind}$	K is a valid kind
$\Gamma \vdash_\Sigma A : K$	A has kind K
$\Gamma; \Delta; \Omega \vdash_\Sigma M : A$	M has type A
$\Gamma \vdash_\Sigma K_1 = K_2 : \mathtt{kind}$	K_1 equals K_2
$\Gamma \vdash_\Sigma A_1 = A_2 : K$	A_1 equals A_2 at kind K
$\Gamma; \Delta; \Omega \vdash_\Sigma M_1 = M_2 : A$	M_1 equals M_2 at type A

We assume Σ is a valid signature in the judgement $\vdash_\Sigma \Gamma\ \mathtt{uctx}$. We assume Σ is a valid signature and Γ is valid in Σ for judgements of the form $\Gamma \vdash_\Sigma J$.

13.2 Typing Rules

Signatures

$$\frac{}{\vdash \cdot\ \mathtt{sig}} \qquad \frac{\vdash \Sigma\ \mathtt{sig} \qquad \cdot \vdash_\Sigma K : \mathtt{kind}}{\vdash \Sigma(a{:}K)\ \mathtt{sig}} \qquad \frac{\vdash \Sigma\ \mathtt{sig} \qquad \cdot \vdash_\Sigma A : \mathtt{type}}{\vdash \Sigma(c{:}A)\ \mathtt{sig}}$$

From this point on, we assume a valid signature, Σ, implicit in all judgements.

Contexts

$$\frac{}{\vdash \cdot\ \mathtt{uctx}} \qquad \frac{\vdash \Gamma\ \mathtt{uctx} \qquad \Gamma \vdash A : \mathtt{type}}{\vdash \Gamma(x{:}A)\ \mathtt{uctx}}$$

$$\frac{}{\Gamma \vdash \cdot\ \mathtt{lctx}} \qquad \frac{\Gamma \vdash \Delta\ \mathtt{lctx} \qquad \Gamma \vdash A : \mathtt{type}}{\Gamma \vdash \Delta(y{:}A)\ \mathtt{lctx}}$$

$$\frac{}{\Gamma \vdash \cdot\ \mathtt{octx}} \qquad \frac{\Gamma \vdash \Omega\ \mathtt{octx} \qquad \Gamma \vdash A : \mathtt{type}}{\Gamma \vdash \Omega(z{:}A)\ \mathtt{octx}}$$

From this point on, we presuppose the validity of all contexts in judgements, rather than explicitly checking this property.

Kinds

$$\frac{}{\Gamma \vdash \mathtt{type} : \mathtt{kind}} \qquad \frac{\Gamma \vdash A : \mathtt{type} \qquad \Gamma(x{:}A) \vdash K : \mathtt{kind}}{\Gamma \vdash \Pi x{:}A.\ K : \mathtt{kind}}$$

Families

$$\frac{a{:}K \in \Sigma}{\Gamma \vdash a : K} \qquad \frac{\Gamma \vdash A : \Pi x{:}A'.\, K \qquad \Gamma; \cdot; \cdot \vdash M : A'}{\Gamma \vdash A\, M : K[M/x]}$$

$$\frac{\Gamma \vdash A_1 : \mathtt{type} \qquad \Gamma(x{:}A_1) \vdash A_2 : \mathtt{type}}{\Gamma \vdash \Pi x{:}A_1.\, A_2 : \mathtt{type}}$$

$$\frac{\Gamma \vdash A : K \qquad \Gamma \vdash K = K' : \mathtt{kind}}{\Gamma \vdash A : K'}$$

Objects

$$\frac{c{:}A \in \Sigma}{\Gamma; \cdot; \cdot \vdash c : A} \qquad \frac{x{:}A \in \Gamma}{\Gamma; \cdot; \cdot \vdash x : A} \qquad \frac{}{\Gamma; y{:}A; \cdot \vdash y : A} \qquad \frac{}{\Gamma; \cdot; z{:}A \vdash z : A}$$

$$\frac{\Gamma \vdash A' : \mathtt{type} \qquad \Gamma(x{:}A'); \Delta; \Omega \vdash M : A}{\Gamma; \Delta; \Omega \vdash \lambda x{:}A'.\, M : \Pi x{:}A'.\, A} \qquad \frac{\Gamma; \Delta; \Omega \vdash M : \Pi x{:}A'.\, A \qquad \Gamma; \cdot; \cdot \vdash M' : A'}{\Gamma; \Delta; \Omega \vdash M\, M' : A[M'/x]}$$

$$\frac{\Gamma \vdash A' : \mathtt{type} \qquad \Gamma; \Delta(y{:}A'); \Omega \vdash M : A}{\Gamma; \Delta; \Omega \vdash \hat{\lambda} y{:}A'.\, M : A' \multimap A} \qquad \frac{\Gamma; \Delta; \Omega \vdash M : A' \multimap A \qquad \Gamma; \Delta'; \cdot \vdash M' : A'}{\Gamma; \Delta \bowtie \Delta'; \Omega \vdash M\,\hat{}\,M' : A}$$

$$\frac{\Gamma \vdash A' : \mathtt{type} \qquad \Gamma; \Delta; \Omega(z{:}A') \vdash M : A}{\Gamma; \Delta; \Omega \vdash \overset{>}{\lambda} z{:}A'.\, M : A' \twoheadrightarrow A} \qquad \frac{\Gamma; \Delta; \Omega \vdash M : A' \twoheadrightarrow A \qquad \Gamma; \Delta'; \Omega'; \cdot \vdash M' : A'}{\Gamma; \Delta \bowtie \Delta'; \Omega\Omega' \vdash M^{>} M' : A}$$

$$\frac{\Gamma \vdash A' : \mathtt{type} \qquad \Gamma; \Delta; (z{:}A')\Omega \vdash M : A}{\Gamma; \Delta; \Omega \vdash \overset{<}{\lambda} z{:}A'.\, M : A' \rightarrowtail A} \qquad \frac{\Gamma; \Delta; \Omega \vdash M : A' \rightarrowtail A \qquad \Gamma; \Delta'; \Omega'; \cdot \vdash M' : A'}{\Gamma; \Delta \bowtie \Delta'; \Omega'\Omega \vdash M^{<} M' : A}$$

$$\frac{}{\Gamma; \Delta; \Omega \vdash \langle\rangle : \top} \qquad \frac{\Gamma; \Delta; \Omega \vdash M_1 : A_1 \qquad \Gamma; \Delta; \Omega \vdash M_2 : A_2}{\Gamma; \Delta; \Omega \vdash \langle M_1 \,,\, M_2 \rangle : A_1 \,\&\, A_2}$$

$$\frac{\Gamma; \Delta; \Omega \vdash M : A_1 \,\&\, A_2}{\Gamma; \Delta; \Omega \vdash \mathbf{fst}\, M : A_1} \qquad \frac{\Gamma; \Delta; \Omega \vdash M : A_1 \,\&\, A_2}{\Gamma; \Delta; \Omega \vdash \mathbf{snd}\, M : A_2}$$

$$\frac{\Gamma; \Delta; \Omega \vdash M : A \qquad \Gamma \vdash A = A' : \mathtt{type}}{\Gamma; \Delta; \Omega \vdash M : A'}$$

13.3 Definitional Equality

In this section we define the equality judgement used in the kind conversion rule for families and the type conversion rule for objects; the last typing rules for families and objects respectively. Following [24], this equality is based on a notion of parallel conversion plus extensionality, rather than directly on $\beta\eta$-conversion. We do not include explicit reflexivity rules since they are admissible (Lemma 58).

Simultaneous Congruence

$$\frac{c{:}A \in \Sigma}{\Gamma; \cdot; \cdot \vdash c = c : A} \qquad \frac{x{:}A \in \Gamma}{\Gamma; \cdot; \cdot \vdash x = x : A}$$

$$\frac{}{\Gamma; y{:}A; \cdot \vdash y = y : A} \qquad \frac{}{\Gamma; \cdot; z{:}A \vdash z = z : A}$$

$$\frac{\Gamma \vdash A'_1 = A' : \texttt{type} \qquad \Gamma \vdash A'_2 = A' : \texttt{type} \qquad \Gamma(x{:}A'); \Delta; \Omega \vdash M = N : A}{\Gamma; \Delta; \Omega \vdash \lambda x{:}A'_1.\ M = \lambda x{:}A'_2.\ N : \Pi x{:}A'.\ A}$$

$$\frac{\Gamma; \Delta; \Omega \vdash M = N : \Pi x{:}A'.\ A \qquad \Gamma; \cdot; \cdot \vdash M' = N' : A'}{\Gamma; \Delta; \Omega \vdash M\ M' = N\ N' : A[M'/x]}$$

$$\frac{\Gamma \vdash A'_1 = A' : \texttt{type} \qquad \Gamma \vdash A'_2 = A' : \texttt{type} \qquad \Gamma; \Delta(y{:}A'); \Omega \vdash M = N : A}{\Gamma; \Delta; \Omega \vdash \hat{\lambda} y{:}A'_1.\ M = \hat{\lambda} y{:}A'_2.\ N : A' \multimap A}$$

$$\frac{\Gamma; \Delta; \Omega \vdash M = N : A' \multimap A \qquad \Gamma; \Delta'; \cdot \vdash M' = N' : A'}{\Gamma; \Delta \bowtie \Delta'; \Omega \vdash M\,\hat{}\,M' = N\,\hat{}\,N' : A}$$

$$\frac{\Gamma \vdash A'_1 = A' : \texttt{type} \qquad \Gamma \vdash A'_2 = A' : \texttt{type} \qquad \Gamma; \Delta; \Omega(z{:}A') \vdash M = N : A}{\Gamma; \Delta; \Omega \vdash \overset{>}{\lambda} z{:}A'_1.\ M = \overset{>}{\lambda} z{:}A'_2.\ N : A' \twoheadrightarrow A}$$

$$\frac{\Gamma; \Delta; \Omega \vdash M = N : A' \twoheadrightarrow A \qquad \Gamma; \Delta'; \Omega' \vdash M' = N' : A'}{\Gamma; \Delta \bowtie \Delta'; \Omega\Omega' \vdash M^{>}M' = N^{>}N' : A}$$

$$\frac{\Gamma \vdash A_1' = A' : \texttt{type} \qquad \Gamma \vdash A_2' = A' : \texttt{type} \qquad \Gamma; \Delta; (z{:}A')\Omega \vdash M = N : A}{\Gamma; \Delta; \Omega \vdash \overset{<}{\lambda} z{:}A_1'.\ M = \overset{<}{\lambda} z{:}A_2'.\ N : A' \rightarrowtail A}$$

$$\frac{\Gamma; \Delta; \Omega \vdash M = N : A' \rightarrowtail A \qquad \Gamma; \Delta'; \Omega' \vdash M' = N' : A'}{\Gamma; \Delta \bowtie \Delta'; \Omega'\Omega \vdash M^{<} M' = N^{<} N' : A}$$

$$\frac{\Gamma; \Delta; \Omega \vdash M_1 = N_1 : A_1 \qquad \Gamma; \Delta; \Omega \vdash M_2 = N_2 : A_2}{\Gamma; \Delta; \Omega \vdash \langle M_1\, ,\, M_2 \rangle = \langle N_1\, ,\, N_2 \rangle : A_1 \mathbin{\&} A_2}$$

$$\frac{\Gamma; \Delta; \Omega \vdash M = N : A_1 \mathbin{\&} A_2}{\Gamma; \Delta; \Omega \vdash \textbf{fst}\, M = \textbf{fst}\, N : A_1} \qquad \frac{\Gamma; \Delta; \Omega \vdash M = N : A_1 \mathbin{\&} A_2}{\Gamma; \Delta; \Omega \vdash \textbf{snd}\, M = \textbf{snd}\, N : A_2}$$

Extensionality

$$\frac{\Gamma \vdash A' : \texttt{type} \qquad \begin{array}{l} \Gamma; \Delta; \Omega \vdash M : \Pi x{:}A'.\ A \\ \Gamma; \Delta; \Omega \vdash N : \Pi x{:}A'.\ A \end{array} \qquad \Gamma(x{:}A'); \Delta; \Omega \vdash M\, x = N\, x : A}{\Gamma; \Delta; \Omega \vdash M = N : \Pi x{:}A'.\ A}$$

$$\frac{\Gamma \vdash A' : \texttt{type} \qquad \begin{array}{l} \Gamma; \Delta; \Omega \vdash M : A' \multimap A \\ \Gamma; \Delta; \Omega \vdash N : A' \multimap A \end{array} \qquad \Gamma; \Delta(y{:}A'); \Omega \vdash M\,\hat{}\,y = N\,\hat{}\,y : A}{\Gamma; \Delta; \Omega \vdash M = N : A' \multimap A}$$

$$\frac{\Gamma \vdash A' : \texttt{type} \qquad \begin{array}{l} \Gamma; \Delta; \Omega \vdash M : A' \twoheadrightarrow A \\ \Gamma; \Delta; \Omega \vdash N : A' \twoheadrightarrow A \end{array} \qquad \Gamma; \Delta; \Omega(z{:}A') \vdash M^{>} z = N^{>} z : A}{\Gamma; \Delta; \Omega \vdash M = N : A' \twoheadrightarrow A}$$

$$\frac{\Gamma \vdash A' : \texttt{type} \qquad \begin{array}{l} \Gamma; \Delta; \Omega \vdash M : A' \rightarrowtail A \\ \Gamma; \Delta; \Omega \vdash N : A' \rightarrowtail A \end{array} \qquad \Gamma; \Delta; (z{:}A')\Omega \vdash M^{<} z = N^{<} z : A}{\Gamma; \Delta; \Omega \vdash M = N : A' \rightarrowtail A}$$

$$\frac{\begin{array}{l}\Gamma;\Delta;\Omega \vdash M : A_1 \mathbin{\&} A_2 \\ \Gamma;\Delta;\Omega \vdash N : A_1 \mathbin{\&} A_2 \qquad \Gamma;\Delta;\Omega \vdash \mathbf{fst}\, M = \mathbf{fst}\, N : A_1 \qquad \Gamma;\Delta;\Omega \vdash \mathbf{snd}\, M = \mathbf{snd}\, N : A_2\end{array}}{\Gamma;\Delta;\Omega \vdash M = N : A_1 \mathbin{\&} A_2}$$

$$\frac{\Gamma;\Delta;\Omega \vdash M : \top \qquad \Gamma;\Delta;\Omega \vdash N : \top}{\Gamma;\Delta;\Omega \vdash M = N : \top}$$

Parallel Conversion

$$\frac{\Gamma \vdash A' : \mathtt{type} \qquad \Gamma(x{:}A');\Delta;\Omega \vdash M = N : A \qquad \Gamma;\cdot;\cdot \vdash M' = N' : A'}{\Gamma;\Delta;\Omega \vdash (\lambda x{:}A'.\; M)\, M' = N[N'/x] : A[N'/x]}$$

$$\frac{\Gamma \vdash A' : \mathtt{type} \qquad \Gamma;\Delta(y{:}A');\Omega \vdash M = N : A \qquad \Gamma;\Delta';\cdot \vdash M' = N' : A'}{\Gamma;\Delta \bowtie \Delta';\Omega \vdash (\hat{\lambda} y{:}A'.\; M)^{\hat{}}\, M' = N[N'/y] : A}$$

$$\frac{\Gamma \vdash A' : \mathtt{type} \qquad \Gamma;\Delta;\Omega(z{:}A') \vdash M = N : A \qquad \Gamma;\Delta';\Omega' \vdash M' = N' : A'}{\Gamma;\Delta \bowtie \Delta';\Omega\Omega' \vdash (\overset{>}{\lambda} z{:}A'.\; M)^{>}\, M' = N[N'/z] : A}$$

$$\frac{\Gamma \vdash A' : \mathtt{type} \qquad \Gamma;\Delta;(z{:}A')\Omega \vdash M = N : A \qquad \Gamma;\Delta';\Omega' \vdash M' = N' : A'}{\Gamma;\Delta \bowtie \Delta';\Omega'\Omega \vdash (\overset{<}{\lambda} z{:}A'.\; M)^{<}\, M' = N[N'/z] : A}$$

$$\frac{\Gamma;\Delta;\Omega \vdash M_1 = N_1 : A_1 \qquad \Gamma;\Delta;\Omega \vdash M_2 = N_2 : A_2}{\Gamma;\Delta;\Omega \vdash \mathbf{fst}\, \langle M_1\, ,\, M_2 \rangle = N_1 : A_1}$$

$$\frac{\Gamma;\Delta;\Omega \vdash M_1 = N_1 : A_1 \qquad \Gamma;\Delta;\Omega \vdash M_2 = N_2 : A_2}{\Gamma;\Delta;\Omega \vdash \mathbf{snd}\, \langle M_1\, ,\, M_2 \rangle = N_2 : A_2}$$

Equivalence

$$\frac{\Gamma;\Delta;\Omega \vdash M = N : A}{\Gamma;\Delta;\Omega \vdash N = M : A} \qquad \frac{\Gamma;\Delta;\Omega \vdash M = M' : A \qquad \Gamma;\Delta;\Omega \vdash M' = N : A}{\Gamma;\Delta;\Omega \vdash M = N : A}$$

Type Conversion

$$\frac{\Gamma; \Delta; \Omega \vdash M = N : A \qquad \Gamma \vdash A = A' : \mathtt{type}}{\Gamma; \Delta; \Omega \vdash M = N : A'}$$

Family Congruence

$$\frac{a{:}K \in \Sigma}{\Gamma \vdash a = a : K}$$

$$\frac{\Gamma \vdash A_1 = A_2 : \Pi x{:}A'.\ K \qquad \Gamma; \cdot; \cdot \vdash M_1 = M_2 : A'}{\Gamma \vdash A_1\, M_1 = A_2\, M_2 : K[N_1/x]}$$

$$\frac{\Gamma \vdash A_1' : \mathtt{type} \qquad \Gamma \vdash A_1' = A_2' : \mathtt{type} \qquad \Gamma(x{:}A_1') \vdash A_1 = A_2 : \mathtt{type}}{\Gamma \vdash \Pi x{:}A_1'.\ A_1 = \Pi x{:}A_2'.\ A_2 : \mathtt{type}}$$

$$\frac{\Gamma \vdash A_1' = A_2' : \mathtt{type} \qquad \Gamma \vdash A_1 = A_2 : \mathtt{type}}{\Gamma \vdash A_1' \multimap A_1 = A_2' \multimap A_2 : \mathtt{type}}$$

$$\frac{\Gamma \vdash A_1' = A_2' : \mathtt{type} \qquad \Gamma \vdash A_1 = A_2 : \mathtt{type}}{\Gamma \vdash A_1' \twoheadrightarrow A_1 = A_2' \twoheadrightarrow A_2 : \mathtt{type}}$$

$$\frac{\Gamma \vdash A_1' = A_2' : \mathtt{type} \qquad \Gamma \vdash A_1 = A_2 : \mathtt{type}}{\Gamma \vdash A_1' \rightarrowtail A_1 = A_2' \rightarrowtail A_2 : \mathtt{type}}$$

$$\frac{\Gamma \vdash A_1' = A_2' : \mathtt{type} \qquad \Gamma \vdash A_1 = A_2 : \mathtt{type}}{\Gamma \vdash A_1' \,\&\, A_1 = A_2' \,\&\, A_2 : \mathtt{type}}$$

$$\frac{}{\Gamma \vdash \top = \top : \mathtt{type}}$$

Family Equivalence

$$\frac{\Gamma \vdash A_1 = A_2 : K}{\Gamma \vdash A_2 = A_1 : K} \qquad \frac{\Gamma \vdash A_1 = A_1' : K \qquad \Gamma \vdash A_1' = A_2 : K}{\Gamma \vdash A_1 = A_2 : K}$$

Kind Conversion

$$\frac{\Gamma \vdash A_1 = A_2 : K \qquad \Gamma \vdash K = K' : \texttt{kind}}{\Gamma \vdash A_1 = A_2 : K'}$$

Kind Congruence

$$\frac{}{\Gamma \vdash \texttt{type} = \texttt{type} : \texttt{kind}}$$

$$\frac{\Gamma \vdash A_1 : \texttt{type} \qquad \Gamma \vdash A_1 = A_2 : \texttt{type} \qquad \Gamma(x{:}A_1) \vdash K_1 = K_2 : \texttt{kind}}{\Gamma \vdash \Pi x{:}A_1.\ K_1 = \Pi x{:}A_2.\ K_2 : \texttt{kind}}$$

Kind Equivalence

$$\frac{\Gamma \vdash K_1 = K_2 : \texttt{kind}}{\Gamma \vdash K_2 = K_1 : \texttt{kind}} \qquad \frac{\Gamma \vdash K_1 = K_1' : \texttt{kind} \qquad \Gamma \vdash K_1' = K_2 : \texttt{kind}}{\Gamma \vdash K_1 = K_2 : \texttt{kind}}$$

13.4 Properties of Typing and Equality

We show some basic properties of definitional equality. We use J to stand for any judgement of the type theory to avoid some repetitive statements. Substitution is extended to J in the obvious way; for example if J is $N : B$, then $J[M/x]$ is $N[M/x] : B[M/x]$.

Lemma 57 (Weakening)

1. $\Gamma\Gamma' \vdash J$ *implies* $\Gamma(x{:}A)\Gamma' \vdash J$.

2. $\Gamma\Gamma'; \Delta; \Omega \vdash J$ *implies* $\Gamma(x{:}A)\Gamma'; \Delta; \Omega \vdash J$.

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

Lemma 58 (Reflexivity)

1. $\Gamma; \Delta; \Omega \vdash M : A$ *implies* $\Gamma; \Delta; \Omega \vdash M = M : A$.

2. $\Gamma \vdash A : K$ *implies* $\Gamma \vdash A = A : K$.

3. $\Gamma \vdash K : \texttt{kind}$ *implies* $\Gamma \vdash K = K : \texttt{kind}$.

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

Lemma 59 (Substitution Property)

Assume all contexts are valid.

1. $\Gamma; \cdot; \cdot \vdash M : A$ *and* $\Gamma(x{:}A)\Gamma' \vdash J$ *implies* $\Gamma(\Gamma'[M/x]) \vdash J[M/x]$.

2. $\Gamma; \cdot; \cdot \vdash M : A$ *and* $\Gamma(x{:}A)\Gamma'; \Delta; \Omega \vdash J$ *implies*
$\Gamma(\Gamma'[M/x]); \Delta[M/x]; \Omega[M/x] \vdash J[M/x]$.

3. $\Gamma; \Delta'; \cdot \vdash M : A$ *and* $\Gamma; \Delta_1(y{:}A)\Delta_2; \Omega \vdash J$ *implies*
$\Gamma; (\Delta_1 \bowtie \Delta')\Delta_2; \Omega \vdash J[M/y]$.

4. $\Gamma; \Delta'; \Omega' \vdash M : A$ *and* $\Gamma; \Delta; \Omega_1(z{:}A)\Omega_2 \vdash J$ *implies*
$\Gamma; \Delta \bowtie \Delta'; \Omega_1\Omega'\Omega_2 \vdash J[M/z]$.

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

Lemma 60 (Context Conversion)

Assume $\vdash \Gamma(x{:}A)\,\texttt{uctx}$ *and* $\Gamma \vdash A' : \texttt{type}$ *and* $\Gamma \vdash A = A' : \texttt{type}$.

1. $\Gamma(x{:}A) \vdash J$ *implies* $\Gamma(x{:}A') \vdash J$.

2. $\Gamma(x{:}A); \Delta; \Omega \vdash J$ *implies* $\Gamma(x{:}A'); \Delta; \Omega \vdash J$.

Proof: By Lemmas 57 and 59. □

A stronger version of the next lemma which includes equality judgements is required. However, that must be postponed until after Lemma 63. For now we state the following weaker version which we use in the proof of Lemma 63.

Lemma 61 (Functionality for Typing)

Assume $\vdash \Gamma(x{:}A)\Gamma'\,\texttt{uctx}$ *and* $\Gamma; \cdot; \cdot \vdash M = M' : A$ *and* $\Gamma; \cdot; \cdot \vdash M : A$ *and* $\Gamma; \cdot; \cdot \vdash M' : A$.

1. $\Gamma(x{:}A)\Gamma'; \Delta; \Omega \vdash N : B$ *implies*
   $\Gamma(\Gamma'[M/x]); \Delta[M/x]; \Omega[M/x] \vdash N[M/x] = N[M'/x] : B[M/x]$.

2. $\Gamma(x{:}A)\Gamma' \vdash B : K$ *implies* $\Gamma(\Gamma'[M/x]) \vdash B[M/x] = B[M'/x] : K[M/x]$.

3. $\Gamma(x{:}A)\Gamma' \vdash K : \mathtt{kind}$ *implies* $\Gamma(\Gamma'[M/x]) \vdash K[M/x] = K[M'/x] : \mathtt{kind}$.

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

Again we postpone a stronger version of the next lemma until after Lemma 63.

Lemma 62 (Inversion on Simple Types and Kinds)

1. $\Gamma \vdash \Pi x{:}A_1.\, A_2 : K$ *implies* $\Gamma \vdash A_1 : \mathtt{type}$ *and* $\Gamma(x{:}A_1) \vdash A_2 : \mathtt{type}$.

2. $\Gamma \vdash A_1 \multimap A_2 : K$ *implies* $\Gamma \vdash A_1 : \mathtt{type}$ *and* $\Gamma \vdash A_2 : \mathtt{type}$.

3. $\Gamma \vdash A_1 \twoheadrightarrow A_2 : K$ *implies* $\Gamma \vdash A_1 : \mathtt{type}$ *and* $\Gamma \vdash A_2 : \mathtt{type}$.

4. $\Gamma \vdash A_1 \rightarrowtail A_2 : K$ *implies* $\Gamma \vdash A_1 : \mathtt{type}$ *and* $\Gamma \vdash A_2 : \mathtt{type}$.

5. $\Gamma \vdash A_1 \,\&\, A_2 : K$ *implies* $\Gamma \vdash A_1 : \mathtt{type}$ *and* $\Gamma \vdash A_2 : \mathtt{type}$.

6. $\Gamma \vdash \Pi x{:}A.\, K : \mathtt{kind}$ *implies* $\Gamma \vdash A : \mathtt{type}$ *and* $\Gamma(x{:}A) \vdash K : \mathtt{kind}$.

Proof: Parts 1 through 5 by induction on the structure of the given derivation. Part 6 is immediate. □

Lemma 63 (Validity)

Assume $\vdash \Gamma\ \mathtt{uctx}$ *and* $\Gamma \vdash \Delta\ \mathtt{lctx}$ *and* $\Gamma \vdash \Omega\ \mathtt{octx}$.

1. $\Gamma; \Delta; \Omega \vdash M : A$ *implies* $\Gamma \vdash A : \mathtt{type}$.

2. $\Gamma; \Delta; \Omega \vdash M = N : A$ *implies*
   $\Gamma \vdash A : \mathtt{type}$ *and* $\Gamma; \Delta; \Omega \vdash M : A$ *and* $\Gamma; \Delta; \Omega \vdash N : A$.

3. $\Gamma \vdash A : K$ *implies* $\Gamma \vdash K : \mathtt{kind}$.

4. $\Gamma \vdash A = B : K$ *implies* $\Gamma \vdash K : \mathtt{kind}$ *and* $\Gamma \vdash A : K$ *and* $\Gamma \vdash B : K$.

5. $\Gamma \vdash K = K' : \mathtt{kind}$ *implies* $\Gamma \vdash K : \mathtt{kind}$ *and* $\Gamma \vdash K' : \mathtt{kind}$.

Proof: By induction on the structure of the given derivation making use of Lemmas 61 and 62. □

Lemma 64 (Functionality for Equality)

Assume $\vdash \Gamma(x{:}A)\ \texttt{uctx}$ *and* $\Gamma(x{:}A) \vdash \Delta\ \texttt{lctx}$ *and* $\Gamma(x{:}A) \vdash \Omega\ \texttt{octx}$ *and* $\Gamma; \cdot; \cdot \vdash M = M' : A$.

