Computational Aspects of Linear Logic

The MIT Press Foundations Of Computing Series
Michael Garey and Albert Meyer, editors

Complexity Issues in VLSI: Optimal Layouts for the Shuffle-Exchange Graph and Other Networks
Frank Thomson Leighton, 1983

Equational Logic as a Programming Language
Michael J. O'Donnell, 1985

General Theory of Deductive Systems and Its Applications
S. Yu Maslov, 1987

Resource Allocation Problems: Algorithmic Approaches
Toshihide Ibaraki and Naoki Katoh, 1988

Algebraic Theory of Processes
Michael Hennessy, 1988

PX: A Computational Logic
Susumu Hayashi and Hiroshi Nakano, 1989

The Stable Marriage Problem: Structure and Algorithms
Dan Gusfield and Robert Irving, 1989

Realistic Compiler Generation
Peter Lee, 1989

Single-Layer Wire Routing and Compaction
F. Miller Maley, 1990

Basic Category Theory for Computer Scientists
Benjamin C. Pierce, 1991

Categories, Types, and Structures: An Introduction to Category Theory for the Working Computer Scientist
Andrea Asperti and Giuseppe Longo, 1991

Semantics of Programming Languages: Structures and Techniques
Carl A. Gunter, 1992

The Formal Semantics of Programming Languages: An Introduction
Glynn Winskel, 1993

Hilbert's Tenth Problem
Yuri V. Matiyasevich, 1993

Exploring Interior-Point Linear Programming: Algorithms and Software
Ami Arbel, 1993

Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design
Carl A. Gunter and John C. Mitchell, 1994

From Logic to Logic Programming
Kees Doets, 1994

The Structure of Typed Programming Languages
David A. Schmidt, 1994

Logic and Information Flow
edited by Jan van Eijck and Albert Visser, 1994

Circuit Complexity and Neural Networks
Ian Parberry, 1994

Computational Aspects of Linear Logic
Patrick D. Lincoln, 1994

Computational Aspects of Linear Logic

Patrick Lincoln

The MIT Press
Cambridge, Massachusetts
London, England

Dedicated to R.G.

This book was set in Computer Modern by the author and printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Computatinoal aspects of linear logic / Patrick Lincoln. p cm. – (Foundations of computing series) Includes bibliographical references (p. -) and index. ISBN 0-262-12195-6 (hc : alk. paper) 1. Logic, Symbolic and mathematical–Data processing. 2. Proof theory. I. Title. II. Seires: Foundations of Computing. QA9.L565 1995. 511.3–dc20 95-38239 CIP

Contents

Series Foreword

Theoretical computer science has now undergone several decades of development. The "classical" topics of automata theory, formal languages, and computational complexity have become firmly established, and their importance to other theoretical work and to practice is widely recognized. Stimulated by technological advances, theoreticians have been rapidly expanding the areas under study, and the time delay between theoretical progress and its practical impact has been decreasing dramatically. Much publicity has been given recently to breakthroughs in cryptography and linear programming, and steady progress is being made on programming language semantics, computational geometry, and efficient data structures. Newer, more speculative, areas of study include relational databases, VLSI theory, and parallel and distributed computation. As this list of topics continues expanding, it is becoming more and more difficult to stay abreast of the progress that is being made and increasingly important that the most significant work be distilled and communicated in a manner that will facilitate further research and application of this work. By publishing comprehensive books and specialized monographs on the theoretical aspects of computer science, the series on Foundations of Computing provides a forum in which important research topics can be presented in their entirety and placed in perspective for researchers, students, and practitioners alike.
Michael R. Garey
Albert R. Meyer

Preface

Linear logic was presented by Girard both as a "logic behind logics," and as a "resource conscious logic." The deep meanings behind these slogans is still being uncovered. This book presents investigations into computational aspects of linear logic. The main results of this work support the proposition that linear logic is a *computational* logic behind logics.

This book introduces the reader to linear logic, linear proof theory, and several computational interpretations of linear logic. A reader is assumed to have a passing knowledge of the basic definitions from computational complexity (see [52]), and to be familiar with some standard presentations of logic.

Linear logic has triggered a moderate amount of interest from mathematicians and theoretical computer scientists. From a mathematical point of view, linear logic provides a primitive constructive logic that may be used to study foundational and proof-theoretic properties of intuitionistic, classical, and other logics. From a computer-science perspective, linear logic provides a logical basis for the study of many thorny issues such as static update-in-place analysis of functional programming languages [144, 17]. Other current and possible future uses of linear logic include logic programming issues, artificial intelligence, models of concurrency, parallel computation, and the study of metamathematics.

Many of the results presented here were originally presented in conferences or journals, and most appeared in the author's Stanford University PhD. dissertation [109]. The main results of this book represent joint work with other authors. In particular, chapters 3, 4, and section 5.1 of chapter 5 stem from work with John Mitchell, Andre Scedrov, and Natarajan Shankar, who all contributed substantively to the content [107]. Section 5.2.2 represents joint work with Natarajan Shankar, and follows independent work of Max Kanovich. Section 5.2.3 is joint work with Tim Winkler [106]. Section 6.2 is joint work with Andre Scedrov and Natarajan Shankar [104]. Section 6.3.1 summarizes joint work with Natarajan Shankar [108], and section 6.3.2 is an overview of work with Andre Scedrov [103]. Section 6.4 presents work jointly accomplished with Andre Scedrov and Natarajan Shankar [105]. Chapter 7 is a summary of natural deduction drawing heavily from Prawitz seminal work [131]. Section 7.7 reviews work by many authors. Chapter 8 is joint work with John Mitchell [102].

Patrick Lincoln, Menlo Park, CA, 1995

Computational Aspects of Linear Logic

1 Introduction

Neither a borrower nor a lender be;
For a loan oft loses both itself and a friend,
And borrowing dulls the edge of husbandry.
This above all: to thine own self be true,
And it must follow, as the night the day,
Thou canst not then be false to any man.
WILLIAM SHAKESPEAR 1564-1616

Hamlet 1601

Linear logic was introduced by Girard in 1987 [57] as a "logic behind logics," and as a "resource-conscious logic." Linear logic can be seen as step along the path started by Hilbert [72] and Gentzen [56] toward formalizing proofs as objects that can be treated algorithmically. The main theoretical concern here is to strengthen the conceptual underpinnings necessary to apply proof theory to reason about computations. We demonstrate the undecidability of propositional linear logic, prove that noncommutative non-additive propositional linear logic is also undecidable, and give tight complexity results for other natural fragments of linear logic. Then, using these results, we explore an application of proof theory to computation, describing functional languages based on linear logic. The linear analysis of this programming language yields compile-time type information about resource manipulation, which may be useful in the control of some aspects of program execution such as storage allocation, garbage collection, and array update in place.

This book investigates two computational interpretations of linear logic. The first set of results demonstrates the power of a correspondence advocated by Girard between proofs and computations. This correspondence links formulas to states of machines, and connects inference steps in a proof to transitions in a computation. This "Girard" correspondence allows the use of proof theory to reason about computations and their properties such as correctness and termination. Moreover, it is the natural way to study the computational complexity of decision problems in various logics. The second set of results revisits the Curry-Howard correspondence between proofs and programs, originally observed for intuitionistic logic. Linear logic adds a greater degree of control over the structure of programs to the Curry-Howard correspondence. The main conclusion drawn from these results is that linear logic is a *computational* logic behind logics. That is, linear logic is not about "Truth"; it is about computation.

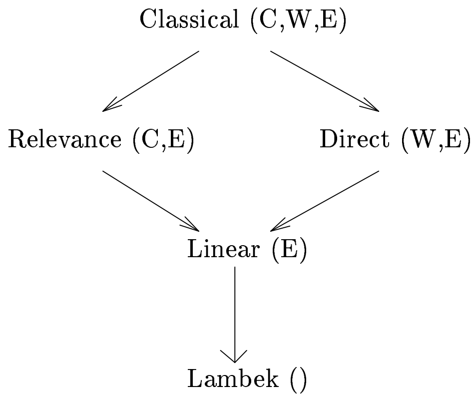


Figure 1.1
Inclusions of structural rules.

1.1 Brief History of Linear Logic

Linear logic, although only recently formulated [57], is closely related to much older logics. Lambek developed the Lambek calculus, a noncommutative logic intended for the analysis of natural language sentence structure a few decades earlier than Girard formulated linear logic [99, 100]. Relevance logic and Direct logic, which are also much older than linear logic, had already been well studied when linear logic appeared on the scene [46, 146, 87]. In an oversimplified view, linear logic sits below relevance and direct logic, and above the Lambek calculus, according to the inclusion or exclusion of certain "structural rules" called weakening (W), contraction (C), and exchange (E), discussed in Section 1.4.3 [30, 124], and summarized in Figure 1.1. Direct logic is closely related to Affine logic, which is linear logic with the addition of the unrestricted exchange rule.

Linear logic arose from the semantic study of intuitionistic implication. Girard gave two separate semantics in his article introducing linear logic [57], and the exploration of alternate semantic bases for linear logic continues [130, 114, 134, 45, 14, 35, 98, 4, 125]. Linear logic proof theory was also begun by Girard with the introduction of a sequent calculus for linear logic, and an alternate notation for linear logic proofs called proof nets [57]. Study of proof net related issues is also still active [59, 58, 60, 95, 4, 66, 43].

This book addresses several issues in the proof theory of linear logic, but is not primarily concerned with semantics. This is not to say that semantic issues are unimportant in the realm of decision problems. Some recent complexity results have been obtained by Lafont [96] and by Lafont and Scedrov [97] using phase-space semantics to demonstrate soundness.

1.2 Overview of Proof-Theoretic Results

This investigation into linear logic first builds a set of proof-theoretic tools for linear logic, including cut-elimination, non-logical theories, and permutabilities of inferences. These tools are not particularly novel but the presentation of them in the sequent calculus and collection of them together aids further developments. Although Gentzen developed natural deduction sytems for classical and intuitionistic logic, he utilized his sequent-caluclus presentations to obtain most of his proof-theoretic results (see Chapter 7). Similarly, the focus here is on sequent-calculus presentations. In particular, Girard proved cut-elimination using proof nets [57] (see Section 7.7), while cut-elimination is proved here in Section 2.6 using the sequent calculus in order to facilitate later proofs which make use of the syntax and exact form of this proof. Non-logical theories for linear logic have been investigated in [67, 115] and are extended here in Section 2.10 to include a wider range of axioms. In the broad context of using linear logic as the basis for a linear logic programming language several authors have studied permutability and impermutability properties [7, 27, 51, 108], which are summarized in Section 2.9. Girard's proofs of soundness and completeness of proof nets for fragments of linear logic also (implicitly) give permutability theorems in the sequent calculus [57]. A complete list of permutabilities is given in Section 2.9.

Two sets of results are built on this expanded proof theoretic foundation corresponding to two modes of interpreting logics computationally. One branch of development investigates the logic programming approach which interprets logical formulas as programs and the process of proof search is interpreted as computation. A subcase of this approach which connects sequents with states, and proofs with computations is utilized frequently here. The second branch of development explores a Curry-Howard-style correspondence between proofs and programs, and proof-normalization with execution. These two modes are summarized briefly in the next subsections.

1.2.1 Logic Programming

The "logic programming" approach to the computational interpretation of a logic, as exemplified by the programming language Prolog, is a correspondence between conclusions of formal proofs and programs, and also between the process of searching for a proof and the execution of the logic program. Cerrito has analyzed Prolog using linear logic [37]. Andreoli and Pareschi, among others, have investigated this paradigm as it is applied to linear logic, and have developed a very useful programming style and implementation [8, 9, 73, 69, 70, 85, 91].

A subcase of the logic programming approach ties together a specific model of computation with the *result* of a sucessful proof search. Thus a connection between formulas and states of computation, and between proofs and computations is estabilished. This correspondence differs from the usual logic programming approach in that it identifies *proofs* with computations, whereas logic programming identifies *proof search* with computation. This correspondence is refined through separating the program from the state of the machine by working in nonlogical theories that are derived from programs. This technique is developed here to maintain a clear distinction between program and state: a program is a non-logical theory, a machine state is a sequent, and a normal proof is a (successful) computation, perhaps best exemplified here in Chapter 3.

This correspondence is used to prove several new complexity results for the decision problems for several fragments of linear logic. Natural fragments of linear logic vary widely in complexity. One fragment has an NP-complete decision problem, one has a PSPACE-complete decision problem, and full propositional logic is undecidable. Using the Girard correspondence, many models of computation can be captured logically. For example, in noncommutative linear logic one can encode several variants of Turing machines (multi-tape, multi-head, parallel). A second example, due to previous researchers, is that Petri net computations can be captured quite naturally in the tensor fragment of linear logic with tensor theories [12, 115]. Also, many logical formalisms used in knowledge representation and linguistics are characterized by fragments of linear logic.

1.2.2 Linearizing Curry-Howard

Another very useful correspondence, known as the "propositions-as-types" or Curry-Howard correspondence, connects natural deduction systems for intuitionistic logic and typed lambda calculus. This connection is quite deep, as cut elimination in intuitionistic logic corresponds to reduction in the lambda calculus. Thus one may view typed lambda terms as notations for proofs in intuitionistic logic. One may also view the soundness of cut-elimination for intuitionistic logic as providing a proof that reduction preserves types in the lambda calculus. Further, the strong normalization property of natural deduction systems for intuitionistic logic (*i.e.* the cut-elimination procedure terminates) immediately yields a strong normalization property for simply-typed lambda terms.

Here we analyze an analogous correspondence between a functional language and linear logic which has been studied by Girard, Lafont, Abramsky, and others [94, 2, 150, 151, 155, 41, 31]. Interesting related work along a similar vein (the geometry of interaction) includes [59, 58, 60, 95, 4, 66, 43]. The main novel points developed here are the proofs of the subject-reduction and most general type theorems for a linear functional language. Also, from an implementation standpoint, a two-space memory model is developed which allows a compiler to generate more efficient code by taking into account the linear type of certain terms. This memory model has been used in an implementation of a linear declarative language based on the Three Instruction Machine (TIM) [48].

1.3 Informal Overview of Linear Logic

This section presents a brief intuitive preview of linear logic. It is meant to help readers unfamiliar with the technicalities of proof theory understand the spirit, if not the detail, of the later results. This section can be skipped by those already passingly familiar with linear logic.

"Linear logic is a resource sensitive logic" — *Girard*

This and other slogans can lend insight into the proof theory and actual mechanics of linear logic, if the correct interpretation is found. This particular slogan can be interpreted as saying that linear logic propositions are like valuable resources that may not be arbitrarily duplicated or thrown away. Other logics such as classical logic treat propositions as static, permanent truths or falsehoods about the world.

"Linear logic is a logic behind logics" — *Girard*

This quote may be taken at first to mean that linear logic is more primitive and more basic than other logics. The majority of this book is dedicated to understanding this statement, and its implications, especially when the slogan is altered somewhat to "Linear logic is a *computational* logic behind logics."

Let us begin with a discussion of the differences between linear logic and classical, old fashioned logic.

Classical logic deals in static propositions about the world, while linear logic manipulates commodities at a lower, more detailed level. A linear logic proposition A might stand for a resource such as "a dollar" while a classical logic proposition P might stand for "the moon is made of green cheese." The classical, abstract view of propositions as being either true or false is extremely useful and long lived: it has been more than two thousand years since Aristotle described an early form of classical logic. The more dynamic, detailed view which forms the basis of linear logic is much newer, only appearing (in various disguises) in the last few decades [99, 100, 57].

Because of the static nature of propositions in classical logic, one may duplicate propositions: P *implies* (P and P). In this logical sentence the proposition P has been duplicated, and implicitly we learn that "one P is as good as two." Also, one may discard propositions: (P *and* Q) *implies* P. Here the proposition Q has been thrown away in moving from one side of the implication to the other. Both of these sentences are valid in classical logic for any P and Q.

In linear logic, these sentences are not valid. A linear logician might ask "Where did the second P come from?" and "Where did the Q go?" Of course, there is no satisfactory answer, since the questions are nonsequiturs if propositions are assumed to be static, unchanging facts about the world. Static logical truths don't "come from" anywhere, and don't "go" anywhere either. However, one may assign a different meaning to propositions which involves dynamic properties of systems and resources. For example, consider the resource A, standing for "a dollar." The proposition A is not a permanent fact about the static world. Rather, linear propositions like A stand for some real and identifiable thing. The connective *and* spoken about above is written $\otimes$ in linear logic. Thus $A \otimes A$ is very different from A, in the same way that two dollars are very different from one dollar.

Thus in linear logic one must keep track of propositions as if they were valuable resources. The number of copies of a proposition is of importance in linear logic, just as the number of dollars in my pocket is usually more important than the fact that there exists some dollar in my pocket. Classical logic (intentionally) confuses the issue, muddying the distinction between "I have a dollar" and "there is a dollar." This difference is sometimes truly unimportant, but in linear logic we study the case when the difference is very important.

Environmentally conscious logics such as linear logic which keep track of the number of copies of formulas are often more expressive than classical logic. This expressive power is evidenced by some very natural encodings of computational models such as Turing machines, Minsky, or counter machines, Petri nets, circuits, and others. The logical embedding of these computational models into linear logic first provides immediate complexity results for the decision problem for linear logic, as well as providing an alternate, logical basis for the study of these computational structures.

In linear logic, one may express the connection between a depletable proposition A and its "permanent" (resource-unconscious) version by means of a modal "storage" operator !. The formula $!A$ has the intuitive meaning that A is available as much or as little as you like. The U.S. Government (Federal Reserve) has ! dollars, since it can print money as the need arises. A magician's hat can appear to contain !silk scarves. However, most everyday objects are annoyingly linear: resources like dollars, gasoline, and laundry soap unfortunately can be used up. Using the modal operator !, one may encode classical and intuitionistic logic into linear logic (see Section 6.1). Such encodings show that linear logic is at least as expressive as classical logic, but in fact linear logic is much more expressive, as we shall see later.

1.3.1 Chemistry Example

As an example of the way in which resource problems can be encoded in linear logic, consider the following chemical reaction.

$2H_2 + O_2 \rightarrow 2H_2O$

This chemical reaction, being well-balanced, can be directly represented by a provable linear logic sequent using the multiplicative conjunction *tensor* $\otimes$ and implication *linear implication* (or lollipop) $\multimap$.

$(H \otimes H)^2 \otimes (O \otimes O) \multimap (H \otimes H \otimes O)^2$

Note that in linear logic, as in chemistry, the number of occurrences of a formula has great importance. In classical or intuitionistic logic the distinction between one and two copies of a (molecular) formula is lost. This is a typical case where the fallacy of classical reasoning about the real world is evident. Althoug linear logic fails to capture all aspects of the chemical situation (e.g. in this encoding the energy released by the reaction is ignored), replacing linear connectives by classical ones would fail to distinguish between this equation and one with only a single H and a single O on the right hand side. A linear logician can rightly ask of such an equation "Where did the other propositional atoms go?"

1.3.2 Menu Example

Another intuitive example of the meaning of linear logic formulas is given by the following restraunt menu example [61]. Consider the complete meal offered at many breakfast joints. For three dollars you get waffles or pancakes, two eggs, fruit (strawberries or blueberries depending on availability), and all the coffee you can drink. In linear logic this offer might be encoded as

$$\begin{gathered} \$ \otimes \$ \otimes \$ \\ \multimap \\ [(\mathrm{Waffles} \& \mathrm{Pancakes}) \otimes (\mathrm{Egg} \otimes \mathrm{Egg}) \otimes (\mathrm{Strawberries} \oplus \mathrm{Blueberries}) \otimes !\mathrm{Coffee}] \end{gathered}$$

The intuitive meaning of these connectives can then be read as follows. $A \otimes B$ means that one has A and B together (i.e. in your possesion at the same time). $A \multimap B$ means that one can exchange A for B. Waffles&Pancakes means that you have your choice between Waffles and Pancakes, but you cannot obtain both as part of one instance of this deal. Strawberries $\oplus$ Blueberries means that you get either Strawberries or Blueberries, but not both, and the choice is not yours: you must be prepared to accept either. !Coffee means that you have a bottomless cup of coffeee: you can ask for and receive more cups of coffee at will, and you can stop asking for more at any point. Again, classical or intuitionistic reasoning would fail to make the linear distinction between $P \otimes P$ and P. This can be an important distinction when P represents something that *counts*, like $ in the above example.

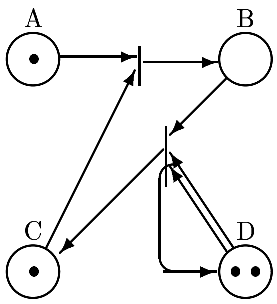
1.3.3 Petri Nets

An interesting example of the expressiveness of linear logic is the following natural and simple encoding of Petri net reachability problems. This example illustrates one way in which linear logic is a logic of concurrency, and concretely demonstrates the usefulness of the propositions-as-resources approach.

A Petri net [127] consists of a set of *places* and a disjoint set of *transitions*, together with a relation between them. A Petri net *marking* is a multiset over the set of places. The places are to be thought of as locations where some number of *tokens* reside. A marking describes how many tokens are located at each place. The relation describes the effect a transition has on a marking: it describes the multiset of tokens that a transition uses (its preset or enabling set) and the multiset of tokens that a transition produces. A transition is *enabled* (can be fired) only if its preset is a submultiset of the current marking. The Petri net reachability problem is defined as: is a given final marking reachable from a given initial marking using some sequence of transitions from a given Petri net? There is also a purely algebraic formulation of Petri nets called vector replacement systems [86].

The reachability problem is difficult because in many instances a transition is not enabled, and there is no known bound on the number of tokens that may occupy a place. Petri-net reachability problem is known to be EXPSPACE-hard, but decidable [110, 118, 117, 116, 93]. An exact upper bound on the complexity of this problem has not yet been provided.

Places are usually indicated graphically with circles and transitions as lines, with the relation being represented by arrows leading from places to transitions and from transitions to places. An example Petri net with two transitions and four places is given below.



The encoding of Petri nets into linear logic is fairly direct. One can encode Petri net transitions as reusable linear implications, and tokens can be represented as atomic propositions. In linear logic the formula $!((a \otimes c) \multimap b)$ may be used to encode a Petri net transition taking tokens from place a and c and adding a token to place b. Similarly, the formula $!((b \otimes d \otimes d) \multimap (c \otimes d))$ may be seen as a transition taking one token from b and two tokens from d, and adding one token to c. A reachability problem for this Petri net may be presented as a sequent such as the following.

$$!((a \otimes c) \multimap b), !((b \otimes d \otimes d) \multimap (c \otimes d)), a, c, d, d \vdash c \otimes d$$

This sequent is provable in linear logic if and only if there is a sequence of Petri net transition firings that transform the marking $\{acdd\}$ to the marking $\{cd\}$. (one can view the comma on the left of a sequent as synonymous with $\otimes$) It happens that this reachability problem is solvable by firing the first rule and then the second rule. The linear logic sequent is provable by analogous logical steps.

The connection between linear logic and Petri nets has been well-studied [12, 67, 115, 13, 53], and extended to cover other models of concurrency [95, 5, 130].

Since Petri net reachability is computationally difficult, this example shows that deciding propositional linear logic formulas is difficult. However, since Petri net reachability is decidable [118, 117, 93], this does not provide an argument that linear logic must be undecidable. However, linear logic contains connectives beyond those used in this encoding of Petri nets (the additives & and $\oplus$) that we shall see later provide enough additional machinery to encode the undecidable halting problem of counter machines.