1. $\Gamma(x{:}A); \Delta; \Omega \vdash N = N' : B$ *implies*
   $\Gamma; \Delta[M/x]; \Omega[M/x] \vdash N[M/x] = N'[M'/x] : B[M/x]$.

2. $\Gamma(x{:}A) \vdash A = B : K$ *implies* $\Gamma \vdash A[M/x] = B[M'/x] : K[M/x]$.

3. $\Gamma(x{:}A) \vdash K = K' : \texttt{kind}$ *implies* $\Gamma \vdash K[M/x] = K'[M'/x] : \texttt{kind}$.

Proof: By Lemmas 63, 59 and 61. □

Lemma 65 (Typing Inversion)

Assume $\vdash \Gamma\ \texttt{uctx}$ *and* $\Gamma \vdash \Delta\ \texttt{lctx}$ *and* $\Gamma \vdash \Omega\ \texttt{octx}$.

1. $\Gamma; \Delta; \Omega \vdash c{:}A$ *implies* $\Delta = \cdot = \Omega$ *and* $c{:}B \in \Sigma$ *and* $\Gamma \vdash A = B : \texttt{type}$.

2. $\Gamma; \Delta; \Omega \vdash x : A$ *implies* $\Delta = \cdot = \Omega$ *and* $x{:}B \in \Gamma$ *and* $\Gamma \vdash A = B : \texttt{type}$.

3. $\Gamma; \Delta; \Omega \vdash y : A$ *implies* $\Delta = y{:}B$ *and* $\Omega = \cdot$ *and* $\Gamma \vdash A = B : \texttt{type}$.

4. $\Gamma; \Delta; \Omega \vdash z : A$ *implies* $\Delta = \cdot$ *and* $\Omega = y{:}B$ *and* $\Gamma \vdash A = B : \texttt{type}$.

5. $\Gamma; \Delta; \Omega \vdash M_1\, M_2 : A$ *implies*
   $\Gamma; \Delta; \Omega \vdash M_1 : \Pi x{:}A_2.\, A_1$ *and* $\Gamma; \cdot; \cdot \vdash M_2 : A_2$ *and*
   $\Gamma \vdash A_1[M_2/x] = A : \texttt{type}$.

6. $\Gamma; \Delta; \Omega \vdash \lambda x{:}A.\, M : B$ *implies*
   $\Gamma \vdash B = \Pi x{:}A.\, A' : \texttt{type}$ *and* $\Gamma \vdash A : \texttt{type}$ *and* $\Gamma(x{:}A); \Delta; \Omega \vdash M : A'$.

7. $\Gamma; \Delta; \Omega \vdash M_1 \hat{}\, M_2 : A$ *implies*
   $\Delta = \Delta_1 \bowtie \Delta_2$ *and* $\Gamma \vdash A_1 = A : \texttt{type}$ *and*
   $\Gamma; \Delta_1; \Omega \vdash M_1 : A_2 \multimap A_1$ *and* $\Gamma; \Delta_2; \cdot \vdash M_2 : A_2$.

8. $\Gamma; \Delta; \Omega \vdash \hat{\lambda} y{:}A.\, M : B$ *implies*
   $\Gamma \vdash B = A \multimap A' : \texttt{type}$ *and* $\Gamma \vdash A : \texttt{type}$ *and* $\Gamma; \Delta(y{:}A); \Omega \vdash M : A'$.

9. $\Gamma; \Delta; \Omega \vdash M_1 \,{}^{>} M_2 : A$ *implies*
$\Delta = \Delta_1 \bowtie \Delta_2$ *and* $\Omega = \Omega_1 \Omega_2$ *and* $\Gamma \vdash A_1 = A : \texttt{type}$ *and*
$\Gamma; \Delta_1; \Omega_1 \vdash M_1 : A_2 \twoheadrightarrow A_1$ *and* $\Gamma; \Delta_2; \Omega_2 \vdash M_2 : A_2$.

10. $\Gamma; \Delta; \Omega \vdash \overset{>}{\lambda} z{:}A.\, M : B$ *implies*
$\Gamma \vdash B = A \twoheadrightarrow A' : \texttt{type}$ *and* $\Gamma \vdash A : \texttt{type}$ *and* $\Gamma; \Delta; \Omega(z{:}A) \vdash M : A'$.

11. $\Gamma; \Delta; \Omega \vdash M_1 \,{}^{<} M_2 : A$ *implies*
$\Delta = \Delta_1 \bowtie \Delta_2$ *and* $\Omega = \Omega_2 \Omega_1$ *and* $\Gamma \vdash A_1 = A : \texttt{type}$ *and*
$\Gamma; \Delta_1; \Omega_1 \vdash M_1 : A_2 \rightarrowtail A_1$ *and* $\Gamma; \Delta_2; \Omega_2 \vdash M_2 : A_2$.

12. $\Gamma; \Delta; \Omega \vdash \overset{<}{\lambda} z{:}A.\, M : B$ *implies*
$\Gamma \vdash B = A \rightarrowtail A' : \texttt{type}$ *and* $\Gamma \vdash A : \texttt{type}$ *and* $\Gamma; \Delta; (z{:}A)\Omega \vdash M : A'$.

13. $\Gamma; \Delta; \Omega \vdash \mathbf{fst}\, M : A$ *implies* $\Gamma; \Delta; \Omega \vdash M : A_1 \,\&\, A_2$ *and* $\Gamma \vdash A = A_1 : \texttt{type}$.

14. $\Gamma; \Delta; \Omega \vdash \mathbf{snd}\, M : A$ *implies* $\Gamma; \Delta; \Omega \vdash M : A_1 \,\&\, A_2$ *and* $\Gamma \vdash A = A_2 : \texttt{type}$.

15. $\Gamma; \Delta; \Omega \vdash \langle M_1 \,,\, M_2 \rangle : A$ *implies*
$\Gamma \vdash A = A_1 \,\&\, A_2 : \texttt{type}$ *and* $\Gamma; \Delta; \Omega \vdash M_1 : A_1$ *and* $\Gamma; \Delta; \Omega \vdash M_2 : A_2$.

16. $\Gamma \vdash a : K$ *implies* $a{:}K' \in \Sigma$ *and* $\Gamma \vdash K = K' : \texttt{kind}$.

17. $\Gamma \vdash A\, M : K$ *implies*
$\Gamma \vdash A : \Pi x{:}A_1.\, K'$ *and* $\Gamma; \cdot; \cdot \vdash M : A_1$ *and* $\Gamma \vdash K = K'[M/x] : \texttt{kind}$.

18. $\Gamma \vdash \Pi x{:}A.\, B : K$ *implies*
$\Gamma \vdash K = \texttt{type} : \texttt{kind}$ *and* $\Gamma \vdash A : \texttt{type}$ *and* $\Gamma(x{:}A) \vdash B : \texttt{type}$.

19. $\Gamma \vdash A \multimap B : K$ *implies*
$\Gamma \vdash K = \texttt{type} : \texttt{kind}$ *and* $\Gamma \vdash A : \texttt{type}$ *and* $\Gamma \vdash B : \texttt{type}$.

20. $\Gamma \vdash A \twoheadrightarrow B : K$ *implies*
$\Gamma \vdash K = \texttt{type} : \texttt{kind}$ *and* $\Gamma \vdash A : \texttt{type}$ *and* $\Gamma \vdash B : \texttt{type}$.

21. $\Gamma \vdash A \rightarrowtail B : K$ *implies*
$\Gamma \vdash K = \texttt{type} : \texttt{kind}$ *and* $\Gamma \vdash A : \texttt{type}$ *and* $\Gamma \vdash B : \texttt{type}$.

22. $\Gamma \vdash A \,\&\, B : K$ *implies*
$\Gamma \vdash K = \texttt{type} : \texttt{kind}$ *and* $\Gamma \vdash A : \texttt{type}$ *and* $\Gamma \vdash B : \texttt{type}$.

23. $\Gamma \vdash \top : K$ *implies* $\Gamma \vdash K = \texttt{type} : \texttt{kind}$.

24. $\Gamma \vdash \Pi x{:}A.\ K : \texttt{kind}$ *implies* $\Gamma \vdash A : \texttt{type}$ *and* $\Gamma(x{:}A) \vdash K : \texttt{kind}$.

Proof: By induction on structure of given derivation using Lemma 63. □

Lemma 66 (Equality Inversion)

Assume $\vdash \Gamma\ \texttt{uctx}$.

1. $\Gamma \vdash K = \texttt{type} : \texttt{kind}$ *or* $\Gamma \vdash \texttt{type} = K : \texttt{kind}$
   implies $K = \texttt{type}$.

2. $\Gamma \vdash K = \Pi x{:}A_1.\ K_1 : \texttt{kind}$ *or* $\Gamma \vdash \Pi x{:}A_1.\ K_1 = K : \texttt{kind}$
   implies $K = \Pi x{:}A_2.\ K_2$ *and* $\Gamma \vdash A_1 = A_2 : \texttt{type}$ *and*
   $\Gamma(x{:}A_1) \vdash K_1 = K_2 : \texttt{kind}$.

3. $\Gamma \vdash A = \Pi x{:}B_1.\ B_2 : \texttt{type}$ *or* $\Gamma \vdash \Pi x{:}B_1.\ B_2 = A : \texttt{type}$
   implies $A = \Pi x{:}A_1.\ A_2$ *and* $\Gamma \vdash A_1 = B_1 : \texttt{type}$ *and*
   $\Gamma(x{:}A_1) \vdash A_2 = B_2 : \texttt{type}$.

4. $\Gamma \vdash A = B_1 \multimap B_2 : \texttt{type}$ *or* $\Gamma \vdash B_1 \multimap B_2 = A : \texttt{type}$
   implies $A = A_1 \multimap A_2$ *and* $\Gamma \vdash A_1 = B_1 : \texttt{type}$ *and* $\Gamma \vdash A_2 = B_2 : \texttt{type}$.

5. $\Gamma \vdash A = B_1 \twoheadrightarrow B_2 : \texttt{type}$ *or* $\Gamma \vdash B_1 \twoheadrightarrow B_2 = A : \texttt{type}$
   implies $A = A_1 \twoheadrightarrow A_2$ *and* $\Gamma \vdash A_1 = B_1 : \texttt{type}$ *and* $\Gamma \vdash A_2 = B_2 : \texttt{type}$.

6. $\Gamma \vdash A = B_1 \rightarrowtail B_2 : \texttt{type}$ *or* $\Gamma \vdash B_1 \rightarrowtail B_2 = A : \texttt{type}$
   implies $A = A_1 \rightarrowtail A_2$ *and* $\Gamma \vdash A_1 = B_1 : \texttt{type}$ *and* $\Gamma \vdash A_2 = B_2 : \texttt{type}$.

7. $\Gamma \vdash A = B_1 \,\&\, B_2 : \texttt{type}$ *or* $\Gamma \vdash B_1 \,\&\, B_2 = A : \texttt{type}$
   implies $A = A_1 \,\&\, A_2$ *and* $\Gamma \vdash A_1 = B_1 : \texttt{type}$ *and* $\Gamma \vdash A_2 = B_2 : \texttt{type}$.

Proof: By induction on structure of given derivation using Lemma 60. □

Lemma 67 (Injectivity)

1. $\Gamma \vdash \Pi x{:}A.\ K = \Pi x{:}A'.\ K' : \texttt{kind}$ *implies*
   $\Gamma \vdash A = A' : \texttt{type}$ *and* $\Gamma(x{:}A) \vdash K = K' : \texttt{kind}$.

2. $\Gamma \vdash \Pi x{:}A_1.\ A_2 = \Pi x{:}B_1.\ B_2 : \texttt{type}$ *implies*
   $\Gamma \vdash A_1 = B_1 : \texttt{type}$ *and* $\Gamma(x{:}A_1) \vdash A_2 = B_2 : \texttt{type}$.

3. $\Gamma \vdash A_1 \multimap A_2 = B_1 \multimap B_2 : \texttt{type}$ *implies*
$\Gamma \vdash A_1 = B_1 : \texttt{type}$ *and* $\Gamma \vdash A_2 = B_2 : \texttt{type}$.

4. $\Gamma \vdash A_1 \twoheadrightarrow A_2 = B_1 \twoheadrightarrow B_2 : \texttt{type}$ *implies*
$\Gamma \vdash A_1 = B_1 : \texttt{type}$ *and* $\Gamma \vdash A_2 = B_2 : \texttt{type}$.

5. $\Gamma \vdash A_1 \rightarrowtail A_2 = B_1 \rightarrowtail B_2 : \texttt{type}$ *implies*
$\Gamma \vdash A_1 = B_1 : \texttt{type}$ *and* $\Gamma \vdash A_2 = B_2 : \texttt{type}$.

6. $\Gamma \vdash A_1 \,\&\, A_2 = B_1 \,\&\, B_2 : \texttt{type}$ *implies*
$\Gamma \vdash A_1 = B_1 : \texttt{type}$ *and* $\Gamma \vdash A_2 = B_2 : \texttt{type}$.

Proof: Immediate from Lemma 66. □

At this point, we have stated all the properties we shall need to carry out our proof. The next chapter introduces a different notion for equality which is obviously decidable. We shall show that the two equalities coincide and thus prove decidablity of type-checking in general for OLF.

Chapter 14

Algorithmic Equality

In this chapter we present an equality algorithm which is sound and complete with respect to definitional equality of Section 13.3. We further show the decidability of this equality procedure in Chapter 15. The algorithm is identical to that given in [59] with a trivial extension to cover ordered types (which are treated identically to linear types). As explained in [24] the algorithm consists of two stages. To compare two objects at some arbitrary type, we first reduce the comparison to objects of base type by applying extensionality rules. Then, to compare objects at base type, we reduce each object to weak head normal form and compare heads. If the heads are equal we continue on to compare the corresponding arguments.

The algorithm is type-directed and thus requires types to be carried along. However, this complicates the treatment of dependent types. The solution proposed in [24] avoids such complications by using a simplified type structure which only contains information about the shape of the object (*e.g.*, it is a function, it is a pair); which is really all that is necessary to parametrically apply extensionality rules and transform a comparison at arbitrary type to one at base type. In practice this simply means changing Π families into (non-dependent) $\rightarrow$ types and erasing the object (family) dependencies from Π families (kinds).

The context splittings required by the definitional equality judgements further complicate the correctness of the algorithm. However, as noted in [59], the linearity constraints upon objects may be ignored for the purposes of equality checking (assuming we know beforehand that the objects being compared are well-typed). The same holds for the ordering constraints upon objects. Therefore our simplified type structure will treat all assumptions as unrestricted. However, in order to not complicate substitution, we maintain the syntactical distinctions among variable names.

Here is the formal specification of the simplified type system:

$$\begin{array}{rrcl}
\text{Simple Kinds} & \kappa & ::= & \mathtt{type}^- \mid \tau \rightarrow \kappa \\
\text{Simple Types} & \tau & ::= & \alpha \mid \\
 & & & \tau_1 \rightarrow \tau_2 \mid \tau_1 \multimap \tau_2 \mid \\
 & & & \tau_1 \twoheadrightarrow \tau_2 \mid \tau_1 \rightarrowtail \tau_2 \mid \\
 & & & \tau_1 \,\&\, \tau_2 \mid \top \\
\text{Simple Contexts} & \Psi & ::= & \cdot \mid \Psi(x{:}\tau) \mid \Psi(y{:}\tau) \mid \Psi(z{:}\tau)
\end{array}$$

Note that the simplified types are exactly the canonical types in Chapter 12.

We sometimes use Θ for simple contexts. Note that simplified contexts may contain variables of all three syntactic forms– however, as we shall see, simplified contexts treat all variables as unrestricted. We assume a simple base type a^- for each constant family a.

We now state an erasure function which simplifies OLF constructs.

$$\begin{array}{rcl}
(\mathtt{kind})^- & = & \mathtt{kind}^- \\
(\Pi x{:}A.\ K)^- & = & A^- \rightarrow K^- \\
(\mathtt{type})^- & = & \mathtt{type}^- \\
(a)^- & = & a^- \\
(A\,M)^- & = & A^- \\
(\Pi x{:}A_1.\ A_2)^- & = & A_1^- \rightarrow A_2^- \\
(A_1 \multimap A_2)^- & = & A_1^- \multimap A_2^- \\
(A_1 \twoheadrightarrow A_2)^- & = & A_1^- \twoheadrightarrow A_2^- \\
(A_1 \rightarrowtail A_2)^- & = & A_1^- \rightarrowtail A_2^- \\
(A_1 \,\&\, A_2)^- & = & A_1^- \,\&\, A_2^- \\
(\top)^- & = & \top \\
(\cdot)^- & = & \cdot \\
(\Gamma(x{:}A))^- & = & \Gamma^-(x{:}A^-) \\
(\Delta(y{:}A))^- & = & \Delta^-(y{:}A^-) \\
(\Omega(z{:}A))^- & = & \Omega^-(z{:}A^-)
\end{array}$$

Erasure remains invariant under equality and substitution.

Lemma 68 (Erasure Preservation)

1. $\Gamma \vdash A_1 = A_2 : K$ *implies* $A_1^- = A_2^-$.

2. $\Gamma \vdash K_1 = K_2 : \mathtt{kind}$ *implies* $K_1^- = K_2^-$.

3. $\Gamma(x{:}A) \vdash B : K$ *implies* $B^- = B^-[M/x]$.

4. $\Gamma(x{:}A) \vdash K : \mathtt{kind}$ *implies* $K^- = K^-[M/x]$.

Proof: By structural induction on the given derivation. □

Here are the judgements for the equality algorithm:

$M \xrightarrow{\mathtt{whr}} M'$	M weak head reduces to M'
$\Psi \vdash M \longleftrightarrow N : \tau$	M is structurally equal to N.
$\Psi \vdash M \Longleftrightarrow N : \tau$	M equals N at simple type τ.
$\Psi \vdash A_1 \longleftrightarrow A_2 : \kappa$	A_1 is structurally equal to A_2.
$\Psi \vdash A_1 \Longleftrightarrow A_2 : \kappa$	A_1 equals A_2 at simple kind κ.
$\Psi \vdash K_1 \Longleftrightarrow K_2 : \mathtt{kind}^-$	Kind K_1 equals kind K_2.

For weak head reduction, $M \xrightarrow{\mathtt{whr}} M'$ we assume M is given and compute M' or fail. For structural equality, $\Psi \vdash M \longleftrightarrow N : \tau$ we assume Ψ, M, and N are given and compute τ or fail. Algorithmic equality, $\Psi \vdash M \Longleftrightarrow N : \tau$, simply checks for equality and succeeds or fails. The same interpretations hold for the analogous judgements on families and kinds.

Weak Head Reduction

$$\frac{}{(\lambda x{:}A.\ M)\, N \xrightarrow{\mathtt{whr}} M[N/x]} \qquad \frac{M \xrightarrow{\mathtt{whr}} M'}{M\, N \xrightarrow{\mathtt{whr}} M'\, N}$$

$$\frac{}{(\hat{\lambda} y{:}A.\ M)\,\hat{}\, N \xrightarrow{\mathtt{whr}} M[N/y]} \qquad \frac{M \xrightarrow{\mathtt{whr}} M'}{M\,\hat{}\, N \xrightarrow{\mathtt{whr}} M'\,\hat{}\, N}$$

$$\frac{}{(\overset{<}{\lambda} z{:}A.\ M)^{<} N \xrightarrow{\mathtt{whr}} M[N/z]} \qquad \frac{M \xrightarrow{\mathtt{whr}} M'}{M^{<} N \xrightarrow{\mathtt{whr}} M'^{<} N}$$

$$\frac{}{(\overset{>}{\lambda} z{:}A.\ M)^{>} N \xrightarrow{\mathtt{whr}} M[N/z]} \qquad \frac{M \xrightarrow{\mathtt{whr}} M'}{M^{>} N \xrightarrow{\mathtt{whr}} M'^{>} N}$$

$$\frac{}{\mathbf{fst}\,\langle M\,,\, N\rangle \xrightarrow{\mathtt{whr}} M} \qquad \frac{M \xrightarrow{\mathtt{whr}} M'}{\langle M\,,\, N\rangle \xrightarrow{\mathtt{whr}} \langle M'\,,\, N\rangle}$$

$$\frac{}{\mathbf{snd}\,\langle M\,,\, N\rangle \xrightarrow{\mathtt{whr}} N} \qquad \frac{N \xrightarrow{\mathtt{whr}} N'}{\langle M\,,\, N\rangle \xrightarrow{\mathtt{whr}} \langle M\,,\, N'\rangle}$$

Structural Object Equality We use u to stand for any syntactic variable class, *i.e.*, x or y or z.

$$\frac{u{:}\tau \in \Psi}{\Psi \vdash u \longleftrightarrow u : \tau} \qquad \frac{c{:}\tau \in \Sigma}{\Psi \vdash c \longleftrightarrow c : A^{-}}$$

$$\frac{\Psi \vdash M_1 \longleftrightarrow N_1 : \tau_2 \to \tau_1 \qquad \Psi \vdash M_2 \Longleftrightarrow N_2 : \tau_2}{\Psi \vdash M_1\, M_2 \longleftrightarrow N_1\, N_2 : \tau_1}$$

$$\frac{\Psi \vdash M_1 \longleftrightarrow N_1 : \tau_2 \multimap \tau_1 \qquad \Psi \vdash M_2 \Longleftrightarrow N_2 : \tau_2}{\Psi \vdash M_1 \hat{}\, M_2 \longleftrightarrow N_1 \hat{}\, N_2 : \tau_1}$$

$$\frac{\Psi \vdash M_1 \longleftrightarrow N_1 : \tau_2 \twoheadrightarrow \tau_1 \qquad \Psi \vdash M_2 \Longleftrightarrow N_2 : \tau_2}{\Psi \vdash M_1{}^{>} M_2 \longleftrightarrow N_1{}^{>} N_2 : \tau_1}$$

$$\frac{\Psi \vdash M_1 \longleftrightarrow N_1 : \tau_2 \rightarrowtail \tau_1 \qquad \Psi \vdash M_2 \Longleftrightarrow N_2 : \tau_2}{\Psi \vdash M_1{}^{<} M_2 \longleftrightarrow N_1{}^{<} N_2 : \tau_1}$$

$$\frac{\Psi \vdash M \longleftrightarrow M' : \tau_1 \,\&\, \tau_2}{\Psi \vdash \mathbf{fst}\, M \longleftrightarrow \mathbf{fst}\, M' : \tau_1} \qquad \frac{\Psi \vdash M \longleftrightarrow M' : \tau_1 \,\&\, \tau_2}{\Psi \vdash \mathbf{snd}\, M \longleftrightarrow \mathbf{snd}\, M' : \tau_2}$$

Type-Directed Object Equality

$$\frac{M \xrightarrow{\mathtt{whr}} M' \qquad \Psi \vdash M' \Longleftrightarrow N : \alpha}{\Psi \vdash M \Longleftrightarrow N : \alpha} \qquad \frac{N \xrightarrow{\mathtt{whr}} N' \qquad \Psi \vdash M \Longleftrightarrow N' : \alpha}{\Psi \vdash M \Longleftrightarrow N : \alpha}$$

$$\frac{\Psi \vdash M \longleftrightarrow N : \alpha}{\Psi \vdash M \Longleftrightarrow N : \alpha}$$

$$\frac{\Psi(x{:}\tau_1) \vdash M\,x \Longleftrightarrow N\,x : \tau_2}{\Psi \vdash M \Longleftrightarrow N : \tau_1 \to \tau_2} \qquad \frac{\Psi(y{:}\tau_1) \vdash M\,\hat{}\,y \Longleftrightarrow N\,\hat{}\,y : \tau_2}{\Psi \vdash M \Longleftrightarrow N : \tau_1 \multimap \tau_2}$$

$$\frac{\Psi(z{:}\tau_1) \vdash M^{>} z \Longleftrightarrow N^{>} z : \tau_2}{\Psi \vdash M \Longleftrightarrow N : \tau_1 \twoheadrightarrow \tau_2} \qquad \frac{\Psi(z{:}\tau_1) \vdash M^{<} z \Longleftrightarrow N^{<} z : \tau_2}{\Psi \vdash M \Longleftrightarrow N : \tau_1 \rightarrowtail \tau_2}$$

$$\frac{}{\Psi \vdash M \Longleftrightarrow N : \top} \qquad \frac{\Psi \vdash \mathbf{fst}\, M \Longleftrightarrow \mathbf{fst}\, N : \tau_1 \qquad \Psi \vdash \mathbf{snd}\, M \Longleftrightarrow \mathbf{snd}\, N : \tau_2}{\Psi \vdash M \Longleftrightarrow N : \tau_1 \,\&\, \tau_2}$$

Structural Family Equality

$$\frac{a{:}K \in \Sigma}{\Psi \vdash a \longleftrightarrow a : K^{-}} \qquad \frac{\Psi \vdash A \longleftrightarrow B : \tau \to \kappa \qquad \Psi \vdash M \Longleftrightarrow N : \tau}{\Psi \vdash A\,M \longleftrightarrow B\,N : \kappa}$$

Kind-Directed Family Equality

$$\frac{\Psi \vdash A \longleftrightarrow B : \mathtt{type}^{-}}{\Psi \vdash A \Longleftrightarrow B : \mathtt{type}^{-}} \qquad \frac{\Psi(x{:}\tau) \vdash A\,x \Longleftrightarrow B\,x : \kappa}{\Psi \vdash A \Longleftrightarrow B : \tau \to \kappa}$$

$$\frac{\Psi \vdash A_1 \Longleftrightarrow B_1 : \mathtt{type}^{-} \qquad \Psi(x{:}A_1^{-}) \vdash A_2 \Longleftrightarrow B_2 : \mathtt{type}^{-}}{\Psi \vdash \Pi x{:}A_1.\, A_2 \Longleftrightarrow \Pi x{:}B_1.\, B_2 : \mathtt{type}^{-}}$$

$$\frac{\Psi \vdash A_1 \Longleftrightarrow B_1 : \mathtt{type}^{-} \qquad \Psi \vdash A_2 \Longleftrightarrow B_2 : \mathtt{type}^{-}}{\Psi \vdash A_1 \multimap A_2 \Longleftrightarrow B_1 \multimap B_2 : \mathtt{type}^{-}}$$

$$\frac{\Psi \vdash A_1 \Longleftrightarrow B_1 : \mathtt{type}^{-} \qquad \Psi \vdash A_2 \Longleftrightarrow B_2 : \mathtt{type}^{-}}{\Psi \vdash A_1 \twoheadrightarrow A_2 \Longleftrightarrow B_1 \twoheadrightarrow B_2 : \mathtt{type}^{-}}$$

$$\dfrac{\Psi \vdash A_1 \Longleftrightarrow B_1 : \mathtt{type}^- \qquad \Psi \vdash A_2 \Longleftrightarrow B_2 : \mathtt{type}^-}{\Psi \vdash A_1 \rightarrowtail A_2 \Longleftrightarrow B_1 \rightarrowtail B_2 : \mathtt{type}^-}$$

$$\dfrac{}{\Psi \vdash \top \Longleftrightarrow \top : \mathtt{type}^-} \qquad \dfrac{\Psi \vdash A_1 \Longleftrightarrow B_1 : \mathtt{type}^- \qquad \Psi \vdash A_2 \Longleftrightarrow B_2 : \mathtt{type}^-}{\Psi \vdash A_1 \mathbin{\&} A_2 \Longleftrightarrow B_1 \mathbin{\&} B_2 : \mathtt{type}^-}$$

Algorithmic Kind Equality

$$\dfrac{}{\Psi \vdash \mathtt{type}^- \Longleftrightarrow \mathtt{type}^- : \mathtt{kind}^-}$$

$$\dfrac{\Psi \vdash A \Longleftrightarrow B : \mathtt{type}^- \qquad \Psi(x{:}A^-) \vdash K \Longleftrightarrow K' : \mathtt{kind}^-}{\Psi \vdash \Pi x{:}A.\ K \Longleftrightarrow \Pi x{:}B.\ K' : \mathtt{kind}^-}$$

We now state some basic properties of our equality derivations.