1.4 Formal Overview of Linear Logic

Below is a brief presentation of the notation used here, and some of the more foundational concepts of logic and proof theory necessary for the remainder of this thesis. This book is self-contained, but interested readers are referred to [56, 89, 57, 64] for some of the proofs referred to in the text. Introductions to linear logic in general are given in [133, 101, 61, 141].

1.4.1 Notation and Terminology

Girard's notation for the logical connectives of linear logic ($\otimes \parr$, $\&$, $\oplus$, 1, $\perp$, $\top$, 0) will be used here throughout. The following notational conventions will be used.

p_i	Positive propositional literal
$p_i^\perp$	Negative propositional literal
A, B, C	Arbitrary formulas
$\Sigma, \Gamma, \Delta, \Theta$	Arbitrary sequences of formulas

Here are the connectives and constants of linear logic with some possible pronunciations and informal meanings.

$\otimes$	"Times," "Tensor," multiplicative conjunction, both at once
1	"One," unit of tensor, true in isolation
$\parr$	"Par," multiplicative disjunction, if not one, then the other
$\perp$	"Bottom," unit of par, false in isolation
$\&$	"With," additive conjunction, classical *and*
$\top$	"Top," unit of with, really true
$\oplus$	"Plus," additive disjunction, classical *or*
0	"Zero," unit of plus, really false
!	"Bang," positive modal, reusable
?	"Why Not," negative modal, redisposable
$\multimap$	"Linear Implication," "Lollipop," $A \multimap B \stackrel{\Delta}{=} A^\perp \parr B$, use-once conversion

There are several alternative notations for the linear logic connectives, perhaps the most popular being those motivated by relevance logic and category theory. This notation, for example found in Troelstra [142] uses $\star$ for $\otimes$, 0 for $\perp$, $\sqcap$ for $\&$, $+$ for $\oplus$, and $\perp$ for 0. Some other authors use I for 1, $\mathbf{t}$ for $\top$, and $\mathbf{f}$ for 0. Note also that many authors use the capital greek letters Σ, Γ, Δ to stand for multisets of formulas rather than sequences. This distinction is minor except when considering noncommutative logics.

1.4.2 Sequent Calculus

The sequent calculus, devised by Gentzen [56], will be used throughout this book.

A *sequent* is composed of two sequences of formulas separated by the ("Girardian") turnstile symbol, $\vdash$. One may read the sequent $\Delta \vdash \Gamma$ as asserting that the (multiplicative) conjunction of the formulas in Δ imply the (multiplicative) disjunction of the formulas in Γ.

A *sequent calculus proof rule* consists of a set of hypothesis or premise sequents, displayed above a horizontal line, and a single conclusion sequent, displayed below the line, as below:

$$\frac{\text{Hypothesis1} \qquad \cdots \qquad \text{HypothesisK}}{\text{Conclusion}}$$

For example, this is the standard rule for conjunction:

$$\frac{\Sigma \vdash A, \Delta \qquad \Sigma \vdash B, \Delta}{\Sigma \vdash A \wedge B, \Delta}$$

The two hypotheses are $\Sigma \vdash A, \Delta$ and $\Sigma \vdash B, \Delta$, and the single conclusion is $\Sigma \vdash A \wedge B, \Delta$. As is standard, rules are implicitly universally quantified schemas. Some sequent calculus rules have one hypothesis, and some have no hypotheses. For example, the rule below for identity (logical axiom) has no hypotheses:

$$\overline{A \vdash A}$$

Given a set of proof rules, a *deduction* is a structure where the conclusion of one proof rule exactly matches the hypothesis of the next rule. Because each rule has exactly one conclusion, all deductions are trees, with the root (conclusion) at the bottom, and the leaves (hypotheses) at the top. Each branch of a deduction is a sequence of applications of proof rules, some, such as $\wedge$, represent branching points in the deduction tree, and some, such as identity, terminate a branch. A *proof* in the sequent calculus is a deduction with no assumptions, *i.e.*, every leaf is either identity or a logical axiom. In other words, each branch terminates with an application of a proof rule with no hypotheses.

A useful extension to the sequent calculus is the introduction of theories. A *theory* is a set of sequents, each element of which is called a (non-logical) axiom. A proof in a theory is a deduction where every branch is terminated by a proof rule with no hypotheses, a logical axiom, or by a sequent that is in the theory.

Sequent proofs of formulas will be displayed with the name of the inference rule near the right hand side of the line delineating the inference. This is an aid to the reader of complicated sequent proofs, although this is not a standard proof notation. For example, here is a sequent calculus proof with two applications of the identity rule, and one application of the conjunction rule:

$$\dfrac{\dfrac{}{A \vdash A}I \qquad \dfrac{}{A \vdash A}I}{A \vdash A \wedge A}\wedge$$

The next is a proof in the theory with the single axiom $A \vdash B$. Note that we have written a line over the use of the axiom as if there were an inference rule for each axiom in a theory, and labeled it T.

$$\dfrac{\dfrac{}{A \vdash B}T \qquad \dfrac{}{A \vdash A}I}{A \vdash B \wedge A}\wedge$$

The *principal formula* of an occurrence of an inference rule is the formula that appears in the conclusion but does not appear in any hypothesis. In the case above, the formula $B \wedge A$ is the principal formula of the $\wedge$ rule. We will also say that the formula $B \wedge A$ is *analyzed* or *introduced* in such a case. This terminology stems from usage in proof search, where one begins with a conclusion, or root sequent, and searches for proofs. In incrementally building that deduction, the proof search procedure is said to analyze a formula.

The occurrence of a rule in a proof is said to be an *inference*. When we speak of a formula in a sequent, we are really referring to an occurrence of the formula.

1.4.3 Structural Rules

The discussion in Section 1.4.2 applies to the sequent calculi for classical or intuitionistic logic as Gentzen originally presented them [56], but the main focus of this book is linear logic.

The rules of inference for linear logic differ from classical logic and intuitionistic logic in many ways, but the most dramatic difference can be explained by the rules of classical logic that are "missing" from linear logic: weakening and contraction.

Classical and intuitionistic logics allow the following rule of inference called *weakening* or *thinning*:

$$\dfrac{\Sigma \vdash \Delta}{\Sigma, A \vdash \Delta}$$

Intuitively, if Σ is the single formula X, and Δ is the single formula Y, and one has a proof of X implies Y, then one can assert that (X and A) implies Y. The effect of this rule is to weaken the conclusion, but if the original sequent was true, then the new one surely is as well.

Classical logic allows weakening to be applied on both sides of a sequent (i.e., both sides of $\vdash$). This is sound because the right hand side of a sequent is interpreted as the disjunction of formulas, and thus adding formulas to the conclusion also weakens the result. Intuitionistic logic restricts weakening to apply only on the left hand side of the $\vdash$, since intuitionistic logic forbids multiple formulas from appearing on the right hand side of a sequent. Linear logic completely forbids the application of weakening on either side of $\vdash$.

Classical and intuitionistic logics also allow the following rule of inference called *contraction*:

$$\frac{\Sigma, A, A \vdash \Delta}{\Sigma, A \vdash \Delta}$$

Intuitively, if one can prove a formula from two assumptions of a formula A, then one assumption of formula A suffices. This is a mild strengthening of the conclusion. Algebraically, the rule asserts that the comma (conjunction) forming the left of a sequent is idempotent.

Classical logic allows contraction to be applied on both sides of a sequent (i.e., both sides of $\vdash$). Intuitionistic logic restricts contraction to apply only on the left hand side of the $\vdash$. Linear logic completely forbids the application of contraction on either side of $\vdash$.

Classical, intuitionistic, and linear logic all allow the following structural rule called *exchange* (or permutation):

$$\frac{\Sigma, A, B, \Gamma \vdash \Delta}{\Sigma, B, A, \Gamma \vdash \Delta}$$

Intuitively, the rule asserts that the order of formulas is unimportant, or in other words, the comma (conjunction) is commutative. In Chapter 4 we consider the noncommutative variant of linear logic, with the structural rule of exchange omitted from the logic.

As a convenience, one may simplify presentations of the sequent calculus for classical or intuitionistic logic by treating sequences of formulas Σ as *sets*. A derivation in this system would no longer require that the conclusion of each rule exactly match the hypothesis of the next rule, but instead one simply requires a match up to applications of the structural rules of weakening, contraction, and exchange. One may view this as simply ignoring the applications of structural rules, leaving their application implicit. The analogous presentation of linear logic treats sequences of formulas as multisets. Here we will be careless with regard to the order of formulas in sequents except when we deal with noncommutative logics.

1.4.4 Conjunction and Disjunction

In sequent calculus with the structural rules of weakening, contraction, and exchange on both sides of $\vdash$, the following two rules for conjunction are equivalent. That is, the rule for $\wedge_1$ may be derived from the rule for $\wedge_2$, and vice versa.

$$\frac{\Sigma \vdash A, \Delta \qquad \Sigma \vdash B, \Delta}{\Sigma \vdash (A \wedge_1 B), \Delta} \qquad\qquad \frac{\Sigma \vdash A, \Delta \qquad \Theta \vdash B, \Gamma}{\Sigma, \Theta \vdash (A \wedge_2 B), \Delta, \Gamma}$$

However, if one removes the structural rules of contraction and weakening, as in linear logic, the rules for $\wedge_1$ and $\wedge_2$ are *not* equivalent. The form of rules where the context (Σ and Δ) is used in both hypotheses ($\wedge_1$) is hereafter referred to as an "additive" rule, and the additive conjunction will be written $\&$. The second kind of rule, where the context is divided among the hypotheses, is referred to as "multiplicative," and will be written $\otimes$. The precise proof rules for these connectives are repeated in Figure 1.2 and Appendix A.1.

For similar reasons, linear logic distinguishes between two kinds of disjunctions, an additive one $\oplus$ (plus) and a multiplicative one $⅋$ (par):

$$\frac{\Sigma \vdash A, \Delta}{\Sigma \vdash (A \oplus B), \Delta} \qquad\qquad \frac{\Sigma \vdash A, B, \Delta}{\Sigma \vdash (A ⅋ B), \Delta}$$

1.4.5 The Cut Rule

The cut rule can be read to imply that a formula A on the right is stronger than the same formula A on the left, symmetric with the reading of the identity rule as saying the opposite.

$$\frac{\Gamma_1 \vdash A, \Sigma_1 \qquad \Gamma_2, A \vdash \Sigma_2}{\Gamma_1, \Gamma_2 \vdash \Sigma_1, \Sigma_2}\; Cut$$

This rule may be read as "if A or Σ_1 can be derived from Γ_1, and Γ_2 and A together derive Σ_2, then Γ_1 and Γ_2 together derive Σ_1 or Σ_2." Of course, we intend multiplicative conjunction and disjunction in the above description. This rule perhaps is easiest to understand if Σ_1 is empty. In that case, this rule essentially states that Γ_1 can be plugged in for A in the derivation of $\Gamma_2, A \vdash \Sigma_2$.

1.4.6 The Units of Linear Logic

In linear logic, the four connectives have separate units (identity elements). The unit of $\otimes$ is called 1 (one), the unit of $\&$ is $\top$ (top), the unit of $⅋$ is $\bot$ (bottom), and the unit of $\oplus$ is 0 (zero). The sequent rules for these constants are given below. Note that there is no rule for 0 on the right hand side of the turnstile, nor for $\top$ on the left.

$$\frac{}{\Sigma \vdash \top, \Delta} \qquad \frac{}{\vdash 1} \qquad \frac{}{\bot \vdash}$$

$$\frac{}{\Sigma, 0 \vdash \Delta} \qquad \frac{\Sigma \vdash \Delta}{\Sigma, 1 \vdash \Delta} \qquad \frac{\Sigma \vdash \Delta}{\Sigma \vdash \bot, \Delta}$$

The $\top$ rule may be thought of as stating that $\top$ is "really true," no matter what the context. On the other hand, 1 is true, but only in the complete absence of any other formulas. $\bot$ is a kind of false that may be disregarded, and 0 is a kind of false that must be accounted for.