Lemma 69 (Weakening for Algorithmic Equality)

For each algorithmic equality judgement J,
$\Psi\Psi' \vdash J$ *implies* $\Psi(x{:}\tau)\Psi' \vdash J$.

Proof: By structural induction on the given derivation. □

Lemma 70 (Determinacy of Algorithmic Equality)

1. $M \xrightarrow{\mathtt{whr}} M'$ *and* $M \xrightarrow{\mathtt{whr}} M''$ *implies* $M' = M''$.

2. $\Psi \vdash M \longleftrightarrow N : \tau$ *implies there is no* M' *s.t.* $M \xrightarrow{\mathtt{whr}} M'$.

3. $\Psi \vdash M \longleftrightarrow N : \tau$ *implies there is no* N' *s.t.* $N \xrightarrow{\mathtt{whr}} N'$.

4. $\Psi \vdash M \longleftrightarrow N : \tau$ *and* $\Psi \vdash M \longleftrightarrow N : \tau'$ *implies* $\tau = \tau'$.

5. $\Psi \vdash A \longleftrightarrow B : \kappa$ *and* $\Psi \vdash A \longleftrightarrow B : \kappa'$ *implies* $\kappa = \kappa'$.

Proof: By structural induction on the given derivation. Parts 2 and 3 may be proved by contradiction using inversion on the structural equality inference rules. □

Lemma 71 (Symmetry of Algorithmic Equality)

1. $\Psi \vdash M \Longleftrightarrow N : \tau$ *implies* $\Psi \vdash N \Longleftrightarrow M : \tau$.

2. $\Psi \vdash M \longleftrightarrow N : \tau$ *implies* $\Psi \vdash N \longleftrightarrow M : \tau$.

3. $\Psi \vdash A \Longleftrightarrow B : \kappa$ *implies* $\Psi \vdash B \Longleftrightarrow A : \kappa$.

4. $\Psi \vdash A \longleftrightarrow B : \kappa$ *implies* $\Psi \vdash B \longleftrightarrow A : \kappa$.

5. $\Psi \vdash K \Longleftrightarrow K' : \mathtt{kind}^-$ *implies* $\Psi \vdash K' \Longleftrightarrow K : \mathtt{kind}^-$.

Proof: By structural induction on the given derivations. □

Lemma 72 (Transitivity of Algorithmic Equality)

1. $\Psi \vdash M \Longleftrightarrow M' : \tau$ *and* $\Psi \vdash M' \Longleftrightarrow N : \tau$ *implies* $\Psi \vdash M \Longleftrightarrow N : \tau$.

2. $\Psi \vdash M \longleftrightarrow M' : \tau$ *and* $\Psi \vdash M' \longleftrightarrow N : \tau$ *implies* $\Psi \vdash M \longleftrightarrow N : \tau$.

3. $\Psi \vdash A \Longleftrightarrow A' : \kappa$ *and* $\Psi \vdash A' \Longleftrightarrow B : \kappa$ *implies* $\Psi \vdash A \Longleftrightarrow B : \kappa$.

4. $\Psi \vdash A \longleftrightarrow A' : \kappa$ *and* $\Psi \vdash A' \longleftrightarrow B : \kappa$ *implies* $\Psi \vdash A \longleftrightarrow B : \kappa$.

5. $\Psi \vdash K \Longleftrightarrow K' : \mathtt{kind}^-$ *and* $\Psi \vdash K' \Longleftrightarrow K'' : \mathtt{kind}^-$ *implies*
   $\Psi \vdash K \Longleftrightarrow K'' : \mathtt{kind}^-$.

Proof: By structural induction on the given derivations using Lemma 70. □

14.1 Completeness of Algorithmic Equality

In this section we show that algorithmic equality is complete with respect to definitional equality. The formal theorem we will show is:

$$\Gamma; \Delta; \Omega \vdash M = N : A \text{ implies } \Gamma^-\Delta^-\Omega^- \vdash M \Longleftrightarrow N : A^-$$

This is proved via logical relations defined on the simplified type of an object. Thus we will show:

1. $\Gamma; \Delta; \Omega \vdash M = N : A$ implies $\Gamma^-\Delta^-\Omega^- \vdash M = N \in [\![A^-]\!]$

2. $\Psi \vdash M = N \in \llbracket A^- \rrbracket$ implies $\Psi \vdash M \Longleftrightarrow N : A^-$

We use σ (and sometimes θ) to stand for substitutions.

$$\text{Substitutions} \qquad \sigma \;::=\; \cdot \mid \sigma, M/x \mid \sigma, M/y \mid \sigma, M/z$$

We assume as usual that no variable occurs more than once in a substitution. We further assume tacit variable renaming takes place when necessary to maintain this constraint. We denote the application of a substitution, σ, to a term, M, with $\sigma(M)$. We also assume parameterized identity substitutions, $\mathtt{id}_x$, on any context x.

We use the notation $\Psi' \geq \Psi$ to denote that Ψ' contains all declarations in Ψ and possibly more.

We define a Kripke logical relation inductively on simple types. As usual, we require that the property we wish to prove holds at base type. At higher types the property is inductively maintained.

A Kripke Logical Relation

1. $\Psi \vdash M = N \in \llbracket \alpha \rrbracket$ iff $\Psi \vdash M \Longleftrightarrow N : \alpha$.

2. $\Psi \vdash M = N \in \llbracket \tau' \to \tau \rrbracket$ iff $\forall \Psi', M', N'.$
   $\Psi' \geq \Psi$ and $\Psi' \vdash M' \Longleftrightarrow N' \in \llbracket \tau' \rrbracket$ implies $\Psi' \vdash M\,M' \Longleftrightarrow N\,N' \in \llbracket \tau \rrbracket$.

3. $\Psi \vdash M = N \in \llbracket \tau' \multimap \tau \rrbracket$ iff $\forall \Psi', M', N'.$
   $\Psi' \geq \Psi$ and $\Psi' \vdash M' \Longleftrightarrow N' \in \llbracket \tau' \rrbracket$ implies $\Psi' \vdash M\,\hat{}\,M' \Longleftrightarrow N\,\hat{}\,N' \in \llbracket \tau \rrbracket$.

4. $\Psi \vdash M = N \in \llbracket \tau' \twoheadrightarrow \tau \rrbracket$ iff $\forall \Psi', M', N'.$
   $\Psi' \geq \Psi$ and $\Psi' \vdash M' \Longleftrightarrow N' \in \llbracket \tau' \rrbracket$ implies $\Psi' \vdash M^{>}M' \Longleftrightarrow N^{>}N' \in \llbracket \tau \rrbracket$.

5. $\Psi \vdash M = N \in \llbracket \tau' \rightarrowtail \tau \rrbracket$ iff $\forall \Psi', M', N'.$
   $\Psi' \geq \Psi$ and $\Psi' \vdash M' \Longleftrightarrow N' \in \llbracket \tau' \rrbracket$ implies $\Psi' \vdash M^{<}M' \Longleftrightarrow N^{<}N' \in \llbracket \tau \rrbracket$.

6. $\Psi \vdash M = N \in \llbracket \tau_1 \,\&\, \tau_2 \rrbracket$ iff
   $\Psi \vdash \mathbf{fst}\, M = \mathbf{fst}\, N \in \llbracket \tau_1 \rrbracket$ and $\Psi \vdash \mathbf{snd}\, M = \mathbf{snd}\, N \in \llbracket \tau_2 \rrbracket$.

7. $\Psi \vdash M = N \in \llbracket \top \rrbracket$.

8. $\Psi \vdash A = B \in \llbracket \mathtt{type}^- \rrbracket$ iff $\Psi \vdash A \Longleftrightarrow B : \mathtt{type}^-$.

9. $\Psi \vdash A = B \in [\![\tau \to \kappa]\!]$ iff $\forall \Psi', M, N.$
   $\Psi' \geq \Psi$ and $\Psi' \vdash M \Longleftrightarrow N \in [\![\tau]\!]$ implies $\Psi' \vdash A\,M \Longleftrightarrow B\,N \in [\![\kappa]\!]$.

10. $\Psi \vdash \sigma = \theta \in [\![\cdot]\!]$ iff $\sigma = \cdot = \theta$.

11. $\Psi \vdash \sigma = \theta \in [\![\Theta, u{:}\tau]\!]$ iff
    $\sigma = (\sigma', M/u)$ and $\theta = (\theta', N/u)$ and
    $\Psi \vdash M = N \in [\![\tau]\!]$ and $\Psi \vdash \sigma' = \theta' \in [\![\Theta]\!]$.

Lemma 73 (Weakening for Logical Relations)

For each logical relation R, $\Psi\Psi' \vdash R$ *implies* $\Psi(x{:}\tau)\Psi' \vdash R$.

Proof: By induction on the type or kind. □

We may now show the second part of our proof.

Theorem 74 (Logically Related Terms are Algorithmically Equal)

1. $\Psi \vdash M = N \in [\![\tau]\!]$ *implies* $\Psi \vdash M \Longleftrightarrow N : \tau$.

2. $\Psi \vdash A = B \in [\![\kappa]\!]$ *implies* $\Psi \vdash A \Longleftrightarrow B : \kappa$.

3. $\Psi \vdash M \longleftrightarrow N : \tau$ *implies* $\Psi \vdash M = N \in [\![\tau]\!]$.

4. $\Psi \vdash A \longleftrightarrow B : \kappa$ *implies* $\Psi \vdash A = B \in [\![\kappa]\!]$.

Proof: By induction on the structure of τ or κ. □

The first part of our proof requires a few extra lemmas.

Lemma 75 (Closure Under Head Expansion)

1. $M \xrightarrow{\texttt{whr}} M'$ *and* $\Psi \vdash M' = N \in [\![\tau]\!]$ *implies* $\Psi \vdash M = N \in [\![\tau]\!]$.

2. $N \xrightarrow{\texttt{whr}} N'$ *and* $\Psi \vdash M = N' \in [\![\tau]\!]$ *implies* $\Psi \vdash M = N \in [\![\tau]\!]$.

Proof: By induction on the structure of τ. □

Lemma 76 (Symmetry of Logical Relation)

1. $\Psi \vdash M = N \in [\![\tau]\!]$ *implies* $\Psi \vdash N = M \in [\![\tau]\!]$.

2. $\Psi \vdash A = B \in [\![\kappa]\!]$ *implies* $\Psi \vdash B = A \in [\![\kappa]\!]$.

3. $\Psi \vdash \sigma = \theta \in [\![\Theta]\!]$ *implies* $\Psi \vdash \theta = \sigma \in [\![\Theta]\!]$.

Proof: By induction on the structure of τ, κ, and Θ using Lemma 71. □

Lemma 77 (Transitivity of Logical Relation)

1. $\Psi \vdash M = M' \in [\![\tau]\!]$ *and* $\Psi \vdash M' = N \in [\![\tau]\!]$ *implies* $\Psi \vdash M = N \in [\![\tau]\!]$.

2. $\Psi \vdash A = A' \in [\![\kappa]\!]$ *and* $\Psi \vdash A' = B \in [\![\kappa]\!]$ *implies* $\Psi \vdash A = B \in [\![\kappa]\!]$.

3. $\Psi \vdash \sigma = \sigma' \in [\![\Theta]\!]$ *and* $\Psi \vdash \sigma' = \theta \in [\![\Theta]\!]$ *implies* $\Psi \vdash \sigma = \theta \in [\![\Theta]\!]$.

Proof: By induction on the structure of τ, κ, and Θ using Lemmas 72. □

We may now prove a generalization of the first part of our proof.

Lemma 78

(Definitionally Equal Terms are Logically Related Under Substitutions)

1. $\Gamma; \Delta; \Omega \vdash M = N : A$ *and* $\Psi \vdash \sigma = \theta \in [\![\Psi']\!]$ *and* $\Psi' \geq \Gamma^- \Delta^- \Omega^-$ *implies*
   $\Psi \vdash \sigma(M) = \theta(N) \in [\![A^-]\!]$.

2. $\Gamma \vdash A = B : K$ *and* $\Psi \vdash \sigma = \theta \in [\![\Psi']\!]$ *and* $\Psi' \geq \Gamma^-$ *implies*
   $\Psi \vdash \sigma(A) = \theta(B) \in [\![K^-]\!]$.

Proof: By induction on the given derivation of definitional equality using the prior results in this section. □

It is now easy to finish off the proof of completeness of algorithmic equality.

Lemma 79 (Identity Substitutions are Logically Related)

$\Psi \vdash \mathtt{id}_\Psi = \mathtt{id}_\Psi \in [\![\Psi]\!]$.

Proof: By definition of $[\![\Psi]\!]$ and Theorem 74 □

Theorem 80 (Definitionally Equal Terms are Logically Related)

1. $\Gamma; \Delta; \Omega \vdash M = N : A$ *implies* $\Gamma^- \Delta^- \Omega^- \vdash M = N \in [\![A^-]\!]$.

2. $\Gamma \vdash A = B : K$ *implies* $\Gamma^- \vdash A = B \in [\![K^-]\!]$.

Proof: By Lemmas 78 and 79. □

Theorem 81 (Completeness of Algorithmic Equality)

1. $\Gamma; \Delta; \Omega \vdash M = N : A$ *implies* $\Gamma^- \Delta^- \Omega^- \vdash M \Longleftrightarrow N : A^-$.

2. $\Gamma \vdash A = B : K$ *implies* $\Gamma^- \vdash A \Longleftrightarrow B : K^-$.

Proof: By Theorems 80 and 74. □

14.2 Soundness of Algorithmic Equality

In this section we show the soundness of algorithmic equality with respect to definitional equality. In general this does not hold since the algorithmic equality does not force the objects being compared to be well-typed, or even have the same types. However, if we restrict attention to well-typed objects of the same type, then we get a soundness result.

A further difficulty lies in the simplified type structure, used by the algorithmic equality, which ignores linearity and ordering constraints. We must therefore rely upon the objects' typing derivations to recover hypothesis management, required by definitional equality derivations, satisfying linearity and ordering constraints. However, due to the presence of $\top$, two typing derivations for equal objects can disagree about which linear and ordered hypotheses are consumed in a derivation branch. The solution adopted in [59] gets around this problem by constructing a *mediating* object, for algorithmically equal objects, which is definitionally equal to both given objects.

Before proceeding to the main lemma, we need to show the following result.

Lemma 82 (Subject Reduction)

$M \xrightarrow{\texttt{whr}} M'$ *and* $\Gamma; \Delta; \Omega \vdash M : A$ *implies* $\Gamma; \Delta; \Omega \vdash M = M' : A$.

Proof: By induction on the given weak head reduction derivation making use of Lemmas 62, 67 and 59. □

We may now prove the following lemma which directly implies the result we want.

Lemma 83 (Algorithmically Equal Well-Formed Terms Have Mediating Terms)

Assume $\vdash \Gamma\ \texttt{uctx}$ *and* $\Gamma \vdash \Delta_1\ \texttt{lctx}$ *and* $\Gamma \vdash \Delta_2\ \texttt{lctx}$ *and* $\Gamma \vdash \Omega_1\ \texttt{lctx}$ *and* $\Gamma \vdash \Omega_2\ \texttt{octx}$.

1. $\Gamma;\Delta_1;\Omega_1 \vdash M : A$ *and* $\Gamma;\Delta_2;\Omega_2 \vdash N : A$ *and* $\Psi \vdash M \Longleftrightarrow N : A^-$ *and* $\Psi \geq \Gamma^-\Delta_1^-\Omega_1^-$ *and* $\Psi \geq \Gamma^-\Delta_2^-\Omega_2^-$ *implies*
   $\exists P.\ \Gamma;\Delta_1;\Omega_1 \vdash P = M : A$ *and* $\Gamma;\Delta_2;\Omega_2 \vdash P = N : A$.

2. $\Gamma;\Delta_1;\Omega_1 \vdash M : A$ *and* $\Gamma;\Delta_2;\Omega_2 \vdash N : B$ *and* $\Psi \vdash M \longleftrightarrow N : \tau$ $\Psi \geq \Gamma^-\Delta_1^-\Omega_1^-$ *and* $\Psi \geq \Gamma^-\Delta_2^-\Omega_2^-$ *implies*
   $\exists C, P.\ \Gamma \vdash C = A : \texttt{type}$ *and* $\Gamma \vdash C = B : \texttt{type}$
   and $\Gamma;\Delta_1;\Omega_1 \vdash P = M : C$ *and* $\Gamma;\Delta_2;\Omega_2 \vdash P = N : C$
   and $A^- = B^- = C^- = \tau$.

3. $\Gamma \vdash A : K$ *and* $\Gamma \vdash B : K$ *and* $\Gamma^- \vdash A \Longleftrightarrow B : K^-$ *implies* $\Gamma \vdash A = B : K$.

4. $\Gamma \vdash A_1 : K_1$ *and* $\Gamma \vdash A_2 : K_2$ *and* $\Gamma^- \vdash A_1 \longleftrightarrow A_2 : \kappa$ *implies*
   $\Gamma \vdash A_1 = A_2 : K_1$ *and* $\Gamma \vdash K_1 = K_2 : \texttt{kind}$ *and* $K_1^- = K_2^- = \kappa$.

5. $\Gamma \vdash K : \texttt{kind}$ *and* $\Gamma \vdash K' : \texttt{kind}$ *and* $\Gamma^- \vdash K \longleftrightarrow K' : \texttt{kind}^-$
   implies $\Gamma \vdash K = K' : \texttt{kind}$.

Proof: By induction on the given algorithmic equality derivation using inversion and injectivity properties. □

Theorem 84 (Soundness of Algorithmic Equality)

Assume $\vdash \Gamma\ \texttt{uctx}$ *and* $\Gamma \vdash \Delta\ \texttt{lctx}$ *and* $\Gamma \vdash \Omega\ \texttt{octx}$.

1. $\Gamma;\Delta;\Omega \vdash M : A$ *and* $\Gamma;\Delta;\Omega \vdash N : A$ *and* $\Gamma^-\Delta^-\Omega^- \vdash M \Longleftrightarrow N : A^-$
   implies $\Gamma;\Delta;\Omega \vdash M = N : A$.

2. $\Gamma \vdash A : K$ *and* $\Gamma \vdash B : K$ *and* $\Gamma^- \vdash A \Longleftrightarrow B : K^-$ *implies* $\Gamma \vdash A = B : K$.

Proof: By Lemma 83, symmetry and transitivity. □

Now that we have shown the correctness of algorithmic equality, with respect to definitional equality, we are ready to show decidability of OLF type-checking.

Chapter 15

Decidability and Canonical Forms

In this chapter we show that the type checking for OLF is decidable and that there exists a notion of canonical form in OLF suitable for logical framework representations. We first show that algorithmic equality is decidable. We then use the correctness of algorithmic equality, Theorems 81 and 84, to show decidability of type-checking.

The notion of canonical form we will define will come from the algorithmic equality derivations. Specifically, we will make use of the fact that the mediating terms mentioned in Lemma 83 are unique up to the type labels on bound variables. We will define a canonical representation for each equivalence class of mediating terms which will serve as a representative for a given type.

15.1 Decidability of Equality

An object is *normalizing* if it is algorithmically equal to some other object.

Lemma 85 (Decidability of Normalizing Terms)

1. $\Psi \vdash M \Longleftrightarrow M' : \tau$ *and* $\Psi \vdash N \Longleftrightarrow N' : \tau$ *implies*
it is decidable whether $\Psi \vdash M \Longleftrightarrow N : \tau$.

2. $\Psi \vdash M \longleftrightarrow M' : \tau_1$ *and* $\Psi \vdash N \longleftrightarrow N' : \tau_2$ *implies*
it is decidable whether $\Psi \vdash M \longleftrightarrow N : \tau_3$ *for some* τ_3.

3. $\Psi \vdash A \Longleftrightarrow A' : \kappa$ *and* $\Psi \vdash B \Longleftrightarrow B' : \kappa$ *implies*
it is decidable whether $\Psi \vdash A \Longleftrightarrow B : \kappa$.

4. $\Psi \vdash A \longleftrightarrow A' : \kappa_1$ *and* $\Psi \vdash B \longleftrightarrow B' : \kappa_2$ *implies*
   it is decidable whether $\Psi \vdash A \longleftrightarrow B : \kappa_3$ *for some* κ_3.

5. $\Psi \vdash K_1 \Longleftrightarrow K_1' : \mathtt{kind}^-$ *and* $\Psi \vdash K_2 \Longleftrightarrow K_2' : \mathtt{kind}^-$ *implies*
   it is decidable whether $\Psi \vdash K_1 \Longleftrightarrow K_2 : \mathtt{kind}^-$.

Proof: By induction on the structure of the given derivation using Lemma 70. □

Lemma 86 (Decidability of Algorithmic Equality)

1. $\Gamma; \Delta; \Omega \vdash M : A$ *and* $\Gamma; \Delta; \Omega \vdash N : A$ *implies*
   it is decidable whether $\Gamma^- \Delta^- \Omega^- \vdash M \Longleftrightarrow N : A^-$.

2. $\Gamma \vdash A : K$ *and* $\Gamma \vdash B : K$ *implies*
   it is decidable whether $\Gamma^- \vdash A \Longleftrightarrow B : K^-$.

3. $\Gamma \vdash K : \mathtt{kind}^-$ *and* $\Gamma \vdash K' : \mathtt{kind}^-$ *implies*
   it is decidable whether $\Gamma^- \vdash K \Longleftrightarrow K' : \mathtt{kind}^-$.

Proof: By Lemma 58 and Theorem 81 both M and N are normalizing. Then, by Lemma 85 we are done. □

Theorem 87 (Decidability of Definitional Equality)

$\Gamma; \Delta; \Omega \vdash M : A$ *and* $\Gamma; \Delta; \Omega \vdash N : A$ *implies*
it is decidable whether $\Gamma; \Delta; \Omega \vdash M = N : A$.

$\Gamma \vdash A : K$ *and* $\Gamma \vdash B : K$ *implies*
it is decidable whether $\Gamma \vdash A = B : K$.

$\Gamma \vdash K : \mathtt{kind}$ *and* $\Gamma \vdash K' : \mathtt{kind}$ *implies*
it is decidable whether $\Gamma \vdash K = K' : \mathtt{kind}$.

Proof: By Theorems 81, 84 and Lemma 86. □

15.2 Decidability of OLF Type-Checking

We now present a decidable version of the typing rules from Section 13.2 which is essentially a trivial extension of the algorithmic type-checking rules given in [24]. These rules use algorithmic equality, rather than definitional equality, and should be used in a bottom-up fashion.

We introduce the following new judgements for algorithmic type-checking:

$$\begin{array}{ll} \Gamma \vdash K \Rightarrow \texttt{kind} & K \text{ is a valid kind} \\ \Gamma \vdash A \Rightarrow K & A \text{ has kind } K \\ \Gamma; \Delta; \Omega \vdash M \Rightarrow A & M \text{ has type } A \end{array}$$

We may operationally think of everything to the left of ⇒ as input and the family, or kind, to the right of ⇒ as output.

Kinds

$$\frac{}{\Gamma \vdash \texttt{type} \Rightarrow \texttt{kind}} \qquad \frac{\Gamma \vdash A \Rightarrow \texttt{type} \qquad \Gamma(x{:}A) \vdash K \Rightarrow \texttt{kind}}{\Gamma \vdash \Pi x{:}A.\ K \Rightarrow \texttt{kind}}$$

Families

$$\frac{a{:}K \in \Sigma}{\Gamma \vdash a \Rightarrow K} \qquad \frac{\Gamma \vdash A \Rightarrow \Pi x{:}B.\ K \qquad \Gamma \vdash M \Rightarrow B' \qquad \Gamma^- \vdash B \Longleftrightarrow B' : \texttt{type}}{\Gamma \vdash A\, M \Rightarrow K[M/x]}$$

$$\frac{\Gamma \vdash A_1 \Rightarrow \texttt{type} \qquad \Gamma(x{:}A_1) \vdash A_2 \Rightarrow \texttt{type}}{\Gamma \vdash \Pi x{:}A_1.\ A_2 \Rightarrow \texttt{type}}$$

Objects

$$\frac{c{:}A \in \Sigma}{\Gamma; \cdot; \cdot \vdash c \Rightarrow A} \qquad \frac{x{:}A \in \Gamma}{\Gamma; \cdot; \cdot \vdash x \Rightarrow A} \qquad \frac{}{\Gamma; y{:}A; \cdot \vdash y \Rightarrow A} \qquad \frac{}{\Gamma; \cdot; z{:}A \vdash z \Rightarrow A}$$

$$\frac{\Gamma \vdash A' \Rightarrow \texttt{type} \qquad \Gamma(x{:}A'); \Delta; \Omega \vdash M \Rightarrow A}{\Gamma; \Delta; \Omega \vdash \lambda x{:}A'.\ M \Rightarrow \Pi x{:}A'.\ A}$$

$$\frac{\Gamma; \Delta; \Omega \vdash M \Rightarrow \Pi x{:}B.\ A \qquad \Gamma; \cdot; \cdot \vdash M' \Rightarrow B' \qquad \Gamma^- \vdash B \Longleftrightarrow B' : \texttt{type}}{\Gamma; \Delta; \Omega \vdash M\, M' \Rightarrow A[M'/x]}$$

$$\frac{\Gamma \vdash A' \Rightarrow \texttt{type} \qquad \Gamma; \Delta(y{:}A'); \Omega \vdash M \Rightarrow A}{\Gamma; \Delta; \Omega \vdash \hat{\lambda} y{:}A'.\ M \Rightarrow A' \multimap A}$$

$$\frac{\Gamma; \Delta; \Omega \vdash M \Rightarrow B \multimap A \qquad \Gamma; \Delta'; \cdot \vdash M' \Rightarrow B' \qquad \Gamma^{-} \vdash B \Longleftrightarrow B' : \texttt{type}}{\Gamma; \Delta \bowtie \Delta'; \Omega \vdash M \,\hat{}\, M' \Rightarrow A}$$

$$\frac{\Gamma \vdash A' \Rightarrow \texttt{type} \qquad \Gamma; \Delta; \Omega(z{:}A') \vdash M \Rightarrow A}{\Gamma; \Delta; \Omega \vdash \overset{>}{\lambda} z{:}A'.\ M \Rightarrow A' \twoheadrightarrow A}$$

$$\frac{\Gamma; \Delta; \Omega \vdash M \Rightarrow B \twoheadrightarrow A \qquad \Gamma; \Delta'; \Omega'; \cdot \vdash M' \Rightarrow B' \qquad \Gamma^{-} \vdash B \Longleftrightarrow B' : \texttt{type}}{\Gamma; \Delta \bowtie \Delta'; \Omega\Omega' \vdash M \,{}^{>}\, M' \Rightarrow A}$$

$$\frac{\Gamma \vdash A' \Rightarrow \texttt{type} \qquad \Gamma; \Delta; (z{:}A')\Omega \vdash M \Rightarrow A}{\Gamma; \Delta; \Omega \vdash \overset{<}{\lambda} z{:}A'.\ M \Rightarrow A' \rightarrowtail A}$$

$$\frac{\Gamma; \Delta; \Omega \vdash M \Rightarrow B \rightarrowtail A \qquad \Gamma; \Delta'; \Omega'; \cdot \vdash M' \Rightarrow B' \qquad \Gamma^{-} \vdash B \Longleftrightarrow B' : \texttt{type}}{\Gamma; \Delta \bowtie \Delta'; \Omega'\Omega \vdash M \,{}^{<}\, M' \Rightarrow A}$$

$$\frac{}{\Gamma; \Delta; \Omega \vdash \langle\rangle \Rightarrow \top} \qquad \frac{\Gamma; \Delta; \Omega \vdash M_1 \Rightarrow A_1 \qquad \Gamma; \Delta; \Omega \vdash M_2 \Rightarrow A_2}{\Gamma; \Delta; \Omega \vdash \langle M_1\, ,\, M_2\rangle \Rightarrow A_1 \,\&\, A_2}$$

$$\frac{\Gamma; \Delta; \Omega \vdash M \Rightarrow A_1 \,\&\, A_2}{\Gamma; \Delta; \Omega \vdash \mathbf{fst}\, M \Rightarrow A_1} \qquad \frac{\Gamma; \Delta; \Omega \vdash M \Rightarrow A_1 \,\&\, A_2}{\Gamma; \Delta; \Omega \vdash \mathbf{snd}\, M \Rightarrow A_2}$$

$$\frac{\Gamma; \Delta; \Omega \vdash M \Rightarrow A \qquad \Gamma \vdash A = A' \Rightarrow \texttt{type}}{\Gamma; \Delta; \Omega \vdash M \Rightarrow A'}$$

Note that these typing rules are completely syntax-directed; each form of object appears in the conclusion of only one rule, whose premises contain only subcomponents of the object; and similarly for families and kinds.