1.4.7 Resource Control Aspects of Linear Logic

An important property of linear logic with $\otimes$ $⅋$, $\&$, $\oplus$, 1, $\bot$, $\top$, 0 is that each connective of the conclusion of a proof is analyzed at most once in any branch of the proof. This can be seen as an advantage, since it affords a great deal of control over the process of proof search, and yields an immediate polynomial space decision procedure. This can also be seen as a disadvantage, because one cannot symmetrically encode intuitionistic logic since intuitionistic logic allows arbitrary uses of certain formulas. Therefore a pair of modal operators ! and ? are included in linear logic to retrieve the kind of expressive power introduced by arbitrary reuse.

The formula $!A$ represents unlimited use of the formula A. Thus in the sequent $!A, \Sigma \vdash \Delta$, the connectives of A may be analyzed any number of times, including zero. In logical terms, weakening and contraction apply to ! formulas on the left of the $\vdash$.

Here are the precise sequent rules for linear !:

$$\textbf{! W} \quad \frac{\Gamma \vdash \Sigma}{\Gamma, !A \vdash \Sigma} \qquad \frac{\Gamma, !A, !A \vdash \Sigma}{\Gamma, !A \vdash \Sigma} \quad \textbf{! C}$$

$$\textbf{! D} \quad \frac{\Gamma, A \vdash \Sigma}{\Gamma, !A \vdash \Sigma} \qquad \frac{!\Gamma \vdash A, ?\Sigma}{!\Gamma \vdash !A, ?\Sigma} \quad \textbf{! S}$$

The **!W** rule is the weakening rule, here restricted to formulas on the left of the turnstile that are prefixed with !. The **!C** rule is the contraction rule, similarly restricted. The **!D** rule is the dereliction rule, connecting the modal formulas to the rest of the logic. That is, one may consider ! to be a "wrapper," that allows arbitrary duplication and discarding, but which must be explicitly removed before the contents can be used. Finally, the **!S** promotion or storage rule allows the generation of ! formulas on the right hand side of a sequent. Intuitively, if one can prove A using resources which are all !'d, then one can produce any number of As (that is, $!A$). One may view ! as a particular kind of necessitation operator, strengthening the force of an assertion, since **!D** and **!S** are the standard proof rules for necessitation. There is also a set of rules for the ? operator, essentially the same as the rules for ! mirrored onto the opposite side of the $\vdash$.

1.4.8 Linear Logic Quantifier Rules

First-order predicate linear logic has the standard quantifier rules:

$$\exists\ \mathbf{R}\quad \frac{\Gamma \vdash A\{t/x\}, \Sigma}{\Gamma \vdash \exists x.A, \Sigma} \qquad \frac{\Gamma \vdash A\{y/x\}, \Sigma}{\Gamma \vdash \forall x.A, \Sigma}\quad \forall\ \mathbf{R}$$

In the $\exists$ **R** rule, substituting an arbitrary term t for the free occurrences of x in A is denoted $A\{t/x\}$, where the bound variables in A are renamed to avoid any clashes. The $\forall$ **R** rule is only applicable if the variable y is not free in Γ, Σ, and any nonlogical theory axioms. As for the other connectives of linear logic, there are right and left versions of the $\exists$ and $\forall$ rules. Provable first-order formulas of linear logic include $\forall x.P(x) \multimap \exists y.P(y)$ and $(\forall x.P(x)) \otimes (\forall y.Q(y)) \multimap (\forall z.P(z) \otimes Q(z))$.

Analogous rules for variables ranging over propositions can be defined with the same restrictions on variable names and instances, resuling in higher-order linear logic.

$$\exists 2\ \mathbf{R}\quad \frac{\Gamma \vdash A\{t/x\}, \Sigma}{\Gamma \vdash \exists x.A, \Sigma} \qquad \frac{\Gamma \vdash A\{y/x\}, \Sigma}{\Gamma \vdash \forall x.A, \Sigma}\quad \forall 2\ \mathbf{R}$$

With these rules one may consider combinations of first-order and higher-order systems of linear logic. Example formulas of second-order linear logic include $\forall X.X$ which behaves very much like the additive constant 0, and $\exists Y.Y$ which is related to the additive constant $\top$. The first of these is not provable in linear logic, and the second is, by instantiation of $A \multimap A$ for Y.

Some properties of propositional linear logic such as the subformula property fail to directly extend to first order or higher order systems, but the most important properties of cut-elimination and consistency continue to hold. We will focus primarily on propositional linear logic, where quantifiers never appear, and discuss the quantified cases in Chapter 6.

1.4.9 The Sequent Calculus of Linear Logic

The sequent rules for linear logic assume that the commas of the sequent are multiplicative. That is, a sequent $\Sigma \vdash \Gamma$ asserts that the multiplicative conjunction of the formulas in Σ together imply the multiplicative disjunction of the formulas in Δ.

The rules given in Figure 1.2 originally appeared in [57]. However, the rules presented here are for the two-sided sequent system, with formulas appearing on both sides of the $\vdash$. It has become somewhat standard now to present linear logic in a one-sided sequent system, by negating every formula which would have appeared on the left of the $\vdash$, and moving them to the right. One-sided systems have the advantage of having half the proof rules as two-sided systems for the same logic, but suffer from the disadvantage that sequents are harder for some to read.

The two-sided Gentzen-style inference rules for propositional linear logic are given in Figure 1.2 and are reproduced in Appendix A.1 for convenience.

It simplifies the presentation to consider negation as defined, rather than being a connective. The symbol $\triangleq$ is used below to denote "is defined as."

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
(P_i)^\perp & \triangleq & P_i^\perp & (P_i^\perp)^\perp & \triangleq & P_i \\
(A \otimes B)^\perp & \triangleq & A \multimap B^\perp & (A \multimap B)^\perp & \triangleq & A \otimes B^\perp \\
(A \otimes B)^\perp & \triangleq & A^\perp \parr B^\perp & (A \parr B)^\perp & \triangleq & A^\perp \otimes B^\perp \\
(A \oplus B)^\perp & \triangleq & A^\perp \,\&\, B^\perp & (A \,\&\, B)^\perp & \triangleq & A^\perp \oplus B^\perp \\
(!A)^\perp & \triangleq & ?A^\perp & (?A)^\perp & \triangleq & !A^\perp \\
(1)^\perp & \triangleq & \perp & (\perp)^\perp & \triangleq & 1 \\
(0)^\perp & \triangleq & \top & (\top)^\perp & \triangleq & 0 \\
(\forall x.A)^\perp & \triangleq & \exists x.(A)^\perp & (\exists x.A)^\perp & \triangleq & \forall x.(A)^\perp
\end{array}
$$

We also define the linear implication connective $\multimap$ with the equation

$$A \multimap B \triangleq A^\perp \parr B$$

and thus we omit explicit rules for the $\multimap$ connective, although many find the $\multimap$ connective much easier to comprehend than $\parr$.

$$
\begin{array}{rccl}
\mathbf{I} & \dfrac{}{A \vdash A} & \dfrac{\Gamma_1 \vdash A, \Sigma_1 \quad \Gamma_2, A \vdash \Sigma_2}{\Gamma_1, \Gamma_2 \vdash \Sigma_1, \Sigma_2} & \mathbf{Cut} \\[2ex]
\mathbf{E\ L} & \dfrac{\Gamma_1, A, B, \Gamma_2 \vdash \Sigma}{\Gamma_1, B, A, \Gamma_2 \vdash \Sigma} & \dfrac{\Gamma \vdash \Sigma_1, A, B, \Sigma_2}{\Gamma \vdash \Sigma_1, B, A, \Sigma_2} & \mathbf{E\ R} \\[2ex]
\otimes\ \mathbf{L} & \dfrac{\Gamma, A, B \vdash \Sigma}{\Gamma, (A \otimes B) \vdash \Sigma} & \dfrac{\Gamma_1 \vdash A, \Sigma_1 \quad \Gamma_2 \vdash B, \Sigma_2}{\Gamma_1, \Gamma_2 \vdash (A \otimes B), \Sigma_1, \Sigma_2} & \otimes\ \mathbf{R} \\[2ex]
\parr\ \mathbf{L} & \dfrac{\Gamma_1, A \vdash \Sigma_1 \quad \Gamma_2, B \vdash \Sigma_2}{\Gamma_1, \Gamma_2, (A \parr B) \vdash \Sigma_1, \Sigma_2} & \dfrac{\Gamma \vdash A, B, \Sigma}{\Gamma \vdash (A \parr B), \Sigma} & \parr\ \mathbf{R} \\[2ex]
\oplus\ \mathbf{L} & \dfrac{\Gamma, A \vdash \Sigma \quad \Gamma, B \vdash \Sigma}{\Gamma, (A \oplus B) \vdash \Sigma} & \dfrac{\Gamma \vdash A, \Sigma \quad \Gamma \vdash B, \Sigma}{\Gamma \vdash (A \& B), \Sigma} & \&\ \mathbf{R} \\[2ex]
\&\ \mathbf{L1} & \dfrac{\Gamma, A \vdash \Sigma}{\Gamma, (A \& B) \vdash \Sigma} & \dfrac{\Gamma \vdash A, \Sigma}{\Gamma \vdash (A \oplus B), \Sigma} & \oplus\ \mathbf{R1} \\[2ex]
\&\ \mathbf{L2} & \dfrac{\Gamma, B \vdash \Sigma}{\Gamma, (A \& B) \vdash \Sigma} & \dfrac{\Gamma \vdash B, \Sigma}{\Gamma \vdash (A \oplus B), \Sigma} & \oplus\ \mathbf{R2} \\[2ex]
\mathbf{!\ W} & \dfrac{\Gamma \vdash \Sigma}{\Gamma, !A \vdash \Sigma} & \dfrac{\Gamma, !A, !A \vdash \Sigma}{\Gamma, !A \vdash \Sigma} & \mathbf{!\ C} \\[2ex]
\mathbf{!\ D} & \dfrac{\Gamma, A \vdash \Sigma}{\Gamma, !A \vdash \Sigma} & \dfrac{!\Gamma \vdash A, ?\Sigma}{!\Gamma \vdash !A, ?\Sigma} & \mathbf{!\ S} \\[2ex]
\mathbf{?\ W} & \dfrac{\Gamma \vdash \Sigma}{\Gamma \vdash ?A, \Sigma} & \dfrac{\Gamma \vdash ?A, ?A, \Sigma}{\Gamma \vdash ?A, \Sigma} & \mathbf{?\ C} \\[2ex]
\mathbf{?\ D} & \dfrac{\Gamma \vdash A, \Sigma}{\Gamma \vdash ?A, \Sigma} & \dfrac{!\Gamma, A \vdash ?\Sigma}{!\Gamma, ?A \vdash ?\Sigma} & \mathbf{?\ S} \\[1ex]
0\ \mathbf{L} & \Gamma, 0 \vdash \Sigma & \Gamma \vdash \top, \Sigma & \top\ \mathbf{R} \\[2ex]
1\ \mathbf{L} & \dfrac{\Gamma \vdash \Sigma}{\Gamma, 1 \vdash \Sigma} & \dfrac{\Gamma \vdash \Sigma}{\Gamma \vdash \bot, \Sigma} & \bot\ \mathbf{R} \\[1ex]
\bot\ \mathbf{L} & \bot \vdash & \vdash 1 & 1\ \mathbf{R}
\end{array}
$$

Figure 1.2
Sequent Rules for Linear Logic (LL)

1.5 Overview of Results

This overview lists the most important results presented in this book and gives a glimpse of the proofs of the key theorems.

1.5.1 Basic Theorems

The most basic theorem of proof theory is the cut-elimination theorem. Most systems presented here enjoy a cut-elimination theorem, which simply states that any formula provable in a logic is also probable in that logic without the cut rule. Most proofs of cut elimination follow Gentzen's proof of the "Hauptsatz" [56]. The main idea is to first add a derived rule of inference that encapsulates the application of several inference rules. Then one eliminates all applications of cut and the derived rule together. The derived rules act as a bookkeeping trick that simplifies the proof of termination of cut elimination. This theorem was proved for linear logic by Girard in [57] using an elegant proof notation called proof-nets, briefly introduced in Section 7.7. In Chapter 2, a full proof of cut elimination in the sequent calculus is given in order to facilitate later proofs that depend on this form of proof of cut elimination. Some formal systems used in this book do not have a cut-elimination theorem, for example those systems with nonlogical theories. Nevertheless, in these cases, one may still find precise normal forms for proofs, even though all uses of cut cannot be eliminated.

Several results follow immediately from the cut-elimination theorem. For instance, the subformula property states that any formula appearing anywhere in a cut-free proof of a sequent also appears in the conclusion sequent. Also, the consistency of a logic follows from cut elimination and the subformula property. Linear logic has the subformula property and is consistent.

1.5.2 Complexity Results

The complexity of the decision problem for various fragments of linear logic are summarized here. Full linear logic with first-order quantifiers was known from the outset to be undecidable, as an immediate corollary of Girard's embedding of first-order intuitionistic logic into full first-order linear logic [57]. However, most of the results described in this book involve the propositional fragment of linear logic. The first result involves full propositional linear logic.

Undecidability Since most propositional logics are decidable, it is reasonable, a priori, to expect that propositional linear logic would be decidable also. Further evidence for decidability is the cut-elimination theorem for linear logic. However, we prove that propositional linear logic is undecidable. That is, the set of valid (provable) sequents in propositional linear logic is not recursive.

The most obvious line of proof is to encode counter machines in the manner of [116]. However, the obvious encoding runs into a difficulty. While it is easy to encode the increment and decrement instructions in linear logic using the multiplicatives, the critical zero-test instruction of counter machines has no natural analog in linear logic. That is, there is no connective that would allow a proof to proceed when there are no occurrences of a certain proposition (i.e., zero-test succeeds), but perhaps any number of other propositions. However, linear logic does have the expressive power to encode such a zero-test by other means.

The linear logic feature used to encode zero test is the additive conjunction. This connective allows one to encode a certain type of and-branching (in the terminology of alternation [39]). Viewing the proof computationally, two branches are fired off, each of which must be provable in the *same context*. This sharing of contexts provided by the additive conjunction enables one to encode zero-test by enforcing that one branch verifies that the counter is zero and then terminates, while the other branch continues assuming that the counter was in fact zero. If the counter in question is not zero, the first "zero-checking" branch will not terminate successfully, preventing successful termination of the entire computation. This kind of "zero-checking" can be accomplished natively in the logic.

Another approach to describing this result is that the reachability problem for alternating (in the sense of [40]) Petri nets is undecidable. One can directly encode normal Petri net transitions using the multiplicatives and exponentials [12, 115], and the alternation can be encoded using the additives. Previously, Urquhart has demonstrated the undecidability of relevant implication [146]. Relevant implication is similar to linear implication with unrestricted contraction: the relevant implication $A \rightarrow B$ might be represented in linear logic as $(A \otimes !A) \multimap B$. However, Urquhart's result does not bear on linear logic since it requires a distributivity axiom that is not present in linear logic [147].

NonCommutative Undecidability The noncommutative variant of propositional linear logic is undecidable, even *without* the additive connectives that are used in the commutative case to encode zero-test. The intuition behind this result is that one *can* test natively in noncommutative logic for zero occurrences of a certain proposition *in a certain position* without resorting to an and-branching (or alternating) encoding. The proof of undecidability is by reduction from the (undecidable) word problem for noncommutative semigroups.

Although the proof of undecidability of noncommutative propositional linear logic stated later is formulated in terms of Semi-Thue systems, it is perhaps easier to see this result informally in terms of standard Turing machines. One can encode a Turing machine tape as a sequence of propositions, one proposition for each tape symbol. The current position of the read/write head of the Turing machine, and its current state can be encoded as a single occurrence of a proposition at the appropriate location in the sequence of propositions corresponding to the Turing tape.

For example, $B, A, Q_i, B \vdash$ is a sequent which might encode a Turing machine with a tape with three non-blank symbols, currently in state Q_i, where the read/write head is currently over the symbol A. Transitions of the machine (the Turing machine program) can be encoded either as nonlogical theory axioms, or as modal implications. The formula $!((A \otimes Q_i) \multimap (Q_j \otimes B))$ may be read as: in state Q_i, if the head is over the symbol A, then erase the A, write a B, move the head to the right, and change to state Q_j.

It should be noted that in order for these results to hold, one must assume that either the modals of linear logic (for which contraction and weakening are applicable) commute with respect to any other formula, or that there is a new modal representing commutativity. In either case, there is an encoding of Turing machines, but without some limited form of exchange the above encoding fails. Since there are many reasonable assumptions one could make about the exact form of noncommutative linear logic, we assume commuting exponentials, as have previous authors, including Yetter [157].

Again in the noncommutative case, the first order version is conservative over the propositional fragment, and thus first order noncommutative linear logic is also undecidable.

pspace-Completeness of Propositional Linear Logic without Modals Without the modal operators ? and ! there is an immediate proof search procedure for linear logic that is complete and always terminates. Therefore, linear logic without the modals is decidable. As mentioned earlier, this logic has the property that every formula which occurs in the conclusion is analyzed at most once along any branch of any cut-free proof. This gives an immediate linear bound on the depth of any cut-free proof. Thus an obvious decision procedure takes only polynomial space, raising the question of whether the decision problem is PSPACE-hard. It happens that one may encode quantified boolean formulas in propositional nonmodal linear logic, and one can also encode (normal) intuitionistic implication in nonmodal linear logic. Both of these logics are known to be PSPACE-hard. Thus the decision problem for propositional linear logic without modals is PSPACE-hard, and therefore PSPACE-complete.

As we will see, the encoding of these PSPACE-hard problems is not straightforward. In the naive encoding of quantified boolean, discussed in detail later, formulas encoding $\forall X, \exists Y, F$ and $\exists Y, \forall X, F$ would be the same. This lack of attention to quantifier order can be corrected with the use of "lock and keys," a kind of multiplicative guard used to enforce an ordering on quantifiers in the encoding of quantified Boolean formulas, and also used to provide a kind of mutual exclusion in the encoding of intuitionistic implication.

np-Completeness of Multiplicative Fragment Without the modals and without additive operators, linear logic is reduced to its core fragment, the so-called multiplicatives. Multiplicative linear logic has the property that every formula that occurs in the conclusion is analyzed exactly once in the entirety of any cut-free proof. This gives an immediate linear bound on the number of inferences in any cut-free proof, showing that this logic is in NP, as one may simply guess and check an entire proof in linear space and time. Of course, this brings up the question of whether there is a polynomial decision algorithm for the multiplicatives, or if they are NP-hard. Previously, multiplicative affine (or direct) logic was shown to directbe NP-complete [107], but this proof relied on the use of weakening, a rule present in direct and affine logic, but not in linear logic. Max Kanovich settled this question by showing that the linear multiplicatives are NP-hard [79]. Chapter 5 presents an alternate proof of the NP-completeness. This proof is presented in order to facilitate the proof of the stronger result that even without propositions the multiplicatives are NP-complete. That is, constant-only multiplicative linear logic where formulas are built from only (multiplicative) and, or, true, and false $(\otimes, ⅋, 1, \perp)$ is NP-complete.

The key to the proof of NP-hardness of the multiplicatives is the existence of NP-complete problems that have the property that each entity in the problem must be used. In many problems such as SAT, if one proposition in a clause is true, the rest of the propositions do not matter, and may be either true or false. This "sloppy" behavior is very difficult to encode using only the multiplicatives. However, for certain NP-complete problems, such as 3-PARTITION, the combinatorial explosion arises from a partitioning of elements that must all be used in the end. Because of this property, 3-PARTITION can easily be encoded using just the multiplicative constants, thus demonstrating the NP-hardness of the linear (multiplicative) circuit evaluation problem.

Kanovich has extended the results regarding constant-only multiplicative linear logic to other fragments. Using a unique encoding of propositions as constant-only expressions, he has obtained the surprising result that constant-only full linear logic is undecidable [82]. He also shows that constant-only multiplicative-additive linear logic, like its propositional variant, is PSPACE-complete.

Quantifiers Recently there have been several other complexity results obtained regarding nicely formed fragments of first- and second-order linear logic. Among the most exciting of these are the undecidability of second-order propositional multiplicative linear logic [105, 96, 97] the NP-completeness of first-order multiplicative linear logic [108], and the NEXPTIME-completeness of first-order multiplicative-additive linear logic [108, 103].

1.5.3 Linear Lambda Calculus

Chapter 8 concerns a functional language LM that is related to languages investigated by Lafont [94], Abramsky [2], Wadler [150, 151], Chirimar, Gunter, and Riecke [41], Benton, Bierman, de Paiva, and Hyland [31], Troelstra [143], and others. The basic idea is to view linear logic in a Curry-Howard style, by assigning program constructors to inference rules. Thus an LM program can be seen as a notation for a linear logic proof, and the type of the program can be seen as the most general conclusion which can be inferred from the proof. Alternatively, one may view linear logic as a type system for LM, where linear formulas correspond to types.

Sequent versus Natural Deduction The Curry-Howard correspondence holds between the lambda calculus and a natural deduction system for intuitionistic logic. However, there are many natural deduction systems for linear logic, including an interesting graph structure called proof nets. There have been some attempts to use proof nets as a basis for forming proof terms, but proof nets suffer from some drawbacks in representing full propositional linear logic including additives, constants, and modals. Attempts to use the "raw" sequent calculus for intuitionistic linear logic, for example by Abramsky [2], have shown some promise.

The presentation here shares properties of both of the above approaches: a sequent calculus presentation of a natural deduction system for intuitionistic linear logic is used as the basis for the proof terms. The presentation of natural deduction for intuitionistic logic in the sequent calculus dates back to some of Gentzen's original work on the sequent calculus. The proof system here is a result of straightforward application of those same ideas to intuitionistic linear logic.

The main idea is that a natural deduction-style introduction rule is represented by sequent calculus rule which introduces the formula on the right of the turnstile. All of the "right" rules of the sequent calculus are preserved unchanged in the natural deduction style sequent system. The "left" rules, however, are changed dramatically, becoming "elimination" rules in the modified system. These elimination rules can be effectively simulated in the sequent calculus with one application of the left rule, and one application of cut on the introduced formula. The "left" rules of the sequent calculus can be derived in the natural deduction system with one application of the elimination rule, one application of identity, and one application of cut. Therefore the same sequents (terms with typings) are provable in both systems, perhaps up to some applications of cut. Many of these ideas for creating a natural deduction style system from a sequent system are due to Prawitz [131].

Showing that the two systems are equivalent, we immediately get a cross-fertilization of theorems between the two systems. Perhaps the most important such theorem is type soundness, or the subject-reduction theorem. This theorem states that if a term has a derivable typing, then the result of reducing the term still has the same type. If this theorem failed, there would be little reason to call a logical framework a "type system." since terms might have a certain type, but after reduction the terms would not have this type. Another important theorem proved here is the existence of a (unique) most general type for any linear term.

Looking at the systems informally, they are different proof systems for the same logic. The sequent system is conservative with respect to types: any type appearing anywhere in a cut-free proof appears as a subformula of the type of the conclusion. On the other hand, the natural deduction system is conservative with respect to terms: any term appearing anywhere in a cut-free proof appears as a subformula of the term in the conclusion. Both of these properties follow immediately from the cut-elimination theorems.

The guiding vision of the research in this chapter has been the development of a type system based on linear logic which would lend practical, useful advice to an ML-style compiler [119]. The main idea of studying such an application of linear logic dates back to the introduction of linear logic and very early work [57, 94]. Others have also studied the further application of linear type information in the study of garbage collection and array update-in-place [41, 155]. In the course of our investigations, we were led to consider decision problems and their complexity, but eventually these investigations led back to the development of the original vision.