We also assume new algorithmic type-checking based judgements to check validity for signatures and contexts.

Lemma 88 (Correctness of Algorithmic Type-Checking)

1. $\Gamma; \Delta; \Omega \vdash M \Rightarrow A$ *implies* $\Gamma; \Delta; \Omega \vdash M : A$.

2. $\Gamma; \Delta; \Omega \vdash M : A$ *implies*
   $\exists A'.\ \Gamma; \Delta; \Omega \vdash A = A' : \mathtt{type}$ *and* $\Gamma; \Delta; \Omega \vdash M \Rightarrow A'$.

Proof: Part 1 is proved by induction on the structure of the given derivation using Lemma 63, Theorem 84, and the type conversion rule.

Part 2 is proved by induction on the structure of the given derivation using transitivity definitional equality rules, Lemma 66 and Theorem 81. □

Theorem 89 (Decidability of Type-Checking)

1. $\vdash \Gamma\ \mathtt{uctx}$ *is decidable.*

2. $\vdash \Gamma\ \mathtt{uctx}$ *implies* $\Gamma \vdash \Delta\ \mathtt{lctx}$ *and* $\Gamma \vdash \Omega\ \mathtt{octx}$ *are decidable.*

3. $\vdash \Gamma\ \mathtt{uctx}$ *and* $\Gamma \vdash \Delta\ \mathtt{lctx}$ *and* $\Gamma \vdash \Omega\ \mathtt{octx}$
   implies $\Gamma; \Delta; \Omega \vdash M : A$ *is decidable.*

4. $\vdash \Gamma\ \mathtt{uctx}$ *implies* $\Gamma \vdash A : K$ *is decidable.*

5. $\vdash \Gamma\ \mathtt{uctx}$ *implies* $\Gamma \vdash K : \mathtt{kind}$ *is decidable.*

Proof: There exists at most one A' such that $\Gamma; \Delta; \Omega \vdash M \Rightarrow A'$ since the algorithmic typing rules are syntax-directed and algorithmic equality is decidable. Then by Lemma 88, $\Gamma; \Delta; \Omega \vdash M : A$ iff $\Gamma \vdash A = A' : \mathtt{type}$ which can be decided by checking $\Gamma^- \vdash A \Longleftrightarrow A' : \mathtt{type}^-$. □

15.3 Canonical Forms

We now turn our attention to identifying canonical forms for OLF.

We will start off by instrumenting the algorithmic equality judgements to produce the mediating object for two equal objects. These mediating objects will be unique. However, since algorithmic equality uses simplified types, we will not be able to construct an actual OLF object (we can recover the type-labels later from the actual type of the terms).

Rather we will construct an OLF object without the type labels on bound variables. This object will be unique. We will call such objects *quasi objects* and define them as follows:

$$
\begin{array}{lll}
\text{Quasi objects} & Q & ::= x \mid y \mid z \mid \\
 & & \quad \lambda x.\, Q \mid Q_1\, Q_2 \mid \\
 & & \quad \hat{\lambda} y.\, Q \mid Q_1 \,\hat{}\, Q_2 \mid \\
 & & \quad \overset{>}{\lambda} z.\, Q \mid Q_1{}^{>} Q_2 \mid \\
 & & \quad \overset{<}{\lambda} z.\, Q \mid Q_1{}^{<} Q_2 \mid \\
 & & \quad \langle Q_1\,,\, Q_2 \rangle \mid \mathbf{fst}\, Q \mid \mathbf{snd}\, Q \mid \langle\rangle
\end{array}
$$

We can now define the notions of *quasi-canonical* and *quasi-atomic* form, with the usual two mutually recursive judgments:

$$
\begin{array}{ll}
\Gamma; \Delta; \Omega \vdash Q \Uparrow A & Q \text{ is quasi-canonical at type } A \\
\Gamma; \Delta; \Omega \vdash Q \downarrow A & Q \text{ is quasi-atomic at type } A
\end{array}
$$

The derivation rules for these judgements are as follows:

Variables

$$
\dfrac{}{\Gamma_1(x{:}A)\Gamma_2; \cdot; \cdot \vdash x \downarrow A}\,\mathbf{uvar}
$$

$$
\dfrac{}{\Gamma; y{:}A; \cdot \vdash y \downarrow A}\,\mathbf{lvar}
$$

$$
\dfrac{}{\Gamma; \cdot; z{:}A \vdash z \downarrow A}\,\mathbf{ovar}
$$

Atomic Types.

$$
\dfrac{\Gamma; \Delta; \Omega \vdash Q \downarrow a' \qquad \Gamma \vdash a' = a : \mathtt{type}}{\Gamma; \Delta; \Omega \vdash Q \Uparrow a}\,\mathbf{coercion}
$$

Unrestricted Functions.

$$\frac{\Gamma(x{:}A);\Delta;\Omega \vdash Q' \Uparrow B}{\Gamma;\Delta;\Omega \vdash \lambda x.\ Q' \Uparrow \Pi x{:}A.\ B}\,\Pi I$$

$$\frac{\Gamma;\Delta;\Omega \vdash Q \downarrow \Pi x{:}A.\ B \qquad \Gamma;\cdot;\cdot \vdash Q' \Uparrow A}{\Gamma;\Delta;\Omega \vdash Q\,Q' \downarrow B[(Q')^{\natural}_{A}/x]}\,\Pi E$$

Linear Functions.

$$\frac{\Gamma;\Delta(y{:}A);\Omega \vdash Q' \Uparrow B}{\Gamma;\Delta;\Omega \vdash \hat{\lambda} y.\ Q' \Uparrow A \multimap B}\,{\multimap}I$$

$$\frac{\Gamma;\Delta;\Omega \vdash Q \downarrow A \multimap B \qquad \Gamma;\Delta';\cdot \vdash Q' \Uparrow A}{\Gamma;\Delta \bowtie \Delta';\Omega \vdash Q\,\hat{}\,Q' \downarrow B}\,{\multimap}E$$

Right Ordered Functions.

$$\frac{\Gamma;\Delta;\Omega(z{:}A) \vdash Q' \Uparrow B}{\Gamma;\Delta;\Omega \vdash \overset{>}{\lambda} z.\ Q' \Uparrow A \twoheadrightarrow B}\,{\twoheadrightarrow}I$$

$$\frac{\Gamma;\Delta;\Omega \vdash Q \downarrow A \twoheadrightarrow B \qquad \Gamma;\Delta';\Omega' \vdash Q' \Uparrow A}{\Gamma;\Delta \bowtie \Delta';\Omega\Omega' \vdash Q^{>}Q' \downarrow B}\,{\twoheadrightarrow}E$$

Left Ordered Functions.

$$\frac{\Gamma;\Delta;(z{:}A)\Omega \vdash Q' \Uparrow B}{\Gamma;\Delta;\Omega \vdash \overset{<}{\lambda} z.\ Q' \Uparrow A \rightarrowtail B}\,{\rightarrowtail}I$$

$$\frac{\Gamma;\Delta;\Omega \vdash Q \downarrow A \rightarrowtail B \qquad \Gamma;\Delta';\Omega' \vdash Q' \Uparrow A}{\Gamma;\Delta \bowtie \Delta';\Omega'\Omega \vdash Q^{<}Q' \downarrow B}\,{\rightarrowtail}E$$

Additive Pairs.

$$\frac{\Gamma;\Delta;\Omega \vdash Q_1 \Uparrow A \qquad \Gamma;\Delta;\Omega \vdash Q_2 \Uparrow B}{\Gamma;\Delta;\Omega \vdash \langle Q_1\,,\,Q_2\rangle \Uparrow A \,\&\, B}\,\&I$$

$$\frac{\Gamma;\Delta;\Omega \vdash Q \downarrow A \,\&\, B}{\Gamma;\Delta;\Omega \vdash \mathbf{fst}\,Q \downarrow A}\,\&E1 \qquad \frac{\Gamma;\Delta;\Omega \vdash Q \downarrow A \,\&\, B}{\Gamma;\Delta;\Omega \vdash \mathbf{snd}\,Q \downarrow B}\,\&E2$$

Additive Unit.

$$\frac{}{\Gamma;\Delta;\Omega \vdash \langle\rangle \Uparrow \top}\,\top I$$

Given a derivation of $\Gamma;\Delta;\Omega \vdash Q \Uparrow A$, one may produce an actual OLF object by adding type labels to Q. We use the notation $(Q)^\sharp_A$ to denote this operation. Note that this operation is compositional, *i.e.*, $(Q[Q'/x])^\sharp_A = (Q)^\sharp_A[(Q')^\sharp_{A'}/x]$ where A is the (given) type of the whole term, and A' is the type of variable x in context.

We make use of the following properties of quasi-canonical forms.

Lemma 90

1. $\Gamma_1\Gamma_2;\Delta;\Omega \vdash Q \Uparrow A$ *implies* $\Gamma_1\Gamma\Gamma_2;\Delta;\Omega \vdash Q \Uparrow A$.
2. $\Gamma;\Delta;\Omega_1(z{:}A')\Omega_2 \vdash Q \Uparrow A$ *implies* $\Gamma(x{:}A');\Delta;\Omega_1\Omega_2 \vdash Q[x/z] \Uparrow A$.
3. $\Gamma;\Delta_1(y{:}A')\Delta_2;\Omega \vdash Q \Uparrow A$ *implies* $\Gamma(x{:}A');\Delta_1\Delta_2;\Omega \vdash Q[x/z] \Uparrow A$.

Proof: By structural induction on the given derivation. □

Note that quasi-canonical objects are simply objects in $\beta\eta$-long form which are missing type labels on bound variables.

Theorem 91
Assume $\Gamma \vdash A : \texttt{type}$.

1. $\Gamma;\Delta;\Omega \vdash Q \Uparrow A$ *implies* $\Gamma;\Delta;\Omega \vdash (Q)^\sharp_A : A$.
2. $\Gamma;\Delta;\Omega \vdash Q \downarrow A$ *implies* $\Gamma;\Delta;\Omega \vdash (Q)^\sharp_A : A$.

Proof: By structural induction on the given derivation. □

We use the notation $M^\flat$ for the result of erasing type labels from an OLF object M to get a quasi object. Note that this operation is compositional, *i.e.*, $(M[M'/x])^\flat = (M)^\flat[(M')^\flat/x]$.

We now restate Lemma 83 to explicitly characterize the form of the mediating terms using the quasi canonical judgements.

Lemma 92 (Almost Canonical Forms)

Assume $\vdash \Gamma\ \texttt{uctx}$ *and* $\Gamma \vdash \Delta_1\ \texttt{lctx}$ *and* $\Gamma \vdash \Delta_2\ \texttt{lctx}$ *and* $\Gamma \vdash \Omega_1\ \texttt{lctx}$ *and* $\Gamma \vdash \Omega_2\ \texttt{octx}$.

1. $\Gamma; \Delta_1; \Omega_1 \vdash M : A$ *and* $\Gamma; \Delta_2; \Omega_2 \vdash N : A$ *and* $\Psi \vdash M \Longleftrightarrow N : A^-$ *and* $\Psi \geq \Gamma^- \Delta_1^- \Omega_1^-$ *and* $\Psi \geq \Gamma^- \Delta_2^- \Omega_2^-$ *implies*
   there exists a P *such that*
   $\Gamma; \Delta_1; \Omega_1 \vdash P = M : A$ *and* $\Gamma; \Delta_2; \Omega_2 \vdash P = N : A$ *and*
   $\Gamma; \Delta_1; \Omega_1 \vdash (P)^\flat \Uparrow A$ *and*
   P *is unique up definitional equality of type labels.*

2. $\Gamma; \Delta_1; \Omega_1 \vdash M : A$ *and* $\Gamma; \Delta_2; \Omega_2 \vdash N : B$ *and* $\Psi \vdash M \longleftrightarrow N : \tau$ *and* $\Psi \geq \Gamma^- \Delta_1^- \Omega_1^-$ *and* $\Psi \geq \Gamma^- \Delta_2^- \Omega_2^-$ *implies*
   there exists C *and* P *such that*
   $\Gamma \vdash C = A : \texttt{type}$ *and* $\Gamma \vdash C = B : \texttt{type}$ *and*
   $\Gamma; \Delta_1; \Omega_1 \vdash P = M : C$ *and* $\Gamma; \Delta_2; \Omega_2 \vdash P = N : C$ *and*
   $A^- = B^- = C^- = \tau$ *and* $\Gamma; \Delta_1; \Omega_1 \vdash (P)^\flat \downarrow C$ *and*
   P *is unique up to definitional equality of type labels.*

3. $\Gamma \vdash A : K$ *and* $\Gamma \vdash B : K$ *and* $\Gamma^- \vdash A \Longleftrightarrow B : K^-$ *implies* $\Gamma \vdash A = B : K$.

4. $\Gamma \vdash A_1 : K_1$ *and* $\Gamma \vdash A_2 : K_2$ *and* $\Gamma^- \vdash A_1 \longleftrightarrow A_2 : \kappa$ *implies*
   $\Gamma \vdash A_1 = A_2 : K_1$ *and* $\Gamma \vdash K_1 = K_2 : \texttt{kind}$ *and* $K_1^- = K_2^- = \kappa$.

5. $\Gamma \vdash K : \texttt{kind}$ *and* $\Gamma \vdash K' : \texttt{kind}$ *and* $\Gamma^- \vdash K \longleftrightarrow K' : \texttt{kind}^-$
   implies $\Gamma \vdash K = K' : \texttt{kind}$.

Proof: By induction on the given algorithmic equality derivation using inversion and injectivity properties. □

Define $\texttt{acnf}(M)$ to be the P resulting from an application of Lemma 92 to two derivations of $\Gamma; \Delta; \Omega \vdash M : A$.

Theorem 93 (Quasi-Canonical Forms)
$\Gamma; \Delta; \Omega \vdash M : A$ *implies* $\Gamma; \Delta; \Omega \vdash (\texttt{acnf}(M))^\flat \Uparrow A$.

Proof: Direct from Lemma 92. □

As stated in Lemma 92, for a given well-typed OLF term M, its almost-canonical form, $\mathtt{acnf}(M)$, is unique up to typing labels. Therefore, the quasi-canonical, $(\mathtt{acnf}(M))^\flat$, is unique. Thus we can use quasi-canonical forms for logical framework representations.

Lemma 94

1. *$\Gamma; \Delta; \Omega \vdash M : A$ implies $((\mathtt{acnf}(M))^\flat)^\sharp_A = \mathtt{acnf}(M)$.*

2. *$\Gamma; \Delta; \Omega \vdash Q \Uparrow A$ implies $((Q)^\sharp_A)^\flat = Q$ and $\mathtt{acnf}((Q)^\sharp_A) = (Q)^\sharp_A$.*

Proof: By structural induction on given derivation. □

We have the following special substitution properties for quasi-canonical forms. Note that the type being substituted into must be constant.

Lemma 95

1. *$\Gamma; \cdot; \cdot \vdash Q' \downarrow a$ and $\Gamma(x{:}a)\Gamma'; \Delta; \Omega \vdash Q \Uparrow A$ implies*
   $\Gamma(\Gamma'[(Q')^\sharp_a/x]); \Delta[(Q')^\sharp_a/x]; \Omega[(Q')^\sharp_a/x] \vdash Q[Q'/x] \Uparrow A$.

2. *$\Gamma; \Delta'; \cdot \vdash Q' \downarrow a$ and $\Gamma; \Delta_1(y{:}a)\Delta_2; \Omega \vdash Q \Uparrow A$ implies*
   $\Gamma; (\Delta_1 \bowtie \Delta')\Delta_2; \Omega \vdash Q[Q'/y] \Uparrow A$.

3. *$\Gamma; \Delta'; \Omega' \vdash Q' : a$ and $\Gamma; \Delta \bowtie \Delta'; \Omega_1(z{:}a)\Omega_2 \vdash Q \Uparrow A$ implies*
   $\Gamma; \Delta; \Omega_1\Omega'\Omega_2 \vdash Q[Q'/z] \Uparrow A$.

4. *$\Gamma; \cdot; \cdot \vdash Q' \downarrow a$ and $\Gamma(x{:}a)\Gamma'; \Delta; \Omega \vdash Q \downarrow A$ implies*
   $\Gamma(\Gamma'[(Q')^\sharp_a/x]); \Delta[(Q')^\sharp_a/x]; \Omega[(Q')^\sharp_a/x] \vdash Q[Q'/x] \downarrow A$.

5. *$\Gamma; \Delta'; \cdot \vdash Q' \downarrow a$ and $\Gamma; \Delta_1(y{:}a)\Delta_2; \Omega \vdash Q \downarrow A$ implies*
   $\Gamma; (\Delta_1 \bowtie \Delta')\Delta_2; \Omega \vdash Q[Q'/y] \downarrow A$.

6. *$\Gamma; \Delta'; \Omega' \vdash Q' : a$ and $\Gamma; \Delta \bowtie \Delta'; \Omega_1(z{:}a)\Omega_2 \vdash Q \downarrow A$ implies*
   $\Gamma; \Delta; \Omega_1\Omega'\Omega_2 \vdash Q[Q'/z] \downarrow A$.

Proof: Direct from Lemma 59, Lemma 94, Theorem 93, Theorem 91, the compositionality of $(-)^\sharp$ and $(-)^\flat$, and the fact that the substitution of a constant type introduces no new redices. □

The next section shows an example of how OLF may be used to represent deductive systems. The adequacy proofs for the example will illustrate how quasi-canonical forms really are strong enough for OLF representations. Chapter 16 shows how OLF can be used to analyze syntactic properties of the CPS transform.

15.4 DeBruijn Terms in OLF

In this section we show a small example application of OLF. In Chapter 10 we gave an Olli program for translating regular lambda terms to deBruijn terms (or vice-versa). We will now recast that example in OLF and prove the correctness of our OLF representation. We begin by reviewing the formal definition of regular terms, deBruijn terms, and the translation between the two.

$$\begin{array}{lrcl} \text{Regular Terms} & e & ::= & x \mid e_1\, e_2 \mid \underline{\lambda} x.\ e \\ \text{deBruijn Terms} & e' & ::= & 1 \mid e' \uparrow \mid e'_1\, e'_2 \mid \Lambda e' \end{array}$$

We use $\underline{\lambda}$ for regular terms to distinguish them from unrestricted OLF functions written with λ.

We use the judgement

$$x_1 \ldots x_n \vdash e \leftrightarrow e'$$

to translate between regular and deBruijn terms. We use K to stand for $x_1 \ldots x_n$. Here are the derivation rules for our translation judgement.

$$\frac{K \vdash e_1 \leftrightarrow e'_1 \qquad K \vdash e_2 \leftrightarrow e'_2}{K \vdash e_1\, e_2 \leftrightarrow e'_1\, e'_2}\,\textbf{tr\_app} \qquad \frac{K\ x \vdash e \leftrightarrow e'}{K \vdash \lambda x.\ e \leftrightarrow \Lambda\, e'}\,\textbf{tr\_lam}^x$$

$$\frac{}{K\ x \vdash x \leftrightarrow 1}\,\textbf{tr\_1} \qquad \frac{K \vdash e \leftrightarrow e'}{K\ x \vdash e \leftrightarrow e' \uparrow}\,\textbf{tr\_}\!\uparrow$$

where x does not occur in K in the $\textbf{tr\_lam}^x$ rule.

We can now write down an OLF signature with constants for representing all of the above judgements and rules. We start with the abstract syntax for regular and deBruijn terms:

```
exp : type.                        exp' : type.

lam : (exp → exp) → exp.           lam' : exp' → exp'.
app : exp → exp → exp.             app' : exp' → exp' → exp'.
                                   shift : exp' → exp'.
                                    one : exp'.
```

Note that these constants are the same as the signature used by the Olli translation program.

We define the following representation function ($\ulcorner - \urcorner$), and its inverse ($\llcorner - \lrcorner$), which form a bijection between regular terms and quasi-canonical objects:

$$\begin{array}{rcl@{\qquad}rcl} \ulcorner x \urcorner & = & x & \llcorner x \lrcorner & = & x \\ \ulcorner \underline{\lambda} x.\ e \urcorner & = & \mathsf{lam}\,(\lambda x.\ \ulcorner e \urcorner) & \llcorner \mathsf{lam}\,(\lambda x.\ E) \lrcorner & = & \underline{\lambda} x.\ \llcorner E \lrcorner \\ \ulcorner e_1\, e_2 \urcorner & = & \mathsf{app}\, \ulcorner e_1 \urcorner \ulcorner e_2 \urcorner & \llcorner \mathsf{app} E_1\, E_2 \lrcorner & = & \llcorner E_1 \lrcorner \llcorner E_2 \lrcorner \end{array}$$

Note that $\ulcorner e[e_1/x] \urcorner = \ulcorner e \urcorner [\ulcorner e_1 \urcorner / x]$.

Lemma 96 *For every regular term* e
$\Gamma; \cdot; \cdot \vdash \ulcorner e \urcorner \Uparrow \mathsf{exp}$
where $\Gamma = x_1{:}\mathsf{exp} \ldots x_n{:}\mathsf{exp}$ *for each* x_i *free in* e.

Proof: By structural induction on e. □

Note that $\Gamma; \cdot; \cdot \vdash (\ulcorner e \urcorner)^{\sharp}_{\mathsf{exp}} : \mathsf{exp}$.

Lemma 97
$\Gamma; \cdot; \cdot \vdash Q \Uparrow \mathsf{exp}$ *implies* $\llcorner Q \lrcorner$ *is a well-formed regular term.*

Proof: Immediate from fact that $\llcorner - \lrcorner$ is a function. □

Note that $\Gamma; \cdot; \cdot \vdash (Q)^{\sharp}_{\mathsf{exp}} : \mathsf{exp}$.

Theorem 98

1. $\llcorner ((\ulcorner e \urcorner)^{\sharp}_{\mathsf{exp}})^{\flat} \lrcorner = e$.
2. $\Gamma; \cdot; \cdot \vdash M : \mathsf{exp}$ *implies* $(\ulcorner \llcorner (\mathtt{acnf}(M))^{\flat} \lrcorner \urcorner)^{\sharp}_{\mathsf{exp}} = \mathtt{acnf}(M)$.

Proof: Direct from previous definitions, Lemma 94 and Theorem 93. □

We define the following function ($\ulcorner - \urcorner$), and its inverse ($\llcorner - \lrcorner$), which form a bijection between deBruijn terms and quasi-canonical objects:

$$\begin{array}{rcl@{\qquad}rcl} \ulcorner 1 \urcorner & = & \mathsf{one} & \llcorner \mathsf{one} \lrcorner & = & 1 \\ \ulcorner e' \uparrow \urcorner & = & \mathsf{shift}\, \ulcorner e' \urcorner & \llcorner \mathsf{shift}\, E' \lrcorner & = & \llcorner E' \lrcorner \uparrow \\ \ulcorner \Lambda e' \urcorner & = & \mathsf{lam'}\, \ulcorner e' \urcorner & \llcorner \mathsf{lam'}\, E' \lrcorner & = & \Lambda \llcorner E' \lrcorner \\ \ulcorner e'_1\, e'_2 \urcorner & = & \mathsf{app'}\, \ulcorner e'_1 \urcorner \ulcorner e'_2 \urcorner & \llcorner \mathsf{app'}\, E'_1\, E'_2 \lrcorner & = & \llcorner E'_1 \lrcorner \llcorner E'_2 \lrcorner \end{array}$$

Note that the quasi-canonical objects do not contain variable binders and thus are also actual OLF objects.

Lemma 99 *For every deBruijn term e'*
$\cdot;\cdot;\cdot \vdash \ulcorner e' \urcorner \Uparrow \mathsf{exp'}$

Proof: By structural induction on e'. □

Lemma 100
$\cdot;\cdot;\cdot \vdash Q \Uparrow \mathsf{exp'}$ *implies* $\llcorner Q \lrcorner$ *is a well-formed deBruijn term.*

Proof: Immediate from fact that $\llcorner - \lrcorner$ is a function. □

Theorem 101

1. $\llcorner((\ulcorner e' \urcorner)^{\sharp}_{\mathsf{exp'}})^{\flat} \lrcorner = e'$.
2. $\Gamma;\cdot;\cdot \vdash M : \mathsf{exp'}$ *implies* $(\ulcorner \llcorner(\mathtt{acnf}(M))^{\flat} \lrcorner \urcorner)^{\sharp}_{\mathsf{exp'}} = \mathtt{acnf}(M)$.

Proof: Direct from previous definitions, Lemma 94 and Theorem 93. □

We now move on to the translation judgements. Because we will use the ordered context to implicitly represent the environments (K from the translation judgements given at the beginning of this section) we also need to define a helper constant which allows us to store terms of type exp in the ordered context.

$$\begin{aligned} \mathsf{tr} &: \mathsf{exp} \to \mathsf{exp'} \to \mathsf{type}. \\ \mathsf{var} &: \mathsf{exp} \to \mathsf{type}. \end{aligned}$$

We can now define constructors to represent the derivation rules.

$$\begin{aligned}
\mathsf{tr\_app'} \;:\;& \Pi E1{:}\mathsf{exp}.\ \Pi E1'{:}\mathsf{exp'}.\ \Pi E2{:}\mathsf{exp}.\ \Pi E2'{:}\mathsf{exp'}.\\
& \mathsf{tr}\, E1\, E1' \,\&\, \mathsf{tr}\, E2\, E2' \twoheadrightarrow\\
& \mathsf{tr}\,(\mathsf{app}\, E1\, E2)\,(\mathsf{app'}\, E1'\, E2').\\
\mathsf{tr\_lam'} \;:\;& \Pi E{:}\mathsf{exp} \rightarrow \mathsf{exp}.\ \Pi E'{:}\mathsf{exp'}.\\
& (\Pi x{:}\mathsf{exp}.\ \mathsf{var}\, x \twoheadrightarrow \mathsf{tr}\,(E\ x)\, E') \twoheadrightarrow\\
& \mathsf{tr}\,(\mathsf{lam}\, E)\,(\mathsf{lam'}\, E').\\
\mathsf{tr\_one} \;:\;& \Pi E{:}\mathsf{exp}.\\
& \top \twoheadrightarrow\\
& \mathsf{var}\, E \twoheadrightarrow\\
& \mathsf{tr}\, E\, \mathsf{one}.\\
\mathsf{tr\_shift} \;:\;& \Pi F{:}\mathsf{exp}.\ \Pi E{:}\mathsf{exp}.\ \Pi E'{:}\mathsf{exp'}.\\
& \mathsf{tr}\, E\, E' \twoheadrightarrow\\
& \mathsf{var}\, F \twoheadrightarrow\\
& \mathsf{tr}\, E\,(\mathsf{shift}\, E').
\end{aligned}$$

Note that these declarations are the clauses of the Olli translation program.

We define $\lceil - \rceil$ and $\lfloor - \rfloor$ on variable lists and ordered contexts as follows:

$$\begin{array}{ll}
\lceil \cdot \rceil = \cdot & \lceil K\ x \rceil = \lceil K \rceil\ (v{:}\mathsf{var}\, x) \quad \text{(for fresh } v\text{)}\\
\lfloor \cdot \rfloor = \cdot & \lfloor \Omega\ (v{:}\mathsf{var}\, x) \rfloor = \lfloor \Omega \rfloor\ x
\end{array}$$

Lemma 102

$K \vdash e \leftrightarrow e'$ implies there exists a Q such that

$$\Gamma; \cdot; \lceil K \rceil \vdash Q \Uparrow \mathsf{tr}\,(\ulcorner e \urcorner)^{\sharp}_{\mathsf{exp}}\,(\ulcorner e' \urcorner)^{\sharp}_{\mathsf{exp'}}$$

where $\Gamma = x_1{:}\mathsf{exp} \ldots x_n{:}\mathsf{exp}$ for each x_i in K.

Proof: By structural induction on the given derivation. □

Note that $\Gamma; \cdot; \lceil K \rceil \vdash (Q)^{\sharp}_{\mathsf{tr}\,(\ulcorner e \urcorner)^{\sharp}_{\mathsf{exp}}\,(\ulcorner e' \urcorner)^{\sharp}_{\mathsf{exp'}}} : \mathsf{tr}\,(\ulcorner e \urcorner)^{\sharp}_{\mathsf{exp}}\,(\ulcorner e' \urcorner)^{\sharp}_{\mathsf{exp'}}$.

Lemma 103

$\Gamma; \cdot; v_1{:}\mathsf{var}\, x_1 \ldots v_n{:}\mathsf{var}\, x_n \vdash Q \Uparrow \mathsf{tr}\, E\, E'$ *implies*

$$\lfloor v_1{:}\mathsf{var}\, x_1 \ldots v_n{:}\mathsf{var}\, x_n \rfloor \vdash \llcorner(\mathtt{acnf}(E))^{\flat}\lrcorner \leftrightarrow \llcorner(\mathtt{acnf}(E'))^{\flat}\lrcorner.$$

Proof: By structural induction on the given derivation. □

Note that $\Gamma; \cdot; v_1{:}\mathsf{var}\,x_1 \ldots v_n{:}\mathsf{var}\,x_n \vdash (Q)^{\sharp}_{\mathsf{tr}\,E\,E'} : \mathsf{tr}\,E\,E'$.

The representation function implicit in Lemma 102 and its inverse in Lemma 103 form a bijection between equivalence classes (up to definitional equality of type labels) of "almost" canonical forms for type family tr and informal translations. This is easily proved by using inversion properties of the quasi-canonical judgements.

Chapter 16

CPS Analysis in OLF

In this chapter we show how an ordered logical framework supports the representation of CPS terms and a machine for their evaluation. The ordering properties of CPS terms which at first appear somewhat ad hoc are directly captured as *typing* properties in the representation that are preserved during evaluation. Complicated stack invariants can be recognized as uniform substitution properties, providing an example of how the organizing principles of a logical framework can contribute conceptually to our understanding of object languages.

The rest of this chapter is organized as follows. In section 16.1 we review CPS terms and their occurrence invariants. In section 16.2 we give a representation of CPS terms in an ordered logical framework in which the occurrence invariants are implicit in the ordered types. In sections 16.3, 16.4 and 16.5, we give representations of the CPS transformation and different evaluation models for CPS terms. In section 16.6 we show how the relationship between the bare evaluation and stack evaluation of a term can be formalized in OLF. We then use this formalization to give a proof that the machines produce the same result. Finally we give some conclusions and directions for further work in section 16.7.

16.1 CPS terms

In this section we review CPS terms and their ordering properties as investigated in [15, 14].

We use the following syntax for direct style (DS) terms:

$$\begin{array}{lrcl} \textit{DS Roots} & r & ::= & e \\ \textit{DS Expressions} & e & ::= & e_1\, e_2 \mid t \\ \textit{DS Trivial Expressions} & t & ::= & \underline{\lambda} x.\, r \mid x \end{array}$$

and for CPS terms:

$$\begin{array}{lrcl} \textit{Root Terms} & r & ::= & \underline{\lambda} k.\, e \\ \textit{Serious Terms} & e & ::= & t_1\, t_2\, c \mid c\, t \\ \textit{Trivial Terms} & t & ::= & \underline{\lambda} x.\, r \mid x \mid v \\ \textit{Continuation Terms} & c & ::= & \underline{\lambda} v.\, e \mid k \end{array}$$

Note that in the CPS syntax, we are distinguishing trivial variables x which are parameters of functions from trivial variables v which are parameters to continuations.

We formulate the left-to-right call-by-value CPS transform as three mutually recursive judgements. A direct-style term r is transformed into a CPS term r' whenever the judgment

$$\vdash r \xrightarrow{DR} r'$$

is satisfied. Given a CPS continuation c, a direct-style expression e is transformed into a CPS expression e' whenever the judgment

$$\vdash e \,;\, c \xrightarrow{DE} e'$$

is satisfied. Finally, a direct-style trivial expression t is transformed into a CPS trivial expression t' whenever the judgment

$$\vdash t \xrightarrow{DT} t'$$

is satisfied.

$$\frac{\vdash e \,;\, k \xrightarrow{DE} e'}{\vdash e \xrightarrow{DR} \underline{\lambda} k.\, e'} \qquad \frac{\vdash t \xrightarrow{DT} t'}{\vdash t \,;\, c \xrightarrow{DE} c\, t'}$$

$$\frac{\vdash e_2 \,;\, \underline{\lambda} v_2.\, v_1\, v_2\, c \xrightarrow{DE} e_2' \qquad \vdash e_1 \,;\, \underline{\lambda} v_1.\, e_2' \xrightarrow{DE} e'}{\vdash e_1\, e_2 \,;\, c \xrightarrow{DE} e'} \quad (v_1 \text{ not free in conclusion})$$

$$\frac{}{\vdash x \xrightarrow{DT} x} \qquad \frac{\vdash r \xrightarrow{DR} r'}{\vdash \underline{\lambda} x.\, r \xrightarrow{DT} \underline{\lambda} x.\, r'}$$

Terms resulting from a left-to-right call-by-value CPS translation of direct-style terms satisfy properties in addition to the CPS syntax, both on occurrences of continuation identifiers k and the parameters v of continuations. In [15] the occurrence properties on continuation identifiers and parameters are separately specified with two judgment families. However these occurrences are tightly coupled and may be naturally captured with just one family of judgments as follows. We shall use four mutually recursive judgments

$$\models^{\mathbf{Root}} r \qquad \Phi \models^{\mathbf{Exp}} e \qquad \Phi \models^{\mathbf{Triv}} t \qquad \Phi \models^{\mathbf{Cont}} c$$

where Φ is a stack of both continuation identifiers and parameters:

$$\Phi ::= \cdot \mid \Phi, k \mid \Phi, v$$

We have the following inference rules for these judgements:

$$\frac{k \models^{\mathbf{Exp}} e}{\models^{\mathbf{Root}} \underline{\lambda} k.\, e}$$

$$\frac{\Phi_t \models^{\mathbf{Triv}} t \qquad \Phi_c \models^{\mathbf{Cont}} c}{\Phi_c \Phi_t \models^{\mathbf{Exp}} c\, t} \qquad \frac{\Phi_1 \models^{\mathbf{Triv}} t_1 \qquad \Phi_0 \models^{\mathbf{Triv}} t_0 \qquad \Phi_c \models^{\mathbf{Cont}} c}{\Phi_c \Phi_o \Phi_1 \models^{\mathbf{Exp}} t_0\, t_1\, c}$$

$$\frac{}{\cdot \models^{\mathbf{Triv}} x} \qquad \frac{\models^{\mathbf{Root}} r}{\cdot \models^{\mathbf{Triv}} \underline{\lambda} x.\, r} \qquad \frac{}{v \models^{\mathbf{Triv}} v}$$

$$\frac{}{k \models^{\mathbf{Cont}} k} \qquad \frac{\Phi, v \models^{\mathbf{Exp}} e}{\Phi \models^{\mathbf{Cont}} \underline{\lambda} v.\, e}$$

Our presentation is general enough to serve as a target for both Plotkin's original call-by-value CPS transform and a one-pass version which avoids administrative redices.

It is easy to see that continuation identifiers, k, are used linearly in each root term and that continuation parameters, v, form a stack during the processing of each serious term. Furthermore, each k is used in a serious term only after all local parameters to continuations are used. These properties can be precisely captured with ordered types.

We next show how to prove that $\vdash r \xrightarrow{DR} r'$ implies $\models^{\mathbf{Root}} r'$ with the aid of an ordered logical framework.

16.2 Ordered Logical Framework Representation

We will now show how the OLL type theory presented in Chapter 13 provides exactly what is needed to capture and reason about the ordering properties of CPS terms. For the sake of readability, we will elide an explicit type argument, A, for the labeling operation, $(-)^{\natural}_{A}$; this should not cause any ambiguity since the necessary type will be apparent from the context. Furthermore, we will use the notation $A' \twoheadrightarrow A$ to stand for $\Pi x{:}A'.\ A$ when x does not appear free in A. Finally, since our representation will not use linear hypotheses, we elide the linear context in all of our judgements.

16.2.1 DS Terms

Our representation of DS terms will use three basic types corresponding to the three kinds of DS terms.

$$\mathsf{droot} : \mathsf{type}. \qquad \mathsf{dexp} : \mathsf{type}. \qquad \mathsf{dtriv} : \mathsf{type}.$$

We will then build our representations from term constructors corresponding to DS terms. Note that representation uses higher-order abstract syntax, so object level functions are represented by meta-level functions.

$$\begin{aligned}
\mathsf{e2r} &: \mathsf{dexp} \twoheadrightarrow \mathsf{droot}.\\
\mathsf{dapp} &: \mathsf{dexp} \twoheadrightarrow \mathsf{dexp} \twoheadrightarrow \mathsf{dexp}.\\
\mathsf{t2e} &: \mathsf{dtriv} \twoheadrightarrow \mathsf{dexp}.\\
\mathsf{dlam} &: (\mathsf{dtriv} \twoheadrightarrow \mathsf{droot}) \twoheadrightarrow \mathsf{dtriv}.
\end{aligned}$$

Given the previous signature, there is an obvious compositional bijection between DS terms and quasi-canonical objects in the above signature. This bijection is established by the following mutually recursive representation functions, $\ulcorner - \urcorner^R$,$\ulcorner - \urcorner^E$,$\ulcorner - \urcorner^T$, and their inverses $\llcorner - \lrcorner^R$,$\llcorner - \lrcorner^E$,$\llcorner - \lrcorner^T$.

$$\ulcorner e \urcorner^R \;=\; \mathsf{e2r}\,\ulcorner e \urcorner^E \qquad\qquad \llcorner \mathsf{e2r}\, E \lrcorner_R \;=\; \llcorner E \lrcorner_E$$

$$\begin{array}{rclrcl}
\ulcorner e_0\, e_1 \urcorner^E & = & \mathsf{dapp}\, \ulcorner e_0 \urcorner^E \ulcorner e_1 \urcorner^E & \llcorner \mathsf{dapp}\, E_0\, E_1 \lrcorner_E & = & \llcorner E_0 \lrcorner_E \llcorner E_1 \lrcorner_E \\
\ulcorner t \urcorner^E & = & \mathsf{d2e}\, \ulcorner t \urcorner^T & \llcorner \mathsf{d2e}\, T \lrcorner_E & = & \llcorner T \lrcorner_T \\
\ulcorner \underline{\lambda} x.\, r \urcorner^T & = & \mathsf{dlam}\, (\lambda x.\, \ulcorner r \urcorner^R) & \llcorner \mathsf{dlam}\, (\lambda x.\ R) \lrcorner_T & = & \underline{\lambda} x.\, \llcorner R \lrcorner_R \\
\ulcorner x \urcorner^T & = & x & \llcorner x \lrcorner_T & = & x
\end{array}$$

Lemma 104

For every DS term e where $\Gamma = x_1{:}\mathsf{dtriv} \ldots x_n{:}\mathsf{dtriv}$ for each x_i free in e:

1. $\Gamma; \cdot; \cdot \vdash \ulcorner e \urcorner^R \Uparrow \mathsf{droot}$

2. $\Gamma; \cdot; \cdot \vdash \ulcorner e \urcorner^E \Uparrow \mathsf{dexp}$

3. $\Gamma; \cdot; \cdot \vdash \ulcorner e \urcorner^T \Uparrow \mathsf{dtriv}$

Proof: By structural induction on e. □

Lemma 105

1. $\Gamma; \cdot; \cdot \vdash Q \Uparrow \mathsf{droot}$ *implies* $\llcorner Q \lrcorner_R$ *is a well-formed DS root.*

2. $\Gamma; \cdot; \cdot \vdash Q \Uparrow \mathsf{dexp}$ *implies* $\llcorner Q \lrcorner_E$ *is a well-formed DS expression.*

3. $\Gamma; \cdot; \cdot \vdash Q \Uparrow \mathsf{dexp}$ *implies* $\llcorner Q \lrcorner_T$ *is a well-formed DS trivial term.*

Proof: Immediate from fact that $\llcorner - \lrcorner$ is a function. □

16.2.2 CPS Terms

Our representation of CPS terms will use four basic types corresponding to the four kinds of CPS terms.

$$\mathsf{root} : \mathsf{type}. \qquad \mathsf{exp} : \mathsf{type}. \qquad \mathsf{triv} : \mathsf{type}. \qquad \mathsf{cont} : \mathsf{type}.$$

We will then build our representations from term constructors corresponding to CPS terms. The use of ordered types forces the CPS term representations to have the linearity and ordering constraints noted at the end of section 16.1.

$$\begin{array}{rcl}
\mathsf{klam} & : & (\mathsf{cont} \twoheadrightarrow \mathsf{exp}) \to \mathsf{root}. \\
\mathsf{kapp} & : & \mathsf{cont} \twoheadrightarrow \mathsf{triv} \twoheadrightarrow \mathsf{exp}. \\
\mathsf{app} & : & \mathsf{cont} \twoheadrightarrow \mathsf{triv} \twoheadrightarrow \mathsf{triv} \twoheadrightarrow \mathsf{exp}. \\
\mathsf{lam} & : & (\mathsf{triv} \to \mathsf{root}) \to \mathsf{triv}. \\
\mathsf{vlam} & : & (\mathsf{triv} \twoheadrightarrow \mathsf{exp}) \twoheadrightarrow \mathsf{cont}.
\end{array}$$

The intuition behind these type declarations may best be gleaned from the representation function and its adequacy theorem below. Note that a positive occurrence of an unrestricted function $\to$ as in the type of klam imposes a restriction on the corresponding argument: it may not depend on the continuation k or parameters v. On the other hand, a negative occurrence of $\to$ as in the type of lam licenses the unrestricted use of the corresponding bound variable x. The right ordered functions $\twoheadrightarrow$ impose the stack-like discipline on parameters of continuations and the continuations themselves explained in the previous section.

Given the previous signature, there is a compositional bijection between CPS terms satisfying the occurrence conditions and quasi-canonical objects in the above signature. This bijection is established by the following representation function, $\ulcorner - \urcorner$ and its inverse $\llcorner - \lrcorner$.

$$\begin{array}{rclcrcl}
\ulcorner \underline{\lambda} k.\, e \urcorner & = & \mathsf{klam}\, (\overset{>}{\lambda} k \ulcorner e \urcorner) & \qquad & \llcorner \mathsf{klam}\, (\overset{>}{\lambda} k.\ E) \lrcorner & = & \underline{\lambda} k.\, \llcorner E \lrcorner \\
 & & & & & & \\
\ulcorner t_0\, t_1\, c \urcorner & = & \mathsf{app}^{>} \ulcorner c \urcorner^{>} \ulcorner t_0 \urcorner^{>} \ulcorner t_1 \urcorner & & \llcorner \mathsf{app}^{>} C^{>} T_0{}^{>} T_1 \lrcorner & = & \llcorner T_0 \lrcorner \llcorner T_1 \lrcorner \llcorner C \lrcorner \\
\ulcorner c\, t \urcorner & = & \mathsf{kapp}^{>} \ulcorner c \urcorner^{>} \ulcorner t \urcorner & & \llcorner \mathsf{kapp}^{>} C^{>} T \lrcorner & = & \llcorner C \lrcorner \llcorner T \lrcorner \\
 & & & & & & \\
\ulcorner \underline{\lambda} x.\, r \urcorner & = & \mathsf{lam}\, (\lambda x.\, \ulcorner r \urcorner) & & \llcorner \mathsf{lam}\, (\lambda x.\ R) \lrcorner & = & \underline{\lambda} x.\, \llcorner R \lrcorner \\
\ulcorner x \urcorner & = & x & & \llcorner x \lrcorner & = & x \\
\ulcorner v \urcorner & = & v & & \llcorner v \lrcorner & = & v \\
 & & & & & & \\
\ulcorner \underline{\lambda} v.\, e \urcorner & = & \mathsf{vlam}^{>} (\overset{>}{\lambda} v.\, \ulcorner e \urcorner) & & \llcorner \mathsf{vlam}^{>} (\overset{>}{\lambda} v.\ E) \lrcorner & = & \underline{\lambda} v.\, \llcorner E \lrcorner \\
\ulcorner k \urcorner & = & k & & \llcorner k \lrcorner & = & k
\end{array}$$

$$\begin{array}{rclcrcl}
\ulcorner . \urcorner & = & \cdot & \qquad & \llcorner \cdot \lrcorner & = & \cdot \\
\ulcorner \Phi, v \urcorner & = & \ulcorner \Phi \urcorner, v{:}\mathsf{triv} & & \llcorner \Phi, v{:}\mathsf{triv} \lrcorner & = & \llcorner \Phi \lrcorner, v \\
\ulcorner \Phi, k \urcorner & = & \ulcorner \Phi \urcorner, k{:}\mathsf{cont} & & \llcorner \Phi, k{:}\mathsf{cont} \lrcorner & = & \llcorner \Phi \lrcorner, k
\end{array}$$

Note that and $\llcorner \ulcorner u \urcorner \lrcorner = u$ for any term u. Additionally, since variables are mapped to variables, the representation function and its inverse are compositional (i.e., commute with substitution).

We formally prove the correspondence in two parts.

Theorem 106 (Representations are Canonical Forms) *Consider terms r, e, and t with free ordinary variables among $x_1, \ldots, x_n$ and let $\Gamma = x_1{:}\mathsf{triv} \ldots x_n{:}\mathsf{triv}$.*

1. *If* $\models^{\mathbf{Root}} r$ *then* $\Gamma; \cdot \vdash \ulcorner r \urcorner \Uparrow \mathsf{root}$.
2. *If* $\Phi \models^{\mathbf{Exp}} e$ *then* $\Gamma; \ulcorner \Phi \urcorner \vdash \ulcorner e \urcorner \Uparrow \mathsf{exp}$.
3. *If* $\Phi \models^{\mathbf{Triv}} t$ *then* $\Gamma; \ulcorner \Phi \urcorner \vdash \ulcorner t \urcorner \Uparrow \mathsf{triv}$.
4. *If* $\Phi \models^{\mathbf{Cont}} c$ *then* $\Gamma; \ulcorner \Phi \urcorner \vdash \ulcorner c \urcorner \Uparrow \mathsf{cont}$.

Proof: By induction on the structure of the given derivations. We give a representative case.

$$\text{case:} \quad \frac{k \models^{\mathbf{Exp}} e}{\models^{\mathbf{Root}} \underline{\lambda} k.\ e}$$

$$\begin{array}{rl}
\Gamma; k{:}\mathsf{cpair} \vdash \ulcorner e \urcorner \Uparrow \mathsf{exp} & \quad\quad \text{ind. hyp.} \\
\Gamma; \cdot \vdash \overset{>}{\lambda} k.\ \ulcorner e \urcorner \Uparrow \mathsf{cpair} \twoheadrightarrow \mathsf{exp} & \quad\quad \twoheadrightarrow_I \\
\Gamma; \cdot \vdash \mathsf{klam} \downarrow (\mathsf{cpair} \twoheadrightarrow \mathsf{exp}) \rightarrow \mathsf{root} & \quad\quad \textbf{const} \\
\Gamma; \cdot \vdash \mathsf{klam}\, (\overset{>}{\lambda} k.\ \ulcorner e \urcorner) \Uparrow \mathsf{root} & \quad\quad \rightarrow_E,\ \textbf{coercion}
\end{array}$$

□

Theorem 107 (Canonical Forms are Representations) *Let $\Gamma = x_1{:}\mathsf{triv}, \ldots, x_n{:}\mathsf{triv}$ be given.*

1. *For any Q such that* $\Gamma; \cdot \vdash Q \Uparrow \mathsf{root}$,
   $\llcorner Q \lrcorner$ *is defined and* $\models^{\mathbf{Root}} \llcorner Q \lrcorner$.
2. *For any* $\Omega = v_1{:}\mathsf{triv} \ldots v_n{:}\mathsf{triv}$ *and Q with* $\Gamma; k{:}\mathsf{cont}, \Omega \vdash Q \Uparrow \mathsf{exp}$, $\llcorner Q \lrcorner$ *is defined and* $k, \llcorner \Omega \lrcorner \models^{\mathbf{Exp}} \llcorner Q \lrcorner$.
3. *For any* $\Omega = v_1{:}\mathsf{triv} \ldots v_n{:}\mathsf{triv}$ *and Q such that* $\Gamma; \Omega \vdash Q \Uparrow \mathsf{triv}$,
   $\llcorner Q \lrcorner$ *is defined and* $\llcorner \Omega \lrcorner \models^{\mathbf{Triv}} \llcorner Q \lrcorner$.

4. *For any* $\Omega = v_1{:}\mathsf{triv} \ldots v_n{:}\mathsf{triv}$ *and* Q *with* $\Gamma; k{:}\mathsf{cont}, \Omega \vdash Q \Uparrow \mathsf{cont}$, $\llcorner Q \lrcorner$ *is defined and* $k, \llcorner \Omega \lrcorner \models^{\mathbf{Cont}} \llcorner Q \lrcorner$.

Proof: By induction on the structure of the given canonical derivations. We give a representative case.

case: $\Gamma; \cdot \vdash \mathsf{klam}\,(\overset{>}{\lambda} k.\ e) \Uparrow \mathsf{root}$

$\Gamma; \cdot \vdash \mathsf{klam}\,(\overset{>}{\lambda} k.\ e) \downarrow \mathsf{root}$	inversion on **coercion**
$\Gamma; \cdot \vdash \overset{>}{\lambda} k.\ e \Uparrow \mathsf{cpair} \twoheadrightarrow \mathsf{exp}$	inversion on $\rightarrow_E$
$\Gamma; k{:}\mathsf{cpair} \vdash e \Uparrow \mathsf{exp}$	inversion on $\twoheadrightarrow_I$
$\llcorner e \lrcorner$ is defined, and $k \models^{\mathbf{Exp}} \llcorner e \lrcorner$	ind. hyp.
$\models^{\mathbf{Root}} \underline{\lambda} k.\ \llcorner e \lrcorner$	by definition

□

16.3 CPS Transform

We represent CPS transform with three basic types corresponding to the three judgements of the transform.

$\mathsf{cps\_r} : \mathsf{droot} \rightarrow \mathsf{root} \rightarrow \mathsf{type}.\quad \mathsf{cps\_e} : \mathsf{dexp} \rightarrow \mathsf{cont} \rightarrow \mathsf{exp} \rightarrow \mathsf{type}.\quad \mathsf{cps\_t} : \mathsf{dtriv} \rightarrow \mathsf{triv} \rightarrow \mathsf{type}.$

We then use the following terms to construct representations of the CPS transform.

$$\begin{aligned}
\mathsf{cps\_root} \ :\ & \Pi E{:}\mathsf{dexp}.\ \Pi E'{:}\mathsf{cont} \twoheadrightarrow \mathsf{exp}.\\
& (\Pi k{:}\mathsf{cont}.\ \mathsf{cps\_e}\, E\, k\, (E' \,^{>} k)) \rightarrow \mathsf{cps\_r}\,(\mathsf{e2r}\, E)\,(\mathsf{klam}\, E').
\end{aligned}$$

$$\begin{aligned}
\mathsf{cps\_app} \ :\ & \Pi E_0{:}\mathsf{dexp}.\ \Pi E_1{:}\mathsf{dexp}.\ \Pi C{:}\mathsf{cont}.\ \Pi E_1'{:}\mathsf{triv} \twoheadrightarrow \mathsf{exp}.\ \Pi E'{:}\mathsf{exp}.\\
& \mathsf{cps\_e}\, E_0\,(\mathsf{vlam}\,^{>} E_1')\, E' \rightarrow\\
& (\Pi v_0{:}\mathsf{triv}.\ \mathsf{cps\_e}\, E_1\,(\mathsf{vlam}\,^{>} \overset{>}{\lambda} v_1{:}\mathsf{triv}.\ \mathsf{app}\,^{>} C\,^{>} v_0\,^{>} v_1)\,(E_1'\,^{>} v_0)) \rightarrow\\
& \mathsf{cps\_e}\,(\mathsf{dapp}\, E_0\, E_1)\, C\, E'.
\end{aligned}$$

$$\begin{aligned}
\mathsf{cps\_triv} \ :\ & \Pi T{:}\mathsf{dtriv}.\ \Pi C{:}\mathsf{cont}.\ \Pi T'{:}\mathsf{triv}.\\
& \mathsf{cps\_t}\, T\, T' \rightarrow \mathsf{cps\_e}\,(\mathsf{t2e}\, T)\, C\,(\mathsf{kapp}\,^{>} C\,^{>} T').
\end{aligned}$$

$$\begin{aligned}
\mathsf{cps\_lam} \ :\ & \Pi R{:}\mathsf{dtriv} \rightarrow \mathsf{droot}.\ \Pi R'{:}\mathsf{triv} \rightarrow \mathsf{root}.\\
& (\Pi x{:}\mathsf{dtriv}.\ \Pi x'{:}\mathsf{triv}.\ \mathsf{cps\_t}\, x\, x' \rightarrow \mathsf{cps\_r}\,(R\, x)\,(R'\, x')) \rightarrow\\
& \mathsf{cps\_t}\,(\mathsf{dlam}\, R)\,(\mathsf{lam}\, R').
\end{aligned}$$

We may now show the adequacy of above representation in two parts. In the informal translation we map variables x to themselves; in the formalization we map each variable x from the direct-style term to a corresponding variable x' in the continuation-passing term. These variables and their relationship are captured in contexts

$$\begin{array}{rcl} \Gamma & = & x_1{:}\mathsf{dtriv} \ldots x_n{:}\mathsf{dtriv} \\ \Gamma' & = & x_1'{:}\mathsf{triv} \ldots x_n'{:}\mathsf{triv} \\ \Gamma_m & = & m_1{:}\mathsf{cps\_t}\, x_1\, x_1' \ldots m_n{:}\mathsf{cps\_t}\, x_n\, x_n' \end{array}$$

which always occur together in this manner.

Theorem 108 (Representations are Canonical Forms) *Let $\Gamma^* = \Gamma, \Gamma', \Gamma_m$ be a context of the form explained above that contains all free variables occurring in the relevant judgment. Then*

1. $\vdash r \xrightarrow{DR} r'$ *implies* $\exists Q.\ \Gamma^*; \cdot \vdash Q \Uparrow \mathsf{cps\_r}\, (\ulcorner r \urcorner^R)^\sharp\, (\ulcorner r' \urcorner)^\sharp$ *and* $\models^{\mathbf{Root}} r'$.
2. $\vdash e \;;\; c \xrightarrow{DE} e'$ *and* $\Phi \models^{Cont} c$ *implies*
   $\exists Q.\ \Gamma^*, \ulcorner \Phi \urcorner; \cdot; \cdot \vdash Q \Uparrow \mathsf{cps\_e}\, (\ulcorner e \urcorner^E)^\sharp\, (\ulcorner c \urcorner)^\sharp\, (\ulcorner e' \urcorner)^\sharp$ *and* $\Phi \models^{Exp} e'$.
3. $\vdash t \xrightarrow{DT} t'$ *implies* $\exists Q.\ \Gamma^*; \cdot; \cdot \vdash Q \Uparrow \mathsf{cps\_t}\, (\ulcorner t \urcorner^T)^\sharp\, (\ulcorner t' \urcorner)^\sharp$ *and* $\cdot \models^{\mathbf{Triv}} t'$.