2 Linear Proof Theory

‘contrariwise,’ continued Tweedledee, ‘if it was so, it might be; and if it were so, it would be; but as it isn’t, it ain’t. That’s logic’.
LEWIS CARROL (CHARLES LUTWIDGE DODSON) 1832-1898

Alice’s Adventures in Wonderland, 1865

Below is a brief presentation of the more foundational concepts of linear logic and proof theory necessary for the remainder of this book. The reader can find more detailed discussion of this introductory material in several works [56, 89, 57].

2.1 Basic Properties of Linear Logic

As mentioned earlier, because linear logic has an involutive negation, $(A^{\perp})^{\perp} = A$, one may formulate linear logic as a one-sided system. Informally, one may consider the ${}^{\perp}$R and ${}^{\perp}$L rules to be “built in” to the system as if these were structural rules. In this case, many rules are identified: $\otimes$R and $⅋$L become identical rules. One translates the sequent $\Sigma \vdash \Gamma$ into an equivalent sequent $\vdash (\Sigma)^{\perp}, \Gamma$, where the negation of a sequence of formulas is the sequence of their negations (in reverse order, if order is important).

One may then work in a one-sided system where all formulas appear on the right of the $\vdash$, and there are no sequent rules for connectives on the left of the $\vdash$. Some of the proofs by case analysis of the sequent calculus rules are greatly simplified by this, although there is some price for this convenience in the difficulty of comprehending complicated sequent proofs in the one sided system. This book uses both one sided and two sided sequents, as is convenient for the topic under consideration.

Also, because linear logic contains the exchange rule, one may consider sequents to be constructed out of *multisets* of formulas on each side of the $\vdash$. This alteration to the sequent calculus is similar to the standard use of *sets* on both sides of a $\vdash$ in classical and intuitionistic logic, but in linear logic the multiplicity of formulas is of crucial importance, while it is completely unimportant in classical logic. Assuming that sequents are built from multisets rather than sequences of formulas, one simply ignores all applications of the exchange rule. For the majority of this book we will make this assumption, although when noncommutative linear logic is considered in Section 4 a great deal more care will be taken with such matters.

We define a *positive* literal to be one of the given propositional symbols p_i or $p_i^{\perp}$ which occurs with positive polarity. A *negative* literal is one of these symbols which occurs with negative polarity.

The class of *subformulas* of a given formula or sequent is defined by the following: A is a subformula of A. If A is a subformula of B, then A is also a subformula of the following formulas: $?B$, $!B$, $B \otimes C$, $C \otimes B$, $B \parr C$, $C \parr B$, $B \multimap C$, and $C \multimap B$. Also, $A\{t/x\}$ for any t is a subformula of $\forall x A$ and $\exists x A$. If A is a subformula of B, then A is also a subformula of the sequents $\Sigma \vdash \Gamma_1, B, \Gamma_2$ and $\Sigma_1, B, \Sigma_2 \vdash \Gamma$.

We next review sequent systems for standard classical (LK) and intuitionistic (LJ) logic.

2.2 Classical Logic: LK

The rules of classical logic are given in Figure 2.1 below. This formulation of LK is nonstandard in that usually [56] sequents are built from sets with implicit contraction, weakening, and exchange. Here sequents are built from multisets with implicit exchange, but explicit contraction and weakening, in order to facilitate comparison with linear logic.

Gentzen [56] showed that LK is equivalent to previous axiomatic systems for classical reasoning due to Gilvenko and Hilbert and Ackermann.

For most people classical logic is "normal" logic. Familiar informal reasoning by contradiction can be formalized in classical logic. However, such reasoning is nonconstructive, and thus not allowed in intuitionistic logic

2.3 Intuitionistic Logic: LJ

Intuitionistic logic is a constructive logic. From existential formulas one can in principle derive constructive procedures to identify the existential witness. The sequent calculus LJ can be seen as a simple restriction on the classical sequent calculus LK: the right hand side of a sequent must contain exactly one formula. This prohibits use of contraction or weakening on the right hand side of a sequent, and places restrictions on the use of negation and rules for implication. Intuitionistic logic also enjoys a crisp and clean natural deduction system and therefore a nice term assignment for proofs (see Chapter 7).

Proof rules for intuitionistic logic are given in Figure 2.2. As for LK above, sequents are built from multisets (rather than the more usual sets), with implicit exchange, and explicit contraction and weakening. In LJ contraction and weakening are only allowed on the left side of $\vdash$. The formulas of intuitionistic logic do not include explicit negation, although $\Rightarrow$ provides implicit negation. One can define intuitionistic negation $\neg P$ as $P \Rightarrow \bot$.

I	$\dfrac{}{A \vdash A}$	$\dfrac{\Sigma \vdash \Delta, A \quad A, \Gamma \vdash \Theta}{\Sigma, \Gamma \vdash \Delta, \Theta}$	**Cut**
E L	$\dfrac{\Gamma_1, A, B, \Gamma_2 \vdash \Delta}{\Gamma_1, B, A, \Gamma_2 \vdash \Delta}$		
$\wedge$**L**	$\dfrac{\Sigma, A, B \vdash \Delta}{\Sigma, (A \wedge B) \vdash \Delta}$	$\dfrac{\Sigma \vdash A, \Delta \quad \Gamma \vdash B, \Theta}{\Sigma, \Gamma \vdash (A \wedge B), \Delta, \Theta}$	$\wedge$**R**
$\Rightarrow$**L**	$\dfrac{\Sigma \vdash A, \Delta \quad \Gamma, B \vdash \Theta}{\Sigma, \Gamma, (A \Rightarrow B) \vdash \Delta, \Theta}$	$\dfrac{\Sigma, A \vdash B, \Delta}{\Sigma \vdash (A \Rightarrow B), \Delta}$	$\Rightarrow$**R**
$\vee$**L**	$\dfrac{\Sigma, A \vdash \Delta \quad \Sigma, B \vdash \Theta}{\Sigma, (A \vee B) \vdash \Delta, \Theta}$	$\dfrac{\Sigma \vdash A, B, \Delta}{\Sigma \vdash (A \vee B), \Delta}$	$\vee$**R**
$\neg$ **L**	$\dfrac{\Sigma \vdash A, \Delta}{\Sigma, \neg A \vdash \Delta}$	$\dfrac{\Sigma, A \vdash \Delta}{\Sigma \vdash \neg A, \Delta}$	$\neg$ **R**
WL	$\dfrac{\Sigma \vdash \Delta}{\Sigma, B \vdash \Delta}$	$\dfrac{\Sigma \vdash \Delta}{\Sigma \vdash B, \Delta}$	**WR**
CL	$\dfrac{\Sigma, A, A \vdash \Delta}{\Sigma, A \vdash \Delta}$	$\dfrac{\Sigma \vdash A, A, \Delta}{\Sigma \vdash A, \Delta}$	**CR**
$\perp$**L**	$\dfrac{}{\perp \vdash}$	$\dfrac{\Sigma \vdash \Delta}{\Sigma \vdash \perp, \Delta}$	$\perp$**R**
1**L**	$\dfrac{\Sigma \vdash \Delta}{\Sigma, 1 \vdash \Delta}$	$\dfrac{}{\vdash 1}$	1**R**

Figure 2.1
Sequent Rules for Classical Logic (LK)

$$
\begin{array}{llll}
\mathbf{I} & \dfrac{}{A \vdash A} & \dfrac{\Sigma \vdash A \quad A, \Gamma \vdash B}{\Sigma, \Gamma \vdash B} & \mathbf{Cut} \\[2ex]
\mathbf{E\ L} & \dfrac{\Gamma_1, A, B, \Gamma_2 \vdash C}{\Gamma_1, B, A, \Gamma_2 \vdash C} & & \\[2ex]
\wedge\mathbf{L} & \dfrac{\Sigma, A, B \vdash C}{\Sigma, (A \wedge B) \vdash C} & \dfrac{\Sigma \vdash A \quad \Gamma \vdash B}{\Sigma, \Gamma \vdash (A \wedge B)} & \wedge\mathbf{R} \\[2ex]
\Rightarrow\mathbf{L} & \dfrac{\Sigma \vdash A \quad \Gamma, B \vdash C}{\Sigma, \Gamma, (A \Rightarrow B) \vdash C} & \dfrac{\Sigma, A \vdash B}{\Sigma \vdash (A \Rightarrow B)} & \Rightarrow\mathbf{R} \\[2ex]
\vee\mathbf{L} & \dfrac{\Sigma, A \vdash C \quad \Sigma, B \vdash C}{\Sigma, (A \vee B) \vdash C} & \dfrac{\Sigma \vdash A}{\Sigma \vdash (A \vee B)} & \vee\mathbf{R} \\[2ex]
\mathbf{WL} & \dfrac{\Sigma \vdash A}{\Sigma, B \vdash A} & & \\[2ex]
\mathbf{CL} & \dfrac{\Sigma, A, A \vdash B}{\Sigma, A \vdash B} & & \\[2ex]
\bot\mathbf{L} & \dfrac{}{\bot \vdash} & \dfrac{\Sigma \vdash}{\Sigma \vdash \bot} & \bot\mathbf{R} \\[2ex]
1\mathbf{L} & \dfrac{\Sigma \vdash A}{\Sigma, 1 \vdash A} & \dfrac{}{\vdash 1} & 1\mathbf{R}
\end{array}
$$

Figure 2.2
Sequent Rules for Intuitionistic Logic (LJ)

As he did for classical LK, Gentzen showed that LJ is equivalent to previous axiomatic systems for intuitionistic logic due to Gilvenko and Hilbert and Ackermann.

Classical logic is not conservative over intuitionistic logic: there are intuitionistic formulas that are provable in LK but not in LJ. For example, consider Peirce's Law: $((A \Rightarrow B) \Rightarrow A) \Rightarrow A$. Semantically this formula is true in classical logic because if A is true, then the whole formula is true because anything implies true, and if A is false then $(A \Rightarrow B)$ is true, since false implies anything, and thus $((A \Rightarrow B) \Rightarrow A)$ is false since true doesn't imply false, and thus the entire formula is true because false implies anything. Peirce's law has the following LK proof:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,I}{A \vdash B, A}\,WR}{\vdash A \Rightarrow B, A}\Rightarrow R \quad \dfrac{}{A \vdash A}\,I}{(A \Rightarrow B) \Rightarrow A \vdash A, A}\Rightarrow L}{(A \Rightarrow B) \Rightarrow A \vdash A}\,CR}{\vdash ((A \Rightarrow B) \Rightarrow A) \Rightarrow A}\Rightarrow R
$$

The CR inference above is critical to completing this proof: Peirce's Law is not provable in LJ. The rightmost A in the law must perform double duty, appearing in at least two axioms with the other two occurances of A. However, the sequent system LJ allows contraction and weakening only on the left of the sequent arrow, and the formulation of $\Rightarrow$L in LJ prevents the rightmost A from appearing in both branches. Thus this formula is not intuitionistically provable. However, one can translate any classically valid formula into an intuitionisticly valid formula by double negation. That is, a formula A is provable in LK if and only if its double-negation translation is provable in LJ.

2.4 Classical Linear Logic: CLL

The proof rules for classical linear logic are given in Figure 1.2 on page 18 and are reproduced in Appendix A.1.

2.5 Intuitionistic Linear Logic: ILL

The intuitionistic version of linear logic (ILL) is generated from (classical) linear logic (CLL) in a manner analogous to the derivation of intuitionistic logic (LJ) from classical logic (LK). ILL sequents are restricted to include exactly one formula on the right hand side of the $\vdash$. In linear logic there are some interesting twists which must be considered. To be clear we will refer to linear logic, as defined in Figure 1.2, as *classical* linear logic, or CLL, to emphasize the distinction between intuitionistic and classical versions of linear logic. The Gentzen-style inference rules for intuitionistic linear logic (ILL) given below in figure 2.3 are reproduced in Appendix A.2 for convenience [6].