Proof: By structural induction on the given derivation. We show a representative case.

$$\text{case:}\quad \frac{\vdash e_2 \;;\; \underline{\lambda} v_2.\, v_1\, v_2\, c \xrightarrow{DE} e_2' \qquad \vdash e_1 \;;\; \underline{\lambda} v_1.\, e_2' \xrightarrow{DE} e'}{\vdash e_1\, e_2 \;;\; c \xrightarrow{DE} e'} \quad (v_1 \text{ not free in conclusion})$$

Then

$$\begin{array}{lr}
\Phi \models^{\mathbf{Cont}} c & \text{assumption} \\
v_1 \models^{\mathbf{Triv}} v_1 \ \text{ and } \ v_2 \models^{\mathbf{Triv}} v_2 & \text{defn.} \\
\Phi, v_1 \models^{\mathbf{Cont}} \underline{\lambda} v_2.\, v_1\, v_2\, c & \text{defn.} \\
(\Gamma^*, \ulcorner \Phi \urcorner, v_1{:}\mathsf{triv}); \cdot \vdash Q_2 \Uparrow \mathsf{cps\_e}\, (\ulcorner e_2 \urcorner^E)^\sharp\, (\mathsf{vlam}\,^{>}\overset{>}{\lambda} v_2.\, \mathsf{app}\, (\ulcorner c \urcorner)^\sharp\, v_1\, v_2)\, (\ulcorner e_2' \urcorner)^\sharp & \\
\quad \text{and } \ \Phi, v_1 \models^{\mathbf{Exp}} e_2' & \text{ind. hyp.} \\
\Gamma^*; (\ulcorner \Phi \urcorner, v_1{:}\mathsf{triv}) \vdash \ulcorner e_2' \urcorner \Uparrow \mathsf{exp} & \text{Theorem 106, Lemma 90} \\
\Gamma^*; \ulcorner \Phi \urcorner \vdash \overset{>}{\lambda} v_1.\, \ulcorner e_2' \urcorner \Uparrow \mathsf{triv} \twoheadrightarrow \mathsf{exp} & \twoheadrightarrow_I \\
(\Gamma^*, \ulcorner \Phi \urcorner); \cdot \vdash \overset{>}{\lambda} v_1.\, \ulcorner e_2' \urcorner \Uparrow \mathsf{triv} \twoheadrightarrow \mathsf{exp} & \text{Lemma 90} \\
(\Gamma^*, \ulcorner \Phi \urcorner); \cdot \vdash \lambda v_1.\, Q_2 \Uparrow &
\end{array}$$

$\Pi v_1{:}\mathsf{triv}.\ \mathsf{cps\_e}\,(\ulcorner e_2 \urcorner^E)^\sharp\,(\mathsf{vlam}^{>}\,\overset{>}{\lambda} v_2.\ \mathsf{app}\,(\ulcorner c \urcorner)^\sharp\, v_1\, v_2)\ (\ulcorner e_2' \urcorner)^\sharp$ Π_I

$\Phi \models^{\mathbf{Cont}} \underline{\lambda} v_1.\ e_2'$ defn.

$(\Gamma^*, \ulcorner \Phi \urcorner); \cdot \vdash Q_1 \Uparrow \mathsf{cps\_e}\,(\ulcorner e_1 \urcorner^E)^\sharp\,(\mathsf{vlam}^{>}\,\overset{>}{\lambda} v_1.\ (\ulcorner e_2' \urcorner)^\sharp)\,(\ulcorner e' \urcorner)^\sharp$
and $\Phi \models^{\mathbf{Exp}} e'$ ind. hyp.

$(\Gamma^*, \ulcorner \Phi \urcorner); \cdot \vdash \mathsf{cps\_app}\,\ulcorner e_1 \urcorner^E\,\ulcorner e_2 \urcorner^E\,\ulcorner c \urcorner\,(\overset{>}{\lambda} v_1.\ \ulcorner e_2' \urcorner)\,\ulcorner e' \urcorner\, Q_1\,(\lambda v_1.\ Q_2) \Uparrow$
$\mathsf{cps\_e}\,(\mathsf{dapp}\,(\ulcorner e_1 \urcorner^E)^\sharp\,(\ulcorner e_2 \urcorner^E)^\sharp)\,(\ulcorner c \urcorner)^\sharp\,(\ulcorner e' \urcorner)^\sharp$ Π_E, **coercion**

□

Theorem 109 (Canonical Forms are Representations) *Assume* Γ *only contains variables of the following types:* dtriv, triv, $\mathsf{cps\_t}\,x\,x'$ *(for* x, x' *in* Γ*), and* cont. *Further assume the types below are canonical.*

1. $\Gamma; \cdot; \cdot \vdash M \Uparrow \mathsf{cps\_r}\,R\,R'$ *implies* $\vdash \llcorner R \lrcorner_R \xrightarrow{DR} \llcorner R' \lrcorner$.
2. $\Gamma; \cdot; \cdot \vdash M \Uparrow \mathsf{cps\_e}\,E\,C\,E'$ *implies* $\vdash \llcorner E \lrcorner_E\,;\,\llcorner C \lrcorner \xrightarrow{DE} \llcorner E' \lrcorner$.
3. $\Gamma; \cdot; \cdot \vdash M \Uparrow \mathsf{cps\_t}\,T\,T'$ *implies* $\vdash \llcorner T \lrcorner_T \xrightarrow{DR} \llcorner T' \lrcorner$.

Proof: By structural induction on the given canonical derivation. We give a representative case.

case: $\Gamma; \cdot \vdash \mathsf{cps\_app}\,E_1\,E_2\,C\,(\overset{>}{\lambda} v_1.\ E_2')\,E'\,D_1\,(\lambda v_1.\ D_2) \Uparrow \mathsf{cps\_e}\,(\mathsf{dapp}\,E_1\,E_2)\,C\,E'$
Then

$\Gamma; \cdot \vdash D_1 \Uparrow \mathsf{cps\_e}\,E_1\,(\mathsf{vlam}^{>}(\overset{>}{\lambda} v_1{:}\mathsf{triv}.\ E_2'))\,E'$ inversion on Π_E

Note $\Gamma; \cdot; \cdot \vdash (\overset{>}{\lambda} v_1{:}\mathsf{triv}.\ E_2')^{>} v_1 = E_2' : \mathsf{exp}$

$(\Gamma, v_1{:}\mathsf{triv}); \cdot \vdash D_2 \Uparrow$
$\mathsf{cps\_e}\,E_2\,(\mathsf{vlam}^{>}(\overset{>}{\lambda} v_2{:}\mathsf{triv}.\ \mathsf{app}^{>} C^{>} v_1{}^{>} v_2))\,E_2'$ inversion on Π_E, Π_I

$\llcorner E_2 \lrcorner_E; \underline{\lambda} v_2.\ v_1\, v_2\, \llcorner C \lrcorner \xrightarrow{DE} \llcorner E_2' \lrcorner$ ind. hyp.

$\llcorner E_1 \lrcorner_E; \underline{\lambda} v_1.\ E_2' \xrightarrow{DE} \llcorner E' \lrcorner$ ind. hyp.

Note v_1 cannot be in Γ

$\llcorner E_2 \lrcorner_E\, \llcorner E_2 \lrcorner_E; \llcorner P \lrcorner \xrightarrow{DE} \llcorner E' \lrcorner$ defn.

□

The adequacy of our representation implies that the terms resulting from a CPS transformation satisfy the occurrence conditions of section 16.1.

$$\vdash r \xrightarrow{DR} r' \text{ implies } \models^{\mathbf{Root}} r'.$$

It is also possible to represent a one-pass CPS transformation directly using third-order constructors and still guarantee ordering properties for the results. A further examination of this optimized translation is left as future work.

16.4 Bare Abstract Machine

We now begin extending our representation to include evaluation of CPS terms. We will begin by showing a representation of a naive evaluator which makes no use of the ordering invariants. The following is a bare abstract machine for CPS evaluation.

$$\frac{\vdash^{\mathbf{Exp}}_{\mathbf{B}} e \hookrightarrow a}{\vdash^{\mathbf{Root}}_{\mathbf{B}} \underline{\lambda} k.\ e \hookrightarrow a} \qquad\qquad \frac{}{\vdash^{\mathbf{Exp}}_{\mathbf{B}} k\, t \hookrightarrow t}$$

$$\frac{\vdash^{\mathbf{Exp}}_{\mathbf{B}} e[t/v] \hookrightarrow a}{\vdash^{\mathbf{Root}}_{\mathbf{B}} (\underline{\lambda} v.\ e)\, t \hookrightarrow a} \qquad \frac{\vdash^{\mathbf{Exp}}_{\mathbf{B}} e[t/x][c/k] \hookrightarrow a}{\vdash^{\mathbf{Root}}_{\mathbf{B}} (\underline{\lambda} x.\ \underline{\lambda} k.\ e)\, t\, c \hookrightarrow a}$$

Notice that this machine describes a regular big-step operational semantics for a λ-calculus—every redex is reduced by a substitution.

We introduce two type constructors to represent bare evaluations:

$$\mathsf{evalr}_B : \mathsf{root} \to \mathsf{triv} \to \mathsf{type}. \qquad \mathsf{evale}_B : \mathsf{exp} \to \mathsf{triv} \to \mathsf{type}.$$

Additionally, we introduce a new object to our signature for CPS terms:

$$\mathsf{ret} : \mathsf{cont}$$

ret[1] will be substituted for the continuation identifiers, k, in bare evaluations. In order to make the inverse representation function on objects well-defined, we augment it

[1] We chose the term ret for continuation identifiers since a continuation is not invoked until the end of the current computation, thus it is like a return statement.

with a continuation identifier k when applied to a continuation term or a serious term:

$$\begin{aligned}
\llcorner \mathsf{klam}\,(\overset{>}{\lambda} k.\ E)\lrcorner &= \underline{\lambda} k.\ \llcorner E \lrcorner_k \\
\llcorner \mathsf{app}\,^{>}C\,^{>}T_0\,^{>}T_1 \lrcorner_k &= \llcorner T_0 \lrcorner\, \llcorner T_1 \lrcorner\, \llcorner C \lrcorner_k \\
\llcorner \mathsf{kapp}\,^{>}C\,^{>}T \lrcorner_k &= \llcorner C \lrcorner_k\, \llcorner T \lrcorner \\
\llcorner \mathsf{lam}\,(\lambda x.\ R)\lrcorner &= \underline{\lambda} x.\ \llcorner R \lrcorner \\
\llcorner x \lrcorner &= x \\
\llcorner v \lrcorner &= v \\
\llcorner \mathsf{vlam}\,^{>}(\overset{>}{\lambda} v.\ E)\lrcorner_k &= \underline{\lambda} v.\ \llcorner E \lrcorner_k \\
\llcorner k \lrcorner_k &= k \\
\llcorner \mathsf{ret} \lrcorner_k &= k
\end{aligned}$$

We use the following object constructors to represent bare evaluations:

$$\begin{aligned}
\mathsf{evr}_B \quad &: \quad \Pi A{:}\mathsf{triv}.\ \Pi E{:}\mathsf{cont} \twoheadrightarrow \mathsf{exp}. \\
&\qquad \mathsf{evale}_B\,(E\,^{>}\mathsf{ret})\,A \to \mathsf{evalr}_B\,(\mathsf{klam}\,E)\,A. \\[1ex]
\mathsf{eve}_B\_0 \quad &: \quad \Pi A{:}\mathsf{triv}.\ \mathsf{evale}_B\,(\mathsf{kapp}\,^{>}\mathsf{ret}\,^{>}A)\,A. \\[1ex]
\mathsf{eve}_B\_1 \quad &: \quad \Pi A{:}\mathsf{triv}.\ \Pi T{:}\mathsf{triv}.\ \Pi E{:}\mathsf{triv} \twoheadrightarrow \mathsf{exp}. \\
&\qquad \mathsf{evale}_B\,(E\,^{>}T)\,A \to \mathsf{evale}_B\,(\mathsf{kapp}\,^{>}(\mathsf{vlam}\,^{>}E)\,^{>}T)\,A. \\[1ex]
\mathsf{eve}_B\_\mathsf{app} \quad &: \quad \Pi A{:}\mathsf{triv}.\ \Pi T{:}\mathsf{triv}.\ \Pi C{:}\mathsf{cont}.\ \Pi E{:}\mathsf{triv} \to \mathsf{cont} \twoheadrightarrow \mathsf{exp}. \\
&\qquad \mathsf{evale}_B\,((E\,T)\,^{>}C)\,A \to \\
&\qquad \mathsf{evale}_B\,(\mathsf{app}\,^{>}C\,^{>}(\mathsf{lam}\,(\lambda x{:}\mathsf{triv}.\ \mathsf{klam}\,(E\,x)))\,^{>}T)\,A.
\end{aligned}$$

We prove that our representation of bare evaluations is in bijective correspondence with actual bare evaluations in two parts as follows.

Theorem 110 (Bare evaluations are canonical forms)
Assume r and e have no free x.

1. $\models^{\mathbf{Root}} r$, *and* $\vdash^{\mathbf{Root}}_B r \hookrightarrow a$ *implies* $\exists Q.\ \ \cdot;\cdot \vdash Q \Uparrow \mathsf{evalr}_B\,(\ulcorner r \urcorner)^\sharp\,(\ulcorner a \urcorner)^\sharp$.
2. $k \models^{\mathbf{Exp}} e$, *and* $\vdash^{\mathbf{Exp}}_B e \hookrightarrow a$ *implies* $\exists Q.\ \ \cdot;\cdot \vdash Q \Uparrow \mathsf{evale}_B\,(\ulcorner e \urcorner[\mathsf{ret}/k])^\sharp\,(\ulcorner a \urcorner)^\sharp$.

Proof: By structural induction on the given canonical derivation.

case: $\dfrac{\vdash^{\mathbf{Exp}}_B\ e[t/x][c/k] \hookrightarrow a}{\vdash^{\mathbf{Exp}}_B (\underline{\lambda} x.\ \underline{\lambda} k.\ e)\,t\,c \hookrightarrow a}$ and $k' \models^{\mathbf{Exp}} (\underline{\lambda} x.\ \underline{\lambda} k.\ e)\,t\,c$

Then

$\cdot \models^{\mathbf{Triv}} t$ and $k \models^{\mathbf{Exp}} e$ and $k' \models^{\mathbf{Cont}} c$ inversion

$\cdot;\cdot \vdash \ulcorner t \urcorner \Uparrow \mathsf{triv}$ and $x{:}\mathsf{triv}; k{:}\mathsf{cont} \vdash \ulcorner e \urcorner \Uparrow \mathsf{exp}$ and
$\quad \cdot; k'{:}\mathsf{cont} \vdash \ulcorner c \urcorner \Uparrow \mathsf{cont}$ Theorem 106

$\cdot;\cdot \vdash \ulcorner t \urcorner \downarrow \mathsf{triv}$ and $x{:}\mathsf{triv}; k{:}\mathsf{cont} \vdash \ulcorner e \urcorner \downarrow \mathsf{exp}$ and
$\quad \cdot; k'{:}\mathsf{cont} \vdash \ulcorner c \urcorner \downarrow \mathsf{cont}$ inversion

$\cdot; k'{:}\mathsf{cont} \vdash \ulcorner e \urcorner [\ulcorner t \urcorner / x][\ulcorner c \urcorner / k] \Uparrow \mathsf{exp}$ Lemma 95

$k' \models^{\mathbf{Exp}} e[t/x][c/k]$ Theorem 107

$\cdot;\cdot \vdash Q \Uparrow \mathsf{evale}_B\, (\ulcorner e[t/x][c/k] \urcorner [\mathsf{ret}/k'])^\sharp\, (\ulcorner a \urcorner)^\sharp$ for some Q ind. hyp.

$\cdot;\cdot \vdash \mathsf{eve}_B\mathsf{\_app}\, \ulcorner a \urcorner \ulcorner t \urcorner\, (\ulcorner c \urcorner [\mathsf{ret}/k'])\, (\lambda x.\ \overset{>}{\lambda} k.\ \ulcorner e \urcorner)\, Q \Uparrow$ Π_E
$\quad \mathsf{evale}_B\, (\mathsf{app}^{>}\, (\ulcorner c \urcorner [\mathsf{ret}/k'])^{\sharp\,>} (\mathsf{lam}\ \lambda x{:}\mathsf{triv}.\ \mathsf{klam}\ \overset{>}{\lambda} k{:}\mathsf{cont}.\ (\ulcorner e \urcorner)^\sharp)^{>} (\ulcorner t \urcorner)^\sharp)\, (\ulcorner a \urcorner)^\sharp.$

□

Theorem 111 (Canonical forms are bare evaluations)

1. $\cdot;\cdot \vdash Q \Uparrow \mathsf{evalr}_B\, R\, A$ *implies* $\models^{\mathbf{Root}} \llcorner R \lrcorner$ *and* $\vdash_B^{\mathbf{Root}} \llcorner R \lrcorner \hookrightarrow \llcorner A \lrcorner$.

2. $\cdot;\cdot \vdash Q \Uparrow \mathsf{evale}_B\, E\, A$ *implies* $\vdash_B^{\mathbf{Exp}} \llcorner E \lrcorner_k \hookrightarrow \llcorner A \lrcorner$ *for any* k.

Proof: By induction on the given canonical typing derivation making use of α-conversion to allow choice of any k.

case: $\cdot;\cdot \vdash \mathsf{evr}_B\, A\, E\, D \Uparrow \mathsf{evalr}\, (\mathsf{klam}\ E)\, A$
Then

$\cdot;\cdot \vdash E \Uparrow \mathsf{cont} \twoheadrightarrow \mathsf{exp}$ and $E = \overset{>}{\lambda} k.\ E'$ inversion

$\cdot;\cdot \vdash D \Uparrow \mathsf{evale}_B\, (E'[\mathsf{ret}/k])^\sharp\, A$ inversion

$k \models^{\mathbf{Exp}} \llcorner E'[\mathsf{ret}/k] \lrcorner_k$ and $\vdash_B^{\mathbf{Exp}} \llcorner E'[\mathsf{ret}/k] \lrcorner_k \hookrightarrow \llcorner A \lrcorner$ ind. hyp.

Note $\llcorner E'[\mathsf{ret}/k] \lrcorner_k = \llcorner E' \lrcorner_k [\llcorner \mathsf{ret}_k \lrcorner / k] = \llcorner E' \lrcorner_k$.

$\vdash_B^{\mathbf{Root}} \underline{\lambda} k.\ \llcorner E' \lrcorner_k \hookrightarrow \llcorner A \lrcorner$ defn

□

In addition to proving that we really have represented bare evaluations, we have also proved that bare evaluation preserves the ordering invariants of CPS terms. This comes for free as a result of theorem 107.

16.5 Stack Abstract Machine

We now consider a more sophisticated evaluation model which makes use of the ordering constraints on CPS terms. Rather than substituting for continuations and continuation parameters, we can evaluate terms by keeping a stack of continuations and their parameters and then effectively treating k and v as pop instructions.

We need stacks, ϕ, of both trivial terms and continuation terms for our stack evaluator.

$$\phi ::= \cdot \mid \phi, t \mid \phi, c$$

We give a big step operational semantics for stack evaluation as follows:

$$\frac{\bullet \vdash^{\mathbf{Exp}}_{\mathbf{St}} e \hookrightarrow a}{\vdash^{\mathbf{Root}}_{\mathbf{St}} \underline{\lambda} k.\ e \hookrightarrow a} \qquad \frac{\phi \vdash^{\mathbf{Triv}}_{\mathbf{St}} t \hookrightarrow a; \bullet}{\phi \vdash^{\mathbf{Exp}}_{\mathbf{St}} k\, t \hookrightarrow a}$$

$$\frac{\phi \vdash^{\mathbf{Triv}}_{\mathbf{St}} t \hookrightarrow t'; \phi', \underline{\lambda} v.\ e \qquad \phi', t' \vdash^{\mathbf{Exp}}_{\mathbf{St}} e \hookrightarrow a}{\phi \vdash^{\mathbf{Exp}}_{\mathbf{St}} k\, t \hookrightarrow a} \qquad \frac{\phi \vdash^{\mathbf{Triv}}_{\mathbf{St}} t \hookrightarrow t'; \phi' \qquad \phi', t' \vdash^{\mathbf{Exp}}_{\mathbf{St}} e \hookrightarrow a}{\phi \vdash^{\mathbf{Exp}}_{\mathbf{St}} (\underline{\lambda} v.\ e)\, t \hookrightarrow a}$$

$$\frac{\phi \vdash^{\mathbf{Triv}}_{\mathbf{St}} t_1 \hookrightarrow t; \phi' \qquad \phi' \vdash^{\mathbf{Triv}}_{\mathbf{St}} t_0 \hookrightarrow \underline{\lambda} x.\ \underline{\lambda} k.\ e; \phi'' \qquad \phi'' \vdash^{\mathbf{Exp}}_{\mathbf{St}} e[t/x] \hookrightarrow a}{\phi \vdash^{\mathbf{Exp}}_{\mathbf{St}} t_0\, t_1\, k \hookrightarrow a}$$

$$\frac{\phi \vdash^{\mathbf{Triv}}_{\mathbf{St}} t_1 \hookrightarrow t; \phi' \qquad \phi' \vdash^{\mathbf{Triv}}_{\mathbf{St}} t_0 \hookrightarrow \underline{\lambda} x.\ \underline{\lambda} k.\ e'; \phi'' \qquad \phi'', \underline{\lambda} v.\ e \vdash^{\mathbf{Exp}}_{\mathbf{St}} e'[t/x] \hookrightarrow a}{\phi \vdash^{\mathbf{Exp}}_{\mathbf{St}} t_0\, t_1\, (\underline{\lambda} v.\ e) \hookrightarrow a}$$

$$\frac{}{\phi \vdash^{\mathbf{Triv}}_{\mathbf{St}} \underline{\lambda} x.\ r \hookrightarrow \underline{\lambda} x.\ r; \phi} \qquad \frac{}{\phi, t \vdash^{\mathbf{Triv}}_{\mathbf{St}} v \hookrightarrow t; \phi}$$

Notice that this machine only performs substitution to reduce redices between trivial terms. Arguments to continuations are instead pushed onto the stack.

We show an OLF representation of the stack machine which uses the ordered context to represent the stack. We introduce the following new type constructors for representing stack evaluations:

$\mathsf{evalr}_{St} : \mathsf{root} \rightarrow \mathsf{triv} \rightarrow \mathsf{type}. \qquad \mathsf{evale}_{St} : \mathsf{exp} \rightarrow \mathsf{triv} \rightarrow \mathsf{type}. \qquad \mathsf{evalt} : \mathsf{triv} \rightarrow \mathsf{triv} \rightarrow \mathsf{type}.$

Since we use the ordered context as a stack, we also need constructors which allow us to put continuation terms and trivial terms into the ordered context:

$$\mathsf{cnt} : \mathsf{cont} \rightarrow \mathsf{type}. \qquad \mathsf{var} : \mathsf{triv} \rightarrow \mathsf{type}.$$

cnt and var will be used to represent continuation terms and trivial terms stored in the stack. We also introduce a new object:

$$\mathsf{pop} : \mathsf{triv}$$

which will be substituted for the continuation parameters, v during stack evaluation. As before we need to augment the inverse representation function with Φ as follows:

$$\begin{array}{rcl}
\llcorner \mathsf{klam}\,(\overset{>}{\lambda} k.\ E) \lrcorner & = & \underline{\lambda} k.\ \llcorner E \lrcorner_{k} \\
\llcorner \mathsf{app}\,{}^{>}C\,{}^{>}T_0\,{}^{>}T_1 \lrcorner_{\Phi\Phi_0\Phi_1} & = & \llcorner T_0 \lrcorner_{\Phi_0}\, \llcorner T_1 \lrcorner_{\Phi_1}\, \llcorner C \lrcorner_{\Phi} \\
\llcorner \mathsf{kapp}\,{}^{>}C\,{}^{>}T \lrcorner_{\Phi\Phi_t} & = & \llcorner C \lrcorner_{\Phi}\, \llcorner T \lrcorner_{\Phi_t} \\
\llcorner \mathsf{lam}\,(\lambda x.\ R) \lrcorner_{.} & = & \underline{\lambda} x.\ \llcorner R \lrcorner \\
\llcorner x \lrcorner_{.} & = & x \\
\llcorner v \lrcorner_{v} & = & v \\
\llcorner \mathsf{pop} \lrcorner_{v} & = & v \\
\llcorner \mathsf{vlam}\,{}^{>}(\overset{>}{\lambda} v.\ E) \lrcorner_{\Phi} & = & \underline{\lambda} v.\ \llcorner E \lrcorner_{\Phi,v} \\
\llcorner k \lrcorner_{k} & = & k \\
\llcorner \mathsf{ret} \lrcorner_{k} & = & k
\end{array}$$

The inverse representation function is stated non-deterministically– the splittings for Φ must be guessed. However, it is easy to see that the constraints on Φ in the variable cases ensure there will be at most one correct splitting of Φ at any point in the execution of the function.

We use the following term constructors to represent stack evaluations:

$$\begin{array}{lcl}
\mathsf{evr}_{St} & : & \Pi A{:}\mathsf{triv}.\ \Pi E{:}\mathsf{cont} \twoheadrightarrow \mathsf{exp}. \\
 & & \mathsf{evale}_{St}\,(E\,^{>}\mathsf{ret})\,A \rightarrow \mathsf{evalr}_{St}\,(\mathsf{klam}\ E)\,A.
\end{array}$$

$$\begin{array}{lcl}
\mathsf{eve}_{St}\_0 & : & \Pi A{:}\mathsf{triv}.\ \Pi T{:}\mathsf{triv}. \\
 & & \mathsf{evalt}\,T\,A \twoheadrightarrow \mathsf{evale}_{St}\,(\mathsf{kapp}\,^{>}\mathsf{ret}\,^{>}T)\,A.
\end{array}$$

$$\begin{array}{lcl}
\mathsf{eve}_{St}\_1 & : & \Pi A{:}\mathsf{triv}.\ \Pi T{:}\mathsf{triv}.\ \Pi T'{:}\mathsf{triv}.\ \Pi E{:}\mathsf{triv} \twoheadrightarrow \mathsf{exp}. \\
 & & (\mathsf{var}\,T' \twoheadrightarrow \mathsf{evale}_{St}\,(E\,^{>}\mathsf{pop})\,A)\twoheadrightarrow \\
 & & \mathsf{cnt}\,(\mathsf{vlam}\,^{>}E)\twoheadrightarrow \\
 & & \mathsf{evalt}\,T\,T'\twoheadrightarrow \\
 & & \mathsf{evale}_{St}\,(\mathsf{kapp}\,^{>}\mathsf{ret}\,^{>}T)\,A.
\end{array}$$

$$\begin{array}{lcl}
\mathsf{eve}_{St}\_2 & : & \Pi A{:}\mathsf{triv}.\ \Pi T{:}\mathsf{triv}.\ \Pi T'{:}\mathsf{triv}.\ \Pi E{:}\mathsf{triv} \twoheadrightarrow \mathsf{exp}. \\
 & & (\mathsf{var}\,T' \twoheadrightarrow \mathsf{evale}_{St}\,(E\,^{>}\mathsf{pop})\,A)\twoheadrightarrow \\
 & & \mathsf{evalt}\,T\,T'\twoheadrightarrow \\
 & & \mathsf{evale}_{St}\,(\mathsf{kapp}\,^{>}(\mathsf{vlam}\,^{>}E)\,^{>}T)\,A.
\end{array}$$

$$\begin{array}{lcl}
\mathsf{eve}_{St}\_\mathsf{app}\_0 & : & \Pi A{:}\mathsf{triv}.\ \Pi T_0{:}\mathsf{triv}.\ \Pi T_1{:}\mathsf{triv}.\ \Pi T{:}\mathsf{triv}.\ \Pi E{:}\mathsf{triv} \rightarrow \mathsf{cont} \twoheadrightarrow \mathsf{exp}. \\
 & & \mathsf{evale}_{St}\,((E\,T)\,^{>}\mathsf{ret})\,A\twoheadrightarrow \\
 & & \mathsf{evalt}\,T_0\,(\mathsf{lam}\ \lambda x{:}\mathsf{triv}.\ \mathsf{klam}\,(E\,x))\twoheadrightarrow \\
 & & \mathsf{evalt}\,T_1\,T\twoheadrightarrow \\
 & & \mathsf{evale}_{St}\,(\mathsf{app}\,^{>}\mathsf{ret}\,^{>}T_0\,^{>}T_1)\,A.
\end{array}$$

$$\begin{array}{lcl}
\mathsf{eve}_{St}\_\mathsf{app}\_1 & : & \Pi A{:}\mathsf{triv}.\ \Pi T_0{:}\mathsf{triv}.\ \Pi T_1{:}\mathsf{triv}.\ \Pi E'{:}\mathsf{triv} \twoheadrightarrow \mathsf{exp}.\ \Pi T{:}\mathsf{triv}. \\
 & & \Pi E{:}\mathsf{triv} \rightarrow \mathsf{cont} \twoheadrightarrow \mathsf{exp}. \\
 & & (\mathsf{cnt}\,(\mathsf{vlam}\,^{>}E') \twoheadrightarrow \mathsf{evale}_{St}\,((E\,T)\,^{>}\mathsf{ret})\,A)\twoheadrightarrow \\
 & & \mathsf{evalt}\,T_0\,(\mathsf{lam}\ \lambda x{:}\mathsf{triv}.\ \mathsf{klam}\,(E\,x))\twoheadrightarrow \\
 & & \mathsf{evalt}\,T_1\,T\twoheadrightarrow \\
 & & \mathsf{evale}_{St}\,(\mathsf{app}\,^{>}(\mathsf{vlam}\,^{>}E')\,^{>}T_0\,^{>}T_1)\,A.
\end{array}$$

$$\begin{array}{lcl}
\mathsf{evt\_lam} & : & \Pi R{:}\mathsf{triv} \rightarrow \mathsf{root}.\ \mathsf{evalt}\,(\mathsf{lam}\ R)\,(\mathsf{lam}\ R).
\end{array}$$

$$\begin{array}{lcl}
\mathsf{evt\_vp} & : & \Pi T{:}\mathsf{triv}.\ \mathsf{var}\,T \twoheadrightarrow \mathsf{evalt}\,\mathsf{pop}\,T.
\end{array}$$

Note that this representation does not contain an explicit stack. Instead, the ordered context of the type theory implicitly provides the representation of the evaluation machine's stack. In order to prove our representations are in bijective correspondence to stack evaluations, we need the following auxiliary definitions.

For any term u, $|\ulcorner u \urcorner|$ denotes $\ulcorner u \urcorner$ with all free k replaced by ret and all free v replaced by pop.

We define validity for evaluation stacks, ϕ, with respect to Φ (as defined in section 16.1) as follows:

$$\frac{}{\models^{E} \cdot : k} \qquad \frac{\cdot \models^{\mathbf{Triv}} t \qquad \models^{E} \phi : \Phi}{\models^{E} (\phi, t) : (\Phi, v)} \qquad \frac{\models^{E} \phi : (\Phi, k', \Phi_c) \qquad (k', \Phi_c) \models^{\mathbf{cont}} c}{\models^{E} (\phi, c) : (\Phi, k', \Phi_c, k)}$$

$$\frac{}{\models^{T} \cdot : \cdot} \qquad \frac{\cdot \models^{\mathbf{Triv}} t \qquad \models^{T} \phi : \Phi}{\models^{T} (\phi, t) : (\Phi, v)}$$

Finally we need a representation function, and its inverse, for evaluation stacks.

$$\begin{array}{lcl@{\qquad\qquad}lcl}
\ulcorner . \urcorner & = & \cdot & \llcorner \cdot \lrcorner & = & \cdot \\
\ulcorner (\phi, t) \urcorner & = & \ulcorner \phi \urcorner, vv{:}\mathsf{var}\, \ulcorner t \urcorner & \llcorner \Omega, vv{:}\mathsf{var}\, T \lrcorner & = & \llcorner \Omega \lrcorner, \llcorner T \lrcorner. \\
\ulcorner (\phi, c) \urcorner & = & \ulcorner \phi \urcorner, cv{:}\mathsf{cnt}\, |\ulcorner c \urcorner| & \llcorner \Omega, cv{:}\mathsf{cnt}\, C \lrcorner & = & \llcorner \Omega \lrcorner, \llcorner C \lrcorner_{\Phi}
\end{array}$$

where we can construct Φ from the form of Ω as follows:
either $\Omega = \Omega', cv{:}\mathsf{cnt}\, C', \Omega_C$ and we construct Φ such that $\models^{E} \llcorner \Omega_C \lrcorner : \Phi$;
or $cv{:}\mathsf{cnt}\, C' \notin \Omega$ and we construct Φ such that $\models^{E} \llcorner \Omega \lrcorner : \Phi$.

Theorem 112 (Stack evaluations are canonical forms) *Assume all r, e, and t have no free x.*

1. $\vdash^{\mathbf{Root}}_{St} r \hookrightarrow a$ *and* $\models^{\mathbf{Root}} r$ *implies* $\exists M.\ \ \cdot; \cdot \vdash M \Uparrow \mathsf{evalr}_{St}\, \ulcorner r \urcorner \ulcorner a \urcorner$.

2. $\phi \vdash^{\mathbf{Exp}}_{St} e \hookrightarrow a$ *and* $\models^{E} \phi : (\Phi, k, \Phi_e)$ *(where* $\diamond \notin \Phi_e$*) and* $(k, \Phi_e) \models^{\mathbf{Exp}} e$ *implies*

   $\exists M.\ \ \cdot; \ulcorner \phi \urcorner \vdash M \Uparrow \mathsf{evale}_{St}\, |\ulcorner e \urcorner|\, \ulcorner a \urcorner$.

3. $\phi, \phi_t \vdash^{\mathbf{Triv}}_{St} t \hookrightarrow a; \phi$ *and* $\models^{T} \phi_t : \Phi_t$ *and* $\Phi_t \models^{\mathbf{Triv}} t$ *implies*

   $\exists M.\ \ \cdot; \ulcorner \phi_t \urcorner \vdash M \Uparrow \mathsf{evalt}\, |\ulcorner t \urcorner|\, \ulcorner a \urcorner$.

Proof: By structural induction on the given derivations. In pt. 2, derivations of $\models^{T} \phi : \Phi$ needed to apply the induction hypthesis are constructed by case analysis on the relevant trivial term.

□

Theorem 113 (Canonical forms are stack evaluations)

1. $\cdot;\cdot;\cdot \vdash M \Uparrow \mathsf{evalr}_{St}\, R\, A$ *implies* $\vdash^{\mathbf{Root}}_{St} \llcorner R \lrcorner \hookrightarrow \llcorner A \lrcorner.$.
2. $\cdot;\cdot;\Omega \vdash M \Uparrow \mathsf{evale}_{St}\, E\, A$ *and* $\models^{E} \llcorner \Omega \lrcorner : \Phi$ *implies* $\llcorner \Omega \lrcorner \vdash^{\mathbf{Exp}}_{St} \llcorner E \lrcorner_{\Phi} \hookrightarrow \llcorner A \lrcorner.$.
3. $\cdot;\cdot;\Omega \vdash M \Uparrow \mathsf{evalt}_{St}\, T\, A$ *and* $\models^{T} \llcorner \Omega \lrcorner : \Phi$ *implies* $\phi', \llcorner \Omega \lrcorner \vdash^{\mathbf{Exp}}_{St} \llcorner T \lrcorner_{\Phi} \hookrightarrow \llcorner A \lrcorner.; \phi'$ *for any* ϕ'.

Proof: By structural induction on the given canonical derivations. □

16.6 Bare and Stack Equivalence

In this section we show that the two evaluation models produce the same result. We will carry out our proof inside OLF. Note that this is different from formally representing a proof in OLF; instead we give an informal proof which happens to use OLF. Danvy and Pfenning give a similar proof in [15] which acts directly on the bare and stack machines. We note that carrying out this analysis on OLF representations does not simplify the proof. However, it does show that OLF is expressive enough to represent the main relation of Danvy and Pfenning's proof as a type family. Furthermore, this representation has a direct computational interpretation as a logic program following the operational interpretation of LF type families as logic programs [43].

We first define a relation which holds when a term evaluated by the stack machine and a term evaluated by the bare machine will both evaluate to the same answer. We define this relation as four OLF type families.

$$\mathsf{trans\_r} : \mathsf{root} \to \mathsf{root} \to \mathsf{type}. \qquad \mathsf{trans\_e} : \mathsf{exp} \to \mathsf{exp} \to \mathsf{type}.$$

$$\mathsf{trans\_t} : \mathsf{triv} \to \mathsf{triv} \to \mathsf{type}. \qquad \mathsf{trans\_c} : \mathsf{cont} \to \mathsf{cont} \to \mathsf{type}.$$

The first argument to each type family will be a term partially evaluated by the stack machine, and the second argument will be a term partially evaluated by the bare machine. The term constructors for these families specify when partially evaluated terms are related. Partially evaluated terms are related if they are the same; pop is related to a trivial term t if $\mathsf{var}\, t'$ is the rightmost ordered hypothesis and t' is related to t; similarly, ret is related to a continuation term c if $\mathsf{cnt}\, c'$ is the rightmost ordered hypothesis and c' is related to c.

$$\begin{array}{lcl}
\mathsf{trans\_klam} & : & \Pi E{:}\mathsf{cont} \twoheadrightarrow \mathsf{exp}.\ \Pi E'{:}\mathsf{cont} \twoheadrightarrow \mathsf{exp}.\\
 & & (\Pi k{:}\mathsf{cont}.\ \mathsf{trans\_c}\,k\,k \twoheadrightarrow \mathsf{trans\_e}\,(E\,^{>}k)\,(E'\,^{>}k)) \rightarrow\\
 & & \mathsf{trans\_r}\,(\mathsf{klam}\,E)\,(\mathsf{klam}\,E')
\end{array}$$

$$\begin{array}{lcl}
\mathsf{trans\_app} & : & \Pi C{:}\mathsf{cont}.\ \Pi C'{:}\mathsf{cont}.\ \Pi T_0{:}\mathsf{triv}.\ \Pi E{:}\mathsf{triv} \rightarrow \mathsf{cont} \twoheadrightarrow \mathsf{exp}.\ \Pi T_1{:}\mathsf{triv}.\ \Pi T_1'{:}\mathsf{triv}.\\
 & & \mathsf{trans\_c}\,C\,C' \twoheadrightarrow\\
 & & \mathsf{trans\_t}\,T_0\,(\mathsf{lam}\,\lambda x{:}\mathsf{triv}.\ \mathsf{klam}\,\overset{>}{\lambda} k{:}\mathsf{cont}.\ E\,x\,^{>}k) \twoheadrightarrow\\
 & & \mathsf{trans\_t}\,T_1\,T_1' \twoheadrightarrow\\
 & & \mathsf{trans\_e}\,(\mathsf{app}\,^{>}C\,^{>}T_0\,^{>}T_1)\,(\mathsf{app}\,^{>}C'\,^{>}(\mathsf{lam}\,\lambda x{:}\mathsf{triv}.\ \mathsf{klam}\,\overset{>}{\lambda} k{:}\mathsf{cont}.\ E\,x\,^{>}k)\,^{>}T_1')
\end{array}$$

$$\begin{array}{lcl}
\mathsf{trans\_kapp} & : & \Pi C{:}\mathsf{cont}.\ \Pi C'{:}\mathsf{cont}.\ \Pi T{:}\mathsf{triv}.\ \Pi T'{:}\mathsf{triv}.\\
 & & \mathsf{trans\_c}\,C\,C' \twoheadrightarrow\\
 & & \mathsf{trans\_t}\,T\,T' \twoheadrightarrow\\
 & & \mathsf{trans\_e}\,(\mathsf{kapp}\,^{>}C\,^{>}T)\,(\mathsf{kapp}\,^{>}C'\,^{>}T')
\end{array}$$

$$\begin{array}{lcl}
\mathsf{trans\_lam} & : & \Pi R{:}\mathsf{triv} \rightarrow \mathsf{root}.\ \Pi R'{:}\mathsf{triv} \rightarrow \mathsf{root}.\\
 & & (\Pi x{:}\mathsf{triv}.\ \mathsf{trans\_t}\,x\,x \rightarrow \mathsf{trans\_r}\,(R\,x)\,(R'\,x)) \rightarrow\\
 & & \mathsf{trans\_t}\,(\mathsf{lam}\,R)\,(\mathsf{lam}\,R')
\end{array}$$

$$\begin{array}{lcl}
\mathsf{trans\_pop} & : & \Pi T{:}\mathsf{triv}.\\
 & & \mathsf{trans\_t}\,T\,T \rightarrow\\
 & & \mathsf{var}\,T \twoheadrightarrow\\
 & & \mathsf{trans\_t}\,\mathsf{pop}\,T
\end{array}$$

$$\begin{array}{lcl}
\mathsf{trans\_vlam} & : & \Pi E{:}\mathsf{triv} \twoheadrightarrow \mathsf{exp}.\ \Pi E'{:}\mathsf{triv} \twoheadrightarrow \mathsf{exp}.\\
 & & (\Pi v{:}\mathsf{triv}.\ \mathsf{trans\_t}\,v\,v \twoheadrightarrow \mathsf{trans\_e}\,(E\,^{>}v)\,(E'\,^{>}v)) \twoheadrightarrow\\
 & & \mathsf{trans\_c}\,(\mathsf{vlam}\,^{>}E)\,(\mathsf{vlam}\,^{>}E')
\end{array}$$

$$\begin{array}{lcl}
\mathsf{trans\_ret} & : & \Pi E{:}\mathsf{triv} \twoheadrightarrow \mathsf{exp}.\ \Pi E'{:}\mathsf{triv} \twoheadrightarrow \mathsf{exp}.\\
 & & \mathsf{trans\_c}\,(\mathsf{vlam}\,^{>}E')\,(\mathsf{vlam}\,^{>}E) \twoheadrightarrow\\
 & & \mathsf{cnt}\,(\mathsf{vlam}\,^{>}E') \twoheadrightarrow\\
 & & \mathsf{trans\_c}\,\mathsf{ret}\,(\mathsf{vlam}\,^{>}E).
\end{array}$$

$$\begin{array}{lcl}
\mathsf{trans\_init} & : & \mathsf{trans\_c}\,\mathsf{ret}\,\mathsf{ret}
\end{array}$$

In the rest of this section, all the outermost arguments explicit in the preceding constructors are left implicit.

We next prove some elementary properties of this relation, or set of type families, which are necessary to prove the main result, Theorem 119, which states that two related terms evaluate, using their respective machines, to the same answer.

For any OLF quasi canonical term Q whose free variables of type triv range over $x_1 \ldots x_n$ and $v_1 \ldots v_m$ we have the following definitions:

$$\Gamma^Q_x = x_1{:}\mathsf{triv}, \ldots, x_n{:}\mathsf{triv} \qquad \Gamma^Q_{xx} = \mathsf{trans\_t}\, x_1\, x_1, \ldots, \mathsf{trans\_t}\, x_n\, x_n$$

$$\Gamma^Q_v = v_1{:}\mathsf{triv}, \ldots, v_m{:}\mathsf{triv} \qquad \Omega^Q_v = \mathsf{trans\_t}\, v_1\, v_1, \ldots, \mathsf{trans\_t}\, v_m\, v_m$$

Our first lemma states that if two terms are related in a context which does not contain variables of type $\mathsf{var}\, t$ nor of type $\mathsf{cnt}\, c$, then the two terms are syntactically equal.

Lemma 114

1. $\Gamma^R_x, \Gamma^R_{xx}; \cdot \vdash Q \Uparrow \mathsf{trans\_r}\, R\, R'$ *implies* $R = R'$.
2. $\Gamma^E_x, \Gamma^E_{xx}, k{:}\mathsf{cont}, \Gamma^E_v; \mathsf{trans\_c}\, k\, k, \Omega^E_v \vdash Q \Uparrow \mathsf{trans\_e}\, E\, E'$ *implies* $E = E'$.
3. $\Gamma^T_x, \Gamma^T_{xx}, \Gamma^T_v; \Omega^T_v \vdash Q \Uparrow \mathsf{trans\_t}\, T\, T'$ *implies* $T = T'$.
4. $\Gamma^C_x, \Gamma^C_{xx}, k{:}\mathsf{cont}, \Gamma^C_v; \mathsf{trans\_c}\, k\, k, \Omega^C_v \vdash Q \Uparrow \mathsf{trans\_c}\, C\, C'$ *implies* $C = C'$.

Proof: By induction on the given derivation. □

We now need a lemma which shows that terms are related to themselves.

Lemma 115

1. $\Gamma^R_x; \cdot \vdash R \Uparrow \mathsf{root}$ *implies* $\exists Q.\ \Gamma^R_x, \Gamma^R_{xx}; \cdot \vdash Q \Uparrow \mathsf{trans\_r}\, R\, R$.
2. $\Gamma^E_x; k{:}\mathsf{cont}, \Gamma^E_v \vdash E \Uparrow \mathsf{exp}$ *implies* $\exists Q.\ \Gamma^E_x, \Gamma^E_{xx}, k{:}\mathsf{cont}, \Gamma^E_v; \mathsf{trans\_c}\, k\, k, \Omega^E_v \vdash Q \Uparrow \mathsf{trans\_e}\, E\, E$.
3. $\Gamma^T_x; \Gamma^T_v \vdash T \Uparrow \mathsf{triv}$ *implies* $\exists Q.\ \Gamma^T_x, \Gamma^T_{xx}, \Gamma^T_v; \Omega^T_v \vdash Q \Uparrow \mathsf{trans\_t}\, T\, T$.
4. $\Gamma^C_x; k{:}\mathsf{cont}, \Gamma^C_v \vdash C \Uparrow \mathsf{cont}$ *implies* $\exists Q.\ \Gamma^C_x, \Gamma^C_{xx}, k{:}\mathsf{cont}, \Gamma^C_v; \mathsf{trans\_c}\, k\, k, \Omega^C_v \vdash Q \Uparrow \mathsf{trans\_c}\, C\, C$.

Proof: By induction on the given derivation. □

The next lemma contains two technical results. Trivial terms partially evaluated by the bare machine are always related to themselves in an empty ordered context since they contain no pop and only one ret (for the initial continuation). It also shows that the information in the evaluation stack can be used to further evaluate a partially stack evaluated trivial term to a bare evaluated trivial term, provided the two terms are related.

Lemma 116

1. $\Gamma; \Omega \vdash Q \Uparrow \mathsf{trans\_t}\, T\, T'$ *implies* $\exists Q'.\ \Gamma; \cdot \vdash Q' \Uparrow \mathsf{trans\_t}\, T'\, T'$
2. $\cdot; \Omega \vdash Q \Uparrow \mathsf{trans\_t}\, T\, T'$ *implies* $\exists Q'.\ \cdot; \Omega \vdash Q' \Uparrow \mathsf{evalt}_{St}\, T\, T'$

Proof: By induction on the given derivation. □

The next two lemmas are essentially substitution lemmas for stack "variables". In the constructors for deciding if two partially evaluated terms are related, recursive descent into functional terms (the $\mathsf{trans\_klam}$, $\mathsf{trans\_lam}$, and $\mathsf{trans\_vlam}$) is achieved, by generating a new parameter and applying it to both terms under the assumption that the parameter is related to itself. The next lemmas state that we may substitute actual terms into such parameters.

Lemma 117 *In the following assume*
$\Gamma = x_1{:}\mathsf{triv}, m_1{:}\mathsf{trans\_t}\, x_1\, x_1, \ldots, x_n{:}\mathsf{triv}, m_n{:}\mathsf{trans\_t}\, x_n\, x_n, k{:}\mathsf{cont}, v_1{:}\mathsf{triv}, \ldots, v_m{:}\mathsf{triv}$*;* *and that* Ω_R *only contains identifiers of types* $\mathsf{var}\, X$ *and at most one identifier of type* $\mathsf{trans\_t}\, v_i\, v_i$ *for each* v_i *in* Γ*.*

1. $\Gamma, v{:}\mathsf{triv}; \Omega_L, m{:}\mathsf{trans\_t}\, v\, v, \Omega_R \vdash Q \Uparrow \mathsf{trans\_e}\, E\, E'$ *and* $\cdot; \cdot \vdash R \Uparrow \mathsf{trans\_t}\, T\, T$
   implies $\exists Q'.\ \Gamma; \Omega_L, m'{:}\mathsf{var}\, T, \Omega_R \vdash Q' \Uparrow \mathsf{trans\_e}\, (E[\mathsf{pop}/v])\, (E'[T/v])$
2. $\Gamma, v{:}\mathsf{triv}; \Omega_L, m{:}\mathsf{trans\_t}\, v\, v, \Omega_R \vdash Q \Uparrow \mathsf{trans\_c}\, C\, C'$ *and* $\cdot; \cdot \vdash R \Uparrow \mathsf{trans\_t}\, T\, T$
   implies $\exists Q'.\ \Gamma; \Omega_L, m'{:}\mathsf{var}\, T, \Omega'_R \vdash Q' \Uparrow \mathsf{trans\_c}\, (C[\mathsf{pop}/v])\, (C'[T/v])$

Proof: By induction on the major given derivation.
Assume $\cdot; \cdot \vdash R \Uparrow \mathsf{trans\_t}\, A\, A$.

- case:

  $\Gamma, v{:}\mathsf{triv}; \Omega_L, m{:}\mathsf{trans\_t}\,v\,v, \Omega_R \vdash \mathsf{trans\_kapp}\,{}^{>}R_C\,{}^{>}R_T \Uparrow \mathsf{trans\_e}\,(\mathsf{kapp}\,{}^{>}C\,{}^{>}T)\,(\mathsf{kapp}\,{}^{>}C'\,{}^{>}T')$

  Note that m must be consumed by either R_C or R_T.

  - subcase:

    $\Gamma, v{:}\mathsf{triv}; \Omega_L, m{:}\mathsf{trans\_t}\,v\,v, \Omega_{RL} \vdash R_C \Uparrow \mathsf{trans\_c}\,C\,C'$ and $\Omega_R = \Omega_{RL}\Omega_{RR}$

    Note v cannot occur free in T or T' $\quad$ $\mathsf{trans\_t}\,v\,v \notin \Gamma, \Omega_{RR}$
    $\Gamma; \Omega_L, m'{:}\mathsf{var}\,A, \Omega_{RL} \vdash R'_C \Uparrow \mathsf{trans\_c}\,(C[\mathsf{pop}/v])\,(C'[A/v])$
    for some R'_C $\quad$ ind. hyp.
    $\Gamma; \Omega_L, m'{:}\mathsf{var}\,A, \Omega_R \vdash \mathsf{trans\_kapp}\,{}^{>}R'_C\,{}^{>}R_T \Uparrow$
    $\mathsf{trans\_e}\,(\mathsf{kapp}\,{}^{>}(C[\mathsf{pop}/v])\,{}^{>}T)\,(\mathsf{kapp}\,{}^{>}(C'[A/v])\,{}^{>}T')$ $\quad$ Π_E

  - subcase:

    $\Gamma, v{:}\mathsf{triv}; \Omega_{LR}, m{:}\mathsf{trans\_t}\,v\,v, \Omega_R \vdash R_T \Uparrow \mathsf{trans\_t}\,T\,T'$ and $\Omega_L = \Omega_{LL}\Omega_{LR}$

    $\Omega_{LR} = \cdot = \Omega_R$ and $R_T = m$ and $T = v = T'$ $\quad$ inversion
    $\Gamma; m'{:}\mathsf{var}\,A \vdash \mathsf{trans\_pop}\,{}^{>}R\,{}^{>}m' \Uparrow \mathsf{trans\_t}\,\mathsf{pop}\,A$ $\quad$ Π_E
    Note we can remove $v{:}\mathsf{triv}$ since v no longer occurs free in goal.
    $\Gamma; \Omega_{LL}, m'{:}\mathsf{var}\,A \vdash \mathsf{trans\_kapp}\,{}^{>}R_C\,{}^{>}(\mathsf{trans\_pop}\,{}^{>}R\,{}^{>}m') \Uparrow$
    $\mathsf{trans\_e}\,(\mathsf{kapp}\,{}^{>}C\,{}^{>}\mathsf{pop})\,(\mathsf{kapp}\,{}^{>}C'\,{}^{>}A)$ $\quad$ Π_E

- case: $\mathsf{trans\_app}$ is similar.

- case:

  $\Gamma, v{:}\mathsf{triv}; \Omega_L, m{:}\mathsf{trans\_t}\,v\,v, \Omega_R \vdash \mathsf{trans\_vlam}\,{}^{>}(\lambda v'{:}\mathsf{triv}.\ \overset{>}{\lambda} m'{:}\mathsf{trans\_t}\,v'\,v'.\ R_E) \Uparrow$
  $\mathsf{trans\_c}\,(\mathsf{vlam}\,{}^{>}\overset{>}{\lambda} v'{:}\mathsf{triv}.\ E)\,(\mathsf{vlam}\,{}^{>}\overset{>}{\lambda} v'{:}\mathsf{triv}.\ E')$

  $\Gamma, v{:}\mathsf{triv}, v'{:}\mathsf{triv}; \Omega_L, m{:}\mathsf{trans\_t}\,v\,v, \Omega_R, m'{:}\mathsf{trans\_t}\,v'\,v' \vdash R_E \Uparrow$
  $\mathsf{trans\_e}\,E\,E'$ $\quad$ inversion
  $\Gamma, v'{:}\mathsf{triv}; \Omega_L, m''{:}\mathsf{var}\,A, \Omega_R, m'{:}\mathsf{trans\_t}\,v'\,v' \vdash R'_E \Uparrow$
  $\mathsf{trans\_e}\,(E[\mathsf{pop}/v])\,(E'[A/v])$ for some R'_E $\quad$ ind. hyp.
  $\Gamma; \Omega_L, m''{:}\mathsf{var}\,A, \Omega_R \vdash \mathsf{trans\_vlam}\,{}^{>}(\lambda v'{:}\mathsf{triv}.\ \overset{>}{\lambda} m'{:}\mathsf{trans\_t}\,v'\,v'.\ R'_E) \Uparrow$
  $\mathsf{trans\_c}\,(\mathsf{vlam}\,{}^{>}\overset{>}{\lambda} v'{:}\mathsf{triv}.\ E[\mathsf{pop}/v])\,(\mathsf{vlam}\,{}^{>}\overset{>}{\lambda} v'{:}\mathsf{triv}.\ E'[A/v])$ $\quad$ Π_E

- No other cases.

□

Lemma 118 *In the following assume*
$\Gamma = x_1{:}\mathsf{triv}, m_1{:}\mathsf{trans\_t}\, x_1\, x_1, \ldots, x_n{:}\mathsf{triv}, m_n{:}\mathsf{trans\_t}\, x_n\, x_n, v_1{:}\mathsf{triv}, \ldots, v_m{:}\mathsf{triv}$*;*
and that Ω *only contains identifiers of types* $\mathsf{var}\, X$ *and at most one identifier of type* $\mathsf{trans\_t}\, v_i\, v_i$ *for each* v_i *in* Γ*.*

1. $\Gamma, k{:}\mathsf{cont}; m{:}\mathsf{trans\_c}\, k\, k, \Omega \vdash Q \Uparrow \mathsf{trans\_e}\, E\, E'$ *and* $\cdot; \Omega' \vdash R \Uparrow \mathsf{trans\_c}\, C_s\, C_b$
   implies $\exists Q'.\ \Gamma; \Omega', m'{:}\mathsf{cnt}\, C_s, \Omega \vdash Q' \Uparrow \mathsf{trans\_e}\, (E[\mathsf{ret}/k])\, (E'[C_b/k])$*.*
2. $\Gamma, k{:}\mathsf{cont}; m{:}\mathsf{trans\_c}\, k\, k, \Omega \vdash Q \Uparrow \mathsf{trans\_c}\, C\, C'$ *and* $\cdot; \Omega' \vdash R \Uparrow \mathsf{trans\_c}\, C_s\, C_b$
   implies $\exists Q'.\ \Gamma; \Omega', m'{:}\mathsf{cnt}\, C_s, \Omega \vdash Q' \Uparrow \mathsf{trans\_c}\, (C[\mathsf{ret}/k])\, (C'[C_b/k])$*.*

Proof: Similar to previous proof. □

We can now prove the main theorem, that related terms evaluate to the same result in both machines.