Note that the $\perp$ connective is omitted from this presentation of ILL. If $\perp$ were included in the logic CLL would not be conservative over ILL, even for conclusion sequents which satisfy the intuitionistic restriction of one formula on the right hand side. For example, consider the formula $(((A \multimap (B \multimap \perp)) \multimap \perp) \multimap (A \otimes B))$. In considering a cut-free proof of this formula, we have no choice but to construct the following partial proof:

$$\dfrac{\begin{array}{c}\vdots\\ ((A \multimap (B \multimap \perp)) \multimap \perp) \vdash A \otimes B\end{array}}{\vdash (((A \multimap (B \multimap \perp)) \multimap \perp) \multimap (A \otimes B))} \multimap R$$

In CLL, one can complete the proof as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A} I \quad \dfrac{}{B \vdash B} I}{A, B \vdash A \otimes B} \otimes R}{A, B \vdash \perp, A \otimes B} \perp R}{A \vdash (B \multimap \perp), A \otimes B} \multimap R}{\vdash (A \multimap (B \multimap \perp)), A \otimes B} \multimap R \quad \dfrac{}{\perp \vdash}}{((A \multimap (B \multimap \perp)) \multimap \perp) \vdash A \otimes B} \multimap L}{\vdash (((A \multimap (B \multimap \perp)) \multimap \perp) \multimap (A \otimes B))} \multimap R$$

However, the exact use of the critical $\multimap$ L inference is not available in ILL, and it can be shown that this sequent is not provable in ILL. The same problem arises with the simpler example $(((A \multimap \perp) \multimap \perp) \multimap A)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A} I}{A \vdash \perp, A} \perp R}{\vdash (A \multimap \perp), A} \multimap R \quad \dfrac{}{\perp \vdash}}{((A \multimap \perp) \multimap \perp) \vdash A} \multimap L}{\vdash (((A \multimap \perp) \multimap \perp) \multimap A)} \multimap R$$

For a time it was conjectured that classical linear logic is conservative over ILL for formulas without the $\perp$ constant, but the following counterexample,

I	$\dfrac{}{A \vdash A}$	$\dfrac{\Sigma \vdash A \quad A, \Gamma \vdash B}{\Sigma, \Gamma \vdash B}$	**Cut**
E L	$\dfrac{\Gamma_1, A, B, \Gamma_2 \vdash C}{\Gamma_1, B, A, \Gamma_2 \vdash C}$		
$\otimes$**L**	$\dfrac{\Sigma, A, B \vdash C}{\Sigma, (A \otimes B) \vdash C}$	$\dfrac{\Sigma \vdash A \quad \Gamma \vdash B}{\Sigma, \Gamma \vdash (A \otimes B)}$	$\otimes$**R**
$\multimap$**L**	$\dfrac{\Sigma \vdash A \quad \Gamma, B \vdash C}{\Sigma, \Gamma, (A \multimap B) \vdash C}$	$\dfrac{\Sigma, A \vdash B}{\Sigma \vdash (A \multimap B)}$	$\multimap$**R**
$\oplus$**L**	$\dfrac{\Sigma, A \vdash C \quad \Sigma, B \vdash C}{\Sigma, (A \oplus B) \vdash C}$	$\dfrac{\Sigma \vdash A \quad \Sigma \vdash B}{\Sigma \vdash (A \,\&\, B)}$	**& R**
& L1	$\dfrac{\Sigma, A \vdash C}{\Sigma, (A \,\&\, B) \vdash C}$	$\dfrac{\Sigma \vdash A}{\Sigma \vdash (A \oplus B)}$	$\oplus$**R1**
& L2	$\dfrac{\Sigma, B \vdash C}{\Sigma, (A \,\&\, B) \vdash C}$	$\dfrac{\Sigma \vdash B}{\Sigma \vdash (A \oplus B)}$	$\oplus$**R2**
!WL	$\dfrac{\Sigma \vdash A}{\Sigma, !B \vdash A}$		
!DL	$\dfrac{\Sigma, A \vdash B}{\Sigma, !A \vdash B}$	$\dfrac{\Sigma \vdash A}{\Sigma \vdash ?A}$	**?DR**
!CL	$\dfrac{\Sigma, !A, !A \vdash B}{\Sigma, !A \vdash B}$		
?SL	$\dfrac{!\Sigma, A \vdash ?B}{!\Sigma, ?A \vdash ?B}$	$\dfrac{!\Sigma \vdash A}{!\Sigma \vdash !A}$	**!SR**
1**L**	$\dfrac{\Sigma \vdash A}{\Sigma, 1 \vdash A}$	$\dfrac{}{\vdash 1}$	1**R**
0**L**	$\dfrac{}{\Sigma, 0 \vdash A}$	$\dfrac{}{\Sigma \vdash \top}$	$\top$**R**

Figure 2.3
Sequent Rules for Intuitionistic Linear Logic (ILL)

due to Schellinx, shows there to be other cases of non-conservativity as well: $((A \multimap B) \multimap 0) \multimap (A \otimes \top)$.

However, conservativity does hold for fragments of ILL which do not include either false constant ($\perp$ or 0), and conservativity also holds for fragments of ILL without $\multimap$.

THEOREM 1 (CONSERVATIVITY) CLL is conservative over ILL for sequents not containing $\perp$ or 0. CLL is also conservative over ILL for sequents not containing $\multimap$.

This theorem may be proven by induction on the assumed cut-free CLL proofs. Essentially, one can show that any cut-free CLL proof of a sequent meeting the above restrictions satisfies the intuitionistic restriction to at most one formula on the right throughout. The only interesting inductive steps are for the false constants and $\multimap$ L. In the first statement of the proof, if a proof ends in $\multimap$ L, then either the CLL application of $\multimap$ L matches the intuitionistic rule for $\multimap$ L, in which case we can immediately appeal to the induction hypothesis, or one of the hypotheses has an empty consequent, or right hand side of the $\vdash$. In the later case, one appeals to the property that for sequents satisfying the above syntactic constraints, no sequent $\Gamma \vdash$ is provable in CLL.

The bottom line here is that classical linear logic CLL is conservative over intuitionistic linear logic ILL only if one *completely* eliminates negation. Negation is first present in the rules for $^\perp$ which allow formulas to move from one side of a sequent to another. For example, the formula $(A^\perp \multimap B^\perp) \multimap (B \multimap A)$ is provable in CLL (utilizing $^\perp$) but not in ILL. Negation may also be implicitly recovered with the use of $\multimap$ and either 0 or $\perp$ (the two flavors of *false*). Conservativity holds if none of these kinds of negation arise, which may be ensured by eliminating $^\perp$ and $\multimap$, or $^\perp$ and 0 and $\perp$.

2.6 Cut Elimination

The cut-elimination theorem, in general, states that whatever can be proven in the full version of a logic may also be proven without the use of the cut rule. This theorem is fundamental to linear logic, and was proven by Girard shortly after the introduction of the logic by presenting a cut-elimination procedure for proof nets [57]. Since Girard demonstrated the correspondence between proof nets and the sequent calculus presentation of linear logic, we could have relied on Girard's proof of cut-elimination. However, in later proofs, we make use of the syntax and exact form of a cut-elimination procedure for the sequent calculus formulation of linear logic. Girard's use of proof nets, and his reliance on the one-sided version of linear logic would complicate the construction of our later theorems. Thus a full cut-elimination proof is given here. Readers not interested in the gory details of sequent-calculus beurocracy can skip ahead to page 49 to see the resulting Theorem 2.

The following demonstration of the cut-elimination theorem consists of a linear logic proof normalization procedure which slowly eliminates cuts from any linear logic proof. The procedure may greatly increase the size of the proof, although of course it will still be a proof of the same sequent. For technical reasons, we add derived rules of inference, **Cut!** and **Cut?**, which simplify the proof of termination. We then give a set of reductions which apply to proofs which end in **Cut**, **Cut!**, or **Cut?**, and using these we eliminate all uses of **Cut**, **Cut!**, and **Cut?** from a proof.

The proof structure is very close to the well known proofs of cut-elimination in classical logic [64], but is complicated by the extra information which must be preserved in a linear proof. The **Cut!** and **Cut?** rules defined below are reminiscent of Gentzen's MIX rule [56], and serve the same purpose, which is to package together certain inference rules. As in Gentzen's work, we add these extra rules, and then show that they (along with **Cut**) may be eliminated entirely from any proof. Thus we show that these new rules and **Cut** are redundant in linear logic.

Let us begin with some definitions. First, we define the following new rules of inference,

$$\textbf{Cut!} \qquad \frac{\Theta \vdash \Sigma, !A \qquad \Gamma, (!A)^n \vdash \Delta}{\Theta, \Gamma \vdash \Sigma, \Delta} \qquad n \geq 1$$

$$\textbf{Cut?} \qquad \frac{\Theta \vdash \Sigma, (?A)^n \qquad \Gamma, ?A \vdash \Delta}{\Theta, \Gamma \vdash \Sigma, \Delta} \qquad n \geq 1$$

As mentioned in the last section, $(?A)^n$ denotes a multiset of formulas. For example, $(?A)^3 = ?A, ?A, ?A$. As stated in the side condition, the **Cut!** and **Cut?** rules are only applicable when n is at least 1.

We will use the symbol "**Cut***" as a general term for the original **Cut** rule or the new **Cut!** and **Cut?** rules ambiguously. These new rules of inference are derivable; they may be simulated by several applications of contraction (**!C** or **?C**) and one application of the standard **Cut** rule. The original **Cut** rule coincides with **Cut!** (or **Cut?**) when $n = 1$. Adding these extra derived rules of inference simplifies the termination argument substantially by packaging together some number of contractions with the cut that eliminates the contracted formula. This package is only opened when the contracted formulas are actually used with the application of the **!D** or **?D** rules, thrown away by the **!W** or **?W** rules, or split into two packages by the $\otimes$ R, $⅋$ L, $\multimap$ L, **Cut*** rules.

We will call a formula which appears in a hypothesis of an application of **Cut***, but which does not occur in the conclusion a *cut formula*. In the list of linear logic rules in Appendix A.1 the cut formula in the **Cut** rule is the formula named A, and in the **Cut!** and **Cut?** rules above, the cut formulas are $!A$ and $?A$.

We also define the *degree* of a **Cut*** to be the number of symbols in its cut formula. For concreteness, we define here what is meant by number of symbols. We will consider each propositional symbol p_i to be a single symbol. We also consider the negation of each propositional symbol ${p_i}^{\perp}$ to be a single symbol. Finally, we count each connective and constant, $\otimes, ⅋, \oplus, \&, ?, !, 1, \perp, 0, \top$, as a single symbol, but do not count parentheses. It is important to note that negation is defined, and therefore is not a connective. This method of accounting has the pleasant property that any linear logic formula A and its negation $A^{\perp}$ have exactly the same number of symbols. (One may prove this by induction on the structure of the formula A). Thus it does not matter which cut formula we count when determining the degree of a cut. We also define the *degree* of a proof to be the maximum degree of any cut in the proof, or zero if there are no cuts.

Recall that the *principal formula* of an application of an inference rule is defined to be any formula which is introduced by that rule. For convenience, we extend the notion of principal formula in the following nonstandard ways. We will consider any formula beginning with ? appearing on the right side, and any formulas prefixed with ! on the left of the conclusion of the **!S**, $\otimes$ R, $⅋$ L, $\multimap$ L, or **Cut*** rules to be principal. By this definition all formulas in the conclusion of **!S** are principal, and the only rule in which a formula beginning with ! may be principal on the right hand side is **!S**. This definition of principal formula simplifies the structure of the following proof somewhat.

Operationally, the cut-elimination procedure defined below first finds one of the "highest" cuts of maximal degree in the proof. That is, an application of **Cut*** for which all applications of **Cut*** in the derivation of either hypothesis is of smaller degree. Then a reduction is applied to that occurrence of **Cut***, which simplifies or eliminates it, although it may replicate some other portions of the original proof. We iterate this procedure to remove all cuts of some degree, and then iterate the entire procedure to eliminate all cuts. In this way, any linear logic proof may be normalized into one without any uses of the **Cut**, **Cut?** or **Cut!** rules, at the possible expense of an (worse than) exponential blowup in the size of the resulting proof tree.