Theorem 119

1. $\exists R.\ \cdot; \vdash R \Uparrow \mathsf{trans\_r}\, Rt\, Rt'$ *implies*
   $\exists Q.\ \cdot; \cdot \vdash Q \Uparrow \mathsf{evalr}_{St}\, Rt\, A$ *iff* $\exists Q'.\ \cdot; \cdot \vdash Q' \Uparrow \mathsf{evalr}_B\, Rt'\, A$*.*
2. $\exists R.\ \cdot; \Omega \vdash R \Uparrow \mathsf{trans\_e}\, E\, E'$ *implies*
   $\exists Q.\ \cdot; \Omega \vdash Q \Uparrow \mathsf{evale}_{St}\, E\, A$ *iff* $\exists Q'.\ \cdot; \cdot \vdash Q' \Uparrow \mathsf{evale}_B\, E'\, A$*.*

Proof: By induction on the given evaluation derivation.
Note that R determines which evaluation is being considered.

- case:
  $$\begin{aligned} \cdot; \cdot \vdash\ & \mathsf{trans\_klam}\, (\lambda k{:}\mathsf{cont}.\ \overset{>}{\lambda} m{:}\mathsf{trans\_c}\, k\, k.\ R_E) \Uparrow \\ & \mathsf{trans\_r}\, (\mathsf{klam}\, \overset{>}{\lambda} k{:}\mathsf{cont}.\ E)\, (\mathsf{klam}\, \overset{>}{\lambda} k{:}\mathsf{cont}.\ E) \end{aligned}$$

  $k{:}\mathsf{cont}; m{:}\mathsf{trans\_c}\, k\, k \vdash R_E \Uparrow \mathsf{trans\_e}\, E\, E'$ inversion
  $\cdot; \cdot \vdash R_E[\mathsf{ret}/k][\mathsf{trans\_init}/m] \Uparrow \mathsf{trans\_e}\, (E[\mathsf{ret}/k])\, (E'[\mathsf{ret}/k])$ Lemma 95

  Suppose $\cdot; \cdot \vdash \mathsf{evr}_{St}\, Q_E \Uparrow \mathsf{evalr}_{St}\, (\mathsf{klam}\, \overset{>}{\lambda} k{:}\mathsf{cont}.\ E)\, A$.
  $\cdot; \cdot \vdash Q_E \Uparrow \mathsf{evale}_{St}\, (E[\mathsf{ret}/k])\, A$ inversion
  $\exists Q'_E\ \cdot; \cdot \vdash Q'_E \Uparrow \mathsf{evale}_B\, (E'[\mathsf{ret}/k])\, A$ ind. hyp.

$\cdot;\cdot \vdash \mathsf{evr}_B\, Q'_E \Uparrow \mathsf{evalr}_B\, (\mathsf{klam}\ \overset{>}{\lambda} k{:}\mathsf{cont}.\ E')\, A$ $\qquad \Pi_E$

Other direction is similar.

- case:

$$\cdot; \Omega_C \Omega_T \vdash \mathsf{trans\ \ \_kapp}^{>} R_c {}^{>} R_T \Uparrow \mathsf{trans\_c}\, (\mathsf{kapp}^{>} C^{>} T)\, (\mathsf{kapp}^{>} C'^{>} T')$$

$\cdot; \Omega_T \vdash R_T \Uparrow \mathsf{trans\_t}\, T\, T'$ $\qquad$ inversion

$\exists Q_T, R_{T'}.\quad \cdot; \Omega_T \vdash Q_T \Uparrow \mathsf{evalt}_{St}\, T\, T'$ and

$\quad \cdot;\cdot \vdash R_{T'} \Uparrow \mathsf{trans\_t}\, T'\, T'$ $\qquad$ Lemma 116

  - subcase: $C = \mathsf{ret}$ and $\Omega_C = \cdot$

    $C' = \mathsf{ret}$ and $R_C = \mathsf{trans\_init}$ $\qquad$ inversion

    $\cdot; \Omega_T \vdash \mathsf{eve\_0}_{St}{}^{>} Q_T \Uparrow \mathsf{evale}_{St}\, (\mathsf{kapp}^{>} \mathsf{ret}^{>} T)\, T'$ and

    $\quad \cdot;\cdot \vdash \mathsf{eve\_0}_B \Uparrow \mathsf{evale}_B\, (\mathsf{kapp}^{>} \mathsf{ret}^{>} T')\, T'$ $\qquad \Pi_E$

  - subcase: $C = \mathsf{ret}$ and $\Omega_C \neq \cdot$

    $C' = \mathsf{vlam}^{>} \overset{>}{\lambda} v{:}\mathsf{triv}.\ E'$ and

    $\Omega_C = \Omega_E, m{:}\mathsf{cnt}\, (\mathsf{vlam}^{>} \overset{>}{\lambda} v{:}\mathsf{triv}.\ E)$ and

    $R_C = \mathsf{trans\_ret}^{>} (\mathsf{trans\_vlam}^{>} \lambda v{:}\mathsf{triv}.\ \overset{>}{\lambda} m_v{:}\mathsf{trans\_t}\, v\, v.\ R_E)^{>} m$ and

    $v{:}\mathsf{triv}; \Omega_E, m_v{:}\mathsf{trans\_t}\, v\, v \vdash R_E \Uparrow \mathsf{trans\_e}\, E\, E'$ $\qquad$ inversion

    $\cdot; \Omega_E, m{:}\mathsf{cnt}\, (\mathsf{vlam}^{>} \overset{>}{\lambda} v{:}\mathsf{triv}.\ E) \vdash \mathsf{eve\_1}_{St}{}^{>} (\overset{>}{\lambda} m_{T'}{:}\mathsf{var}\, T'.\ Q_E)^{>} m^{>} Q_T \Uparrow$

    $\quad \mathsf{evale}_{St}\, (\mathsf{kapp}^{>} \mathsf{ret}^{>} T)\, A$ $\qquad$ new assumption

    $\cdot; \Omega_E, m_{T'}{:}\mathsf{var}\, T' \vdash Q_E \Uparrow \mathsf{evale}_{St}\, (E[\mathsf{pop}/v])\, A$ $\qquad$ inversion

    $\exists R'_E.\quad \cdot; \Omega_E, m_{T'}{:}\mathsf{var}\, T' \vdash R'_E \Uparrow \mathsf{trans\ \ \ \_e}(E[\mathsf{pop}/v])\, (E'[T'/v)$ $\qquad$ Lemma 117

    $\exists Q'_E.\quad \cdot;\cdot \vdash Q'_E \Uparrow \mathsf{evale}_B\, (E'[T'/v])\, A$ $\qquad$ ind. hyp.

    $\cdot;\cdot \vdash \mathsf{eve\_1}_B\, Q'_E \Uparrow \mathsf{evale}_B\, (\mathsf{kapp}^{>} (\mathsf{vlam}^{>} \overset{>}{\lambda} v{:}\mathsf{triv}.\ E')^{>} T')\, A$ $\qquad \Pi_E$

    Similar reasoning for the other direction.

  - subcase: $C = \mathsf{vlam}^{>} \overset{>}{\lambda} v{:}\mathsf{triv}.\ E$

    $C' = \mathsf{vlam}^{>} \overset{>}{\lambda} v{:}\mathsf{triv}.\ E'$ and

    $R_C = \mathsf{trans\_vlam}^{>} \lambda v{:}\mathsf{triv}.\ \overset{>}{\lambda} m_v{:}\mathsf{trans\_t}\, v\, v.\ R_E$ $\qquad$ inversion

    Then similar reasoning to previous applies.

- case:

$$\cdot;\Omega_C,\Omega_{T_0},\Omega_{T_1}\vdash \mathsf{trans\_app}^{>}R_C{}^{>}R_{T_0}{}^{>}R_{T_1}\Uparrow$$
$$\mathsf{trans\_e}\,(\mathsf{app}^{>}C^{>}T_0{}^{>}T_1)\,(\mathsf{app}^{>}C'^{>}(\mathsf{lam}\ \lambda x{:}\mathsf{triv}.\ \mathsf{klam}\ \overset{>}{\lambda} k{:}\mathsf{cont}.\ E)^{>}T_1')$$

$\cdot;\Omega_C\vdash R_C\Uparrow\mathsf{trans\_c}\,C\,C'$ and
$\cdot;\Omega_{T_1}\vdash R_{T_1}\Uparrow\mathsf{trans\_t}\,T_1\,T_1'$ and
$\cdot;\Omega_{T_0}\vdash R_{T_0}\Uparrow\mathsf{trans\_t}\,T_0\,(\mathsf{lam}\ \lambda x{:}\mathsf{triv}.\ \mathsf{klam}\ \overset{>}{\lambda} k{:}\mathsf{cont}.\ E)$ inversion
$\cdot;\cdot\vdash R'_{T_1}\Uparrow\mathsf{trans\_t}\,T_1'\,T_1'$ and
$\cdot;\cdot\vdash R'_{T_0}\Uparrow$
$\mathsf{trans\_t}(\mathsf{lam}\ \lambda x{:}\mathsf{triv}.\ \mathsf{klam}\ \overset{>}{\lambda} k{:}\mathsf{cont}.\ E)\,(\mathsf{lam}\ \lambda x{:}\mathsf{triv}.\ \mathsf{klam}\ \overset{>}{\lambda} k{:}\mathsf{cont}.\ E)$
for some R'_{T_0} and R'_{T_1} Lemma 116
$R'_{T_0}=$ inversion
$\mathsf{trans\_lam}\,(\lambda x{:}\mathsf{triv}.\ \lambda m_x{:}\mathsf{trans\_t}\,x\,x.\ \mathsf{trans\_klam}\ \lambda k{:}\mathsf{cont}.\ \overset{>}{\lambda} m_k{:}\mathsf{trans\_c}\,k\,k.\ R_E)$
$x{:}\mathsf{triv},m_x{:}\mathsf{trans\_t}\,x\,x,k{:}\mathsf{cont};m_k{:}\mathsf{trans\_c}\,k\,k\vdash R_E\Uparrow\mathsf{trans\_e}\,E\,E$ inversion
$k{:}\mathsf{cont};m_k{:}\mathsf{trans\_c}\,k\,k\vdash R_E[T_1'/x][R'_{T_1}/m_x]\Uparrow$
$\mathsf{trans\_e}\,(E[T_1'/x])\,E([T_1'/x])$ Lemma 95
Then similar reasoning to previous case,
using both lemma 117 and lemma 118, can be employed to complete the proof.
Note there are 2 subcases, $C=\mathsf{ret}$ and $C=\mathsf{vlam}^{>}\overset{>}{\lambda} v{:}\mathsf{triv}.\ E_v$.

□

Theorem 120 (Equivalence of Machine Representations)
$\exists M.\ \cdot;\cdot\vdash M\Uparrow\mathsf{evalr}_{St}\,R\,A\quad$ *iff* $\quad\exists M'.\ \cdot;\cdot\vdash M'\Uparrow\mathsf{evalr}_B\,R\,A$

Proof: Immediate from previous lemmas and theorems □

We may now use Theorem 111 and Theorem 113 to transfer the result of Theorem 120 back to the informal bare and stack evaluation machines.

16.7 Conclusion

We have shown that an ordered logical framework provides the necessary machinery for a natural encoding of CPS terms satisfying the given occurrence invariants. We have further shown that the framework is rich enough to allow a natural representation of stack-based evaluation. Furthermore we have seen that preservation of CPS invariants under evaluation is then trivial to prove. We feel this in itself is significant considering the difficulty involved in carrying out such representations in a framework with no inherent notion of order. Dzafic [17] has shown how to represent system and properties closely related to ours in LF, with considerable overhead since stacks and the necessary substitution properties all have to be represented explicitly.

Finally, we mention that these techniques are easily extended to represent and reason about other systems which rely upon an ordering of resources. For instance, [54] extends the analysis, and results, of this chapter to a CPS transform which removes explicit exceptions. We also strongly conjecture that both the analysis of evaluation with first class continuations, carried out in [13], and the analysis of information flow using ordered continuations in [62] can be naturally re-formulated in OLF.

Part IV

Conclusions

Chapter 17

Conclusions and Future Work

Ordered linear logic is a new logical formalism which conservatively extends (intuitionistic) linear logic with ordered hypotheses– hypotheses which must be used exactly once subject to the order in which they were assumed. The logic is constructed, in the style of Martin-Löf, from the basic notion of a hypothetical judgement and its associated substitution principle. Specifically, ordered linear logic results from combining three different kinds of hypothetical judgements– unrestricted, linear, and ordered.

Ordered linear logic seems to have several advantages over other logical systems which combine unrestricted, linear and ordered reasoning such as Non-Commutative Logic [1, 57], cyclic linear logic [61], and pomset logic [56]. Since ordered linear logic is intuitionistic and associates structural properties with formulas, rather than contexts[1], it is a simpler system than its counterparts. In fact, most of the proofs of basic results (*e.g.*, cut elimination, normalization, focussing, etc...) in this thesis are straightforward extensions of those for linear logic. Proving similar properties for Non-Commutative Logic has not turned out to be so easy, as evidenced by [5, 18]. Furthermore, since ordered linear logic is intuitionistic, it gives rise to a typed term calculus which can be used as the basis of a logical framework.

While we have presented the basic system of ordered linear logic, there is much room for further exploration. As mentioned in Section 2.7, there are other ways to combine the three modes of reasoning present in the system. It seems quite likely that a logical system which incorporates both formula-level and context-level structural properties exists. This logic could be formulated as a sequent system with three

[1] as described in Section 2.7

context constructors and two modalities.

Another possibility for exploration involves making the ordered context circular. Investigations of intuitionistic logics have typically only considered non-cyclic contexts, however it seems likely that a coherent intuitionistic logical system with a circular context exists. Such a system would be quite different from ordered linear logic, and probably would not be useful for the same applications; however a circular context might also remove certain unsatisfactory situations such as the inability to residuate ordered formulas.

Some other directions for exploration include the addition of a (multiplicative) negation, *i.e.*, $\perp$; and the definition of a formal mathematical semantics for ordered linear logic.

After presenting the basic system of ordered linear logic, we showed that proof search in the logic behaves similarly to proof search in linear logic. In particular, there is a readily identifiable uniform fragment of ordered linear logic which can serve as the basis for a logic programming language, as well as a logical framework. Ordered versions of these linear logic applications afford considerably more elegant solutions than their linear counterparts. Specifically, the ordered context can be used to implicitly represent simple data structures (*e.g.*, stacks, queues), eliminating the need to explicitly construct such structures at the term level.

For logic programming, this expressivity results in more elegant code than is possible using just unrestricted and/or linear hypotheses. Furthermore, shifting some data structures to the logical level (as opposed to the term level) opens up the possibility of more efficient executions as compiler technology for ordered/linear logic programming languages improves. For logical frameworks, it allows representations using higher-order abstract syntax which would not be possible without ordered hypotheses; this in turn makes representation of meta-theoretic properties tractable. For example, using LF (or linear LF), it is not feasible to completely represent the equivalence between lambda terms represented via higher-order abstract syntax and DeBruijn terms. However, the ordered logical framework permits a natural encoding of DeBruijn terms which allows a straightforward encoding of the proof of equivalence using the standard LF methodology.

There are, of course, limitations to what can be represented with the ordered context. So far, we have only really used the ordered context as a stack, or a queue. Furthermore, it only seems possible to have one logical queue or stack in a representation. Thus, an Olli program implementing an algorithm requiring two separate work queues would need to represent one of the queues in the term level. On a slightly different (though probably related) note, writing a meta-circular interpreter in Olli is surprisingly difficult. The very natural approach which works for both (pure) λProlog and Lolli breaks down for Olli. The problem lies in the inability to store information in a formula about its position in the ordered context, which is necessary for the meta-circular interpreter to correctly match the operational behavior of Olli and maintain the ordering constraints. Writing a correct meta-circular interpreter seems to require explicitly creating machinery to maintain the ordering constraints. A similar difficulty arises when trying to formalize a proof of cut-elimination for ordered linear logic in the ordered logical framework. In addition to investigating possible solutions to the above mentioned problems, we would like to expand the range of applications for ordered linear logic, which we feel is largely unexplored.

Finally, we note that a largely ignored aspect of this work is in applications of the ordered lambda calculus. This dissertation has only employed ordered terms in the logical framework setting. It would be interesting to explore direct applications of the ordered lambda calculus.

Bibliography

[1] V. Michele Abrusci and Paul Ruet. Non-commutative logic I: The multiplicative fragment. *Annals of Pure and Applied Logic*, 101(1):29–64, 1999.

[2] J.-M. Andreoli. *Proposal for a Synthesis of Logic and Object-Oriented Programming Paradigms.* PhD thesis, University of Paris VI, 1990.

[3] J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[4] J.-M. Andreoli. Focussing and proof construction. *Annals of Pure and Applied Logic*, 2000. to appear.

[5] Jean-Marc Andreoli and Roberto Maieli. Focusing and proof-nets in linear and non-commutative logic. In Harald Ganzinger, David McAllester, and Andrei Voronkov, editors, *Proceedings of 6th International Conference on Logic Programming and Automated Reasoning*, pages 320–336, Tbilisi, Republic of Georgia, September 1999. Springer-Verlag LNAI 1705.

[6] Henk P. Barendregt. Lambda calculi with types. In S. Abramsky, D. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, chapter 2, pages 117–309. Oxford University Press, 1992.

[7] G. M. Bierman. A note on full intuitionistic linear logic. *Annals of Pure and Applied Logic*, 79(3):281–287, June 1996.

[8] C. Brown and D. Gurr. Relations and non-commutative linear logic. Technical Report DAIMI PB-372, Computer Science Department, Aarhus University, November 1991.

[9] 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.

[10] Iliano Cervesato, Joshua S. Hodas, and Frank Pfenning. Efficient resource management for linear logic proof search. *Theoretical Computer Science*, 232:133–163, 2000. Revised version of paper in the Proceedings of the 5th International Workshop on Extensions of Logic Programming, Leipzig, Germany, March 1996.

[11] Iliano Cervesato and Frank Pfenning. A linear logical framework. *Information and Computation*, 1999. To appear in the special issue with invited papers from LICS'96, E. Clarke, editor.

[12] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[13] Olivier Danvy. Formalizing implementation strategies for first-class continuations. In *Programming Languages and Systems, The Proceedings of the 9th European Symposium on Programming*, volume 1782 of *lncs*, pages 88–103, 2000.

[14] Olivier Danvy, Belmina Dzafic, and Frank Pfenning. On proving syntactic properties of CPS programs. In *Third International Workshop on Higher Order Operational Techniques in Semantics (HOOTS'99)*, Paris, France, September 1999.

[15] Olivier Danvy and Frank Pfenning. The occurrence of continuation parameters in CPS terms. Technical Report CMU-CS-95-121, Department of Computer Science, Carnegie Mellon University, February 1995.

[16] Philippe de Groote. Partially commutative linear logic: sequent calculus and phase semantics. In V.M. Abrusci and C. Casadio, editors, *Proofs and Linguistic Categories, Application of Logic to the Analysis and Implementation of Natural Language Proceedings 1996 Roma Workshop*, pages 199–208. Cooperativa Libraria Universitaria Editrice Bologna, 1996.

[17] Belmina Dzafic. Formalizing program transformations. Master's thesis, DAIMI, Department of Computer Science, University of Aarhus, Aarhus, Denmark, December 1998.

[18] Claudia Faggian. Proof construction and non-commutativity: a cluster calculus. In *Proceedings of the 2nd International ACM SIGPLAN Conference on Prin-*

ciples and Practice of Declarative Programming (PPDP'00), Montreal,Canada, September 2000.

[19] Jean Gallier. On Girard's "candidats de reductibilité". In Odifreddi, editor, *Logic and Computer Science*, pages 123–203. Academic Press, 1990.

[20] Jean Gallier. Constructive logics. Part I: A tutorial on proof systems and typed λ-calculi. *Theoretical Computer Science*, 110(2):249–339, 1993.

[21] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[22] J.A. Harland and David J. Pym. Resource-distribution via boolean constraints. In W. McCune and G. Sutcliffe, editors, *Proc. CADE-14*, pages 222–236. Springer-Verlag LNCS 1249, 1997.

[23] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[24] Robert Harper and Frank Pfenning. On equivalence and canonical forms in the LF type theory. Technical Report CMU-CS-00-148, Department of Computer Science, Carnegie Mellon University, July 2000. An extended abstract appeared at Workshop on Logical Frameworks and Meta-Languages (LFM'99), Paris, France, September 1999.

[25] Hugo Herbelin. *Séquents qu'on calcule.* PhD thesis, Université Paris 7, January 1995.

[26] J. S. Hodas. *Logic Programming in Intuitionistic Linear Logic: Theory, Design and Implementation.* PhD thesis, University of Pennsylvania, Department of Computer and Information Science, 1994.

[27] J. S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994. Extended abstract in the Proceedings of the Sixth Annual Symposium on Logic in Computer Science, Amsterdam, July 15–18, 1991.

[28] Joshua S. Hodas and Naoyuki Tamura. lolliCOP - a linear logic encoding of a lean connection-method theorem prover for first-order classical logic. In *Proceedings of the International Joint Conference on Automated Reasoning*, pages 670–684, Siena, Italy, June 2001.

[29] Martin Hofmann. Linear types and non-size-increasing polynomial time computation. In G. Longo, editor, *Proceedings of the Fourteenth Annual Symposium on Logic in Computer Science*, pages 464–473, Trento, Italy, July 1999. IEEE Computer Society Press.

[30] W.A. Howard. The formula-as-types notion of construction. In J.R. Hindley and J.P. Seldin, editors, *To H.B. Curry, Essays on Combinatory Logic, Lambda Calculus, and Formalism.* Academic Press, 1980.

[31] Twan Laan, Fairouz Kamareddine, and Rob Nederpelt. Refining the Barendregt cube using parameters. In Herbert Kuchen and Kazunori Ueda, editors, *Proceedings of the 5th International Symposium on Functional and Logic Programming (FLOPS'01)*, pages 375–389, Tokyo, Japan, March 2001. Springer-Verlag LNCS 2024.

[32] Joachim Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:363–386, 1958.

[33] Pablo Lopez and Ernesto Pimentel. Resource management in linear logic proof search revisited. In Harald Ganzinger, David McAllester, and Andrei Voronkov, editors, *Proceedings of 6th International Conference on Logic Programming and Automated Reasoning*, pages 304–319, Tbilisi, Republic of Georgia, September 1999. Springer-Verlag LNAI 1705.

[34] Per Martin-Löf. Constructive mathematics and computer programming. In *Logic, Methodology and Philosophy of Science VI*, pages 153–175. North-Holland, 1980.

[35] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

[36] Dale Miller and Gopalan Nadathur. Higher-order logic programming. In Ehud Shapiro, editor, *Proceedings of the Third International Logic Programming Conference*, pages 448–462, London, June 1986.

[37] 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.

[38] Gopalan Nadathur and Dustin J. Mitchell. System description: Teyjus—a compiler and abstract machine based implementation of lambda prolog. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction (CADE-16)*, pages 287–291, Trento, Italy, July 1999. Springer-Verlag LNCS.

[39] P. W. O'Hearn and J. C. Reynolds. From algol to polymorphic linear lambda-calculus. *Journal of the ACM*, 47(1):167–223, January 2000.

[40] P.W. O'Hearn and D. J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, June 1999.

[41] Chris Okasaki. Breadth-first numbering: Lessons from a small exercise in algorithm design. In *Proceedings of International Conference on Functional Programming*, pages 131–136, September 2000.

[42] Remo Pareschi. *Type-Driven Natural Language Analysis.* PhD thesis, University of Edinburgh, Edinburgh, Scotland, July 1989. Available as technical report MS-CIS-89-45, Department of Computer and Information Sciences, University of Pennsylvania.

[43] 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.

[44] Frank Pfenning. Structural cut elimination in linear logic. Technical Report CMU-CS-94-222, Department of Computer Science, Carnegie Mellon University, December 1994.

[45] 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 Press.

[46] Frank Pfenning. The practice of logical frameworks. In Hélène Kirchner, editor, *Proceedings of the Colloquium on Trees in Algebra and Programming*, pages 119–134, Linköping, Sweden, April 1996. Springer-Verlag LNCS 1059. Invited talk.

[47] Frank Pfenning. *Computation and Deduction.* Cambridge University Press, 2001. In preparation. Draft from April 1997 available electronically.

[48] 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.

[49] Jeff Polakow. Linear logic programming with an ordered context. In *Proceedings of the 2nd International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP'00)*, pages 68–79, Montreal,Canada, September 2000.

[50] Jeff Polakow and Frank Pfenning. Ordered linear logic programming. Technical Report CMU-CS-98-183, Department of Computer Science, Carnegie Mellon University, December 1998.

[51] Jeff Polakow and Frank Pfenning. Natural deduction for intuitionistic non-commutative linear logic. In J.-Y. Girard, editor, *Proceedings of the Fourth International Conference on Typed Lambda Calculi and Applications (TLCA'99)*, pages 295–309, l'Aquila, Italy, April 1999. Springer-Verlag LNCS 1581.

[52] Jeff Polakow and Frank Pfenning. Relating natural deduction and sequent calculus for intuitionistic non-commutative linear logic. In Andre Scedrov and Achim Jung, editors, *Proceedings of the 15th Conference on Mathematical Foundations of Programming Semantics*, pages 311–328, New Orleans, Louisiana, April 1999. Electronic Notes in Theoretical Computer Science, Volume 20.

[53] Jeff Polakow and Frank Pfenning. Properties of terms in continuation passing style in an ordered logical framework. In *Workshop on Logical Frameworks and Meta-Languages (LFM 2000)*, Santa Barbara, California, June 2000.

[54] Jeff Polakow and Kwangkeun Yi. Proving syntactic properties of exceptions in an ordered logical framework. In Herbert Kuchen and Kazunori Ueda, editors, *Proceedings of the 5th International Symposium on Functional and Logic Programming (FLOPS'01)*, pages 61–77, Tokyo, Japan, March 2001. Springer-Verlag LNCS 2024.

[55] D. Prawitz. Ideas and results in proof theory. In Jens Erik Fenstad, editor, *Proceedings of the 2nd Scandinavian Logic Symposium*, pages 235–307, North Holland, Amsterdam, June 1970.

[56] Christian Retore. Pomset logic: a non-commutative extension of classical linear logic. In *Proceedings of the Second International Conference on Typed Lambda Calculi and Applications (TLCA'97)*, pages 300–318. Springer-Verlag LNCS 1210, 1997.

[57] P. Ruet. Non-commutative logic II : sequent calculus and phase semantics. *Mathematical Structures in Computer Science*, 10(2):277–312, 2000.

[58] Paul Ruet. *Logique non-commutative et programmation concurrente par contraintes.* PhD thesis, Universite Denis Diderot, Paris 7, 1997.

[59] Joseph Vanderwaart and Karl Crary. A simplified account of the metatheory of linear LF. unpublished manuscript, 2000.

[60] P. Wadler. Linear types can change the world. In M. Broy and C. B. Jones, editors, *IFIP TC 2 Working Conference on Programming Concepts and Methods*, pages 561–581, Sea of Gallilee, Israel, April 1990. North-Holland.

[61] D.N. Yetter. Quantales and (non-commutative) linear logic. *Journal of Symbolic Logic*, 55(1):41–64, 1990.

[62] Steve Zdancewic and Andrew C. Myers. Secure information flow and CPS. In *Proceedings of the 10th European Symposium on Programming (ESOP'01)*, pages 46–61. Springer-Verlag LNCS 2028, April 2001.