Technically, we begin with a lemma which constitutes the heart of the proof of cut-elimination. Although the proof of this lemma is rather lengthy, the reasoning is straightforward, and the remainder of the proof of cut-elimination is quite simple.

LEMMA 1 (REDUCE ONE CUT) Given a proof of the sequent $\Theta \vdash \Gamma$ in linear logic which ends in an application of **Cut*** of degree $d > 0$, and where the degree of the proofs of both hypothesis is less than d, we may construct a proof of $\Theta \vdash \Gamma$ in linear logic of degree less than d.

Proof 1 By induction on the number of proof rules applied in the derivation of $\Theta \vdash \Gamma$.

Given a derivation which ends in a **Cut***, we perform case analysis on the rules which were applied immediately above the **Cut***. One of the following cases must apply to any such derivation:
1. The cut formula is *not* principal in one or both hypotheses.
2. The cut formula *is* principal in both hypotheses.

In each case we will provide a reduction, which may eliminate the cut entirely, or replace it with one or two smaller cuts. Since this is a proof by induction on the size of a derivation, one may view this proof as a procedure which pushes applications of **Cut*** of large degree up a derivation. Informally, this procedure pushes applications of **Cut*** up through proof rules where the cut formula is non-principal, until the critical point is reached where the cut formula is principal in both hypotheses. In Girard's proof of cut-elimination for linear logic using proof nets, the non-principal cases are circumvented by following proof links. In both approaches, however, the principal cases require significant detailed analysis.

2.6.1 Cut of Nonprincipal Formulas

If the derivation of a hypothesis ends in a rule yielding a non-principal cut formula, then the rule must be one of the following: $\otimes$ R, $\otimes$ L, $\parr$ R, $\parr$ L, $\oplus$ R, $\oplus$ L, & R, & L, **? W**, **? C**, **? D**, **! W**, **! C**, **! D**, $\perp$ R, 1 L, $\top$ R, 0 L, or **Cut***. The rules **I**, **! S**, **? S**, $\perp$ L, and 1 R are absent from this list since those rules have no non-principal formulas in their conclusions. The later analysis of principal formula cuts considers these three cases. Also, most of the following cases come in two directly analogous cases, such as $\otimes$ R vs $\parr$ L. We will only present one of each such pair of cases.

$\otimes$ R, or $\parr$ L If the last rule applied in one hypothesis is $\otimes$ **R**, the cut formula is not the main formula introduced by that application of $\otimes$ **R**, and the cut formula does not begin with ?, then we may propagate the **Cut*** upward, through the application of $\otimes$:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Theta_1 \vdash \Sigma, A\end{array} \qquad \begin{array}{c}\vdots\\ \Theta_2 \vdash B, \Delta, C\end{array}}{\Theta_1, \Theta_2 \vdash \Sigma, (A \otimes B), \Delta, C}{\otimes R} \qquad \begin{array}{c}\vdots\\ \Theta_3, C \vdash \Gamma\end{array}}{\Theta_1, \Theta_2, \Theta_3 \vdash \Sigma, (A \otimes B), \Delta, \Gamma}{Cut}$$

$$\Downarrow$$

$$\dfrac{\begin{array}{c}\vdots\\ \Theta_1 \vdash \Sigma, A\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ \Theta_2 \vdash B, \Delta, C\end{array} \qquad \begin{array}{c}\vdots\\ \Theta_3, C \vdash \Gamma\end{array}}{\Theta_2, \Theta_3 \vdash B, \Delta, \Gamma}{Cut}}{\Theta_1, \Theta_2, \Theta_3 \vdash \Sigma, (A \otimes B), \Delta, \Gamma}{\otimes R}$$

For the rules such as $\otimes$ R with two hypotheses, we give the reduction for the case where the non-principal cut formula appears in the right hand hypothesis of the $\otimes$ rule, and appears in one specific position in that sequent. The symmetric case of the cut formula appearing in the left hypothesis is very similar, and is always omitted. Since exchange is considered built-in to the system, sequents are considered multi-sets. Thus the exact position of formulas in sequents is unimportant. (Note that in noncommutative linear logic the relative position becomes vitally important.)

The proof ending in **Cut** after this transformation is smaller than the original proof, since the entire proof of $\Theta_1 \vdash \Sigma, A$, and the last application of $\otimes$ R are no longer above the **Cut**. Thus by induction on the size of proofs, we can construct the desired proof of degree less than d.

Note that the **Cut!** and **Cut?** rule only applies to formulas which begin with ! or ?, and thus this reduction, which is only used if the cut formula does not begin with ? or !, applies only to **Cut** and not to **Cut!** or **Cut?**. Thus, we have disambiguated this case, and write only **Cut**. Transformations are later presented in terms of **Cut***, in order to cover all three possibilities simultaneously. The reductions given later (in Section 2.6.2) handle the case of **Cut!** and **Cut?**.

$\parr$ R or $\otimes$ L If the last rule applied in one hypothesis is $\parr$ R, and the cut formula is not the main formula introduced by that application of $\parr$ R, then we may propagate the **Cut*** upward, through the application of $\parr$ R:

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \Theta_1 \vdash \Sigma, A, B, C\end{matrix}}{\Theta_1 \vdash \Sigma, (A \mathbin{⅋} B), C}{\scriptstyle ⅋ R} \qquad \begin{matrix}\vdots\\ \Theta_2, C \vdash \Gamma\end{matrix}}{\Theta_1, \Theta_2 \vdash \Sigma, (A \mathbin{⅋} B), \Gamma}{\scriptstyle Cut*}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \Theta_1 \vdash \Sigma, A, B, C\end{matrix} \qquad \begin{matrix}\vdots\\ \Theta_2, C \vdash \Gamma\end{matrix}}{\Theta_1, \Theta_2 \vdash \Sigma, A, B, \Gamma}{\scriptstyle Cut*}}{\Theta_1, \Theta_2 \vdash \Sigma, (A \mathbin{⅋} B), \Gamma}{\scriptstyle ⅋ R}$$

Again, the proof above the **Cut*** is smaller after this transformation, and thus by induction we have our result.

$\oplus$ **R or & L** Applications of **Cut*** involving the two symmetric $\oplus$ R rules (where the cut formula is not principle, that is, not introduced by this application of $\oplus$ R) may be eliminated in similar ways:

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \Theta_1 \vdash \Sigma, A, C\end{matrix}}{\Theta_1 \vdash \Sigma, (A \oplus B), C}{\scriptstyle \oplus R} \qquad \begin{matrix}\vdots\\ \Theta_2, C \vdash \Gamma\end{matrix}}{\Theta_1, \Theta_2 \vdash \Sigma, (A \oplus B), \Gamma}{\scriptstyle Cut*}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \Theta_1 \vdash \Sigma, A, C\end{matrix} \qquad \begin{matrix}\vdots\\ \Theta_2, C \vdash \Gamma\end{matrix}}{\Theta_1, \Theta_2 \vdash \Sigma, A, \Gamma}{\scriptstyle Cut*}}{\Theta_1, \Theta_2 \vdash \Sigma, (A \oplus B), \Gamma}{\scriptstyle \oplus R}$$

The second case of this rule is the same except the conclusion would contain the formula $(B \oplus A)$, instead of the formula $(A \oplus B)$ seen above.

& **R or** $\oplus$ **L** It is the elimination of this type of cut (among others) which may lead to an exponential blowup in the size of cut-free proofs.

$$
\dfrac{\dfrac{\begin{matrix}\vdots \\ \Theta_1 \vdash \Sigma, A, C\end{matrix} \quad \begin{matrix}\vdots \\ \Theta_1 \vdash \Sigma, B, C\end{matrix}}{\Theta_1 \vdash \Sigma, (A\&B), C}\&R \qquad \begin{matrix}\vdots \\ \Theta_2, C \vdash \Gamma\end{matrix}}{\Theta_1, \Theta_2 \vdash \Sigma, (A\&B), \Gamma}Cut*
$$

$$\Downarrow$$

$$
\dfrac{\dfrac{\begin{matrix}\vdots \\ \Theta_1 \vdash \Sigma, A, C\end{matrix} \quad \begin{matrix}\vdots \\ \Theta_2, C \vdash \Gamma\end{matrix}}{\Theta_1, \Theta_2 \vdash \Sigma, A, \Gamma}Cut* \quad \dfrac{\begin{matrix}\vdots \\ \Theta_1 \vdash \Sigma, B, C\end{matrix} \quad \begin{matrix}\vdots \\ \Theta_2, C \vdash \Gamma\end{matrix}}{\Theta_1, \Theta_2 \vdash \Sigma, B, \Gamma}Cut*}{\Theta_1, \Theta_2 \vdash \Sigma, (A\&B), \Gamma}\&R
$$

The increase in proof size comes from replicating the entire proof tree above $\Theta_2, C \vdash \Gamma$. Note that even though there are now two cuts instead of one, we may assume that both may be reduced in degree to less than d by induction on the size of the derivations. That is, there are fewer proof rules applied above each **Cut*** than there were above the single application of **Cut*** originally.

? W or ! W For this and the remaining cases, we omit discussion and simply indicate the reduction:

$$
\dfrac{\dfrac{\begin{matrix}\vdots \\ \Theta_1 \vdash \Sigma, A\end{matrix}}{\Theta_1 \vdash \Sigma, A, ?B}?W \qquad \begin{matrix}\vdots \\ \Theta_2 \vdash \Gamma, A^\perp\end{matrix}}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, ?B}Cut*
$$

$$\Downarrow$$

$$
\dfrac{\dfrac{\begin{matrix}\vdots \\ \Theta_1 \vdash \Sigma, A\end{matrix} \qquad \begin{matrix}\vdots \\ \Theta_2 \vdash \Gamma, A^\perp\end{matrix}}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma}Cut*}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, ?B}?W
$$

? C or ! C

$$\dfrac{\dfrac{\dfrac{\vdots}{\Theta_1 \vdash \Sigma, A, ?B, ?B}}{\Theta_1 \vdash \Sigma, A, ?B}{?C} \qquad \dfrac{\vdots}{\Theta_2 \vdash \Gamma, A^{\perp}}}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, ?B}{Cut*}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\Theta_1 \vdash \Sigma, A, ?B, ?B} \qquad \dfrac{\vdots}{\Theta_2 \vdash \Gamma, A^{\perp}}}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, ?B, ?B}{Cut*}}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, ?B}{?C}$$

? D or ! D

$$\dfrac{\dfrac{\dfrac{\vdots}{\Theta_1 \vdash \Sigma, A, B}}{\Theta_1 \vdash \Sigma, A, ?B}{?D} \qquad \dfrac{\vdots}{\Theta_2 \vdash \Gamma, A^{\perp}}}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, ?B}{Cut*}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\Theta_1 \vdash \Sigma, A, B} \qquad \dfrac{\vdots}{\Theta_2 \vdash \Gamma, A^{\perp}}}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, B}{Cut*}}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, ?B}{?D}$$

$\perp$ R or 1 L

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Theta_1 \vdash \Sigma, A\end{array}}{\Theta_1 \vdash \Sigma, A, \bot}\bot R \qquad \begin{array}{c}\vdots\\ \Theta_2 \vdash \Gamma, A^\bot\end{array}}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, \bot}Cut*$$

$$\Downarrow$$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Theta_1 \vdash \Sigma, A\end{array} \qquad \begin{array}{c}\vdots\\ \Theta_2 \vdash \Gamma, A^\bot\end{array}}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma}Cut*}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, \bot}\bot R$$

$\top$ **R or** 0 **L**

$$\dfrac{\dfrac{}{\Theta_1 \vdash \Sigma, A, \top}\top R \qquad \begin{array}{c}\vdots\\ \Theta_2 \vdash \Gamma, A^\bot\end{array}}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, \top}Cut*$$

$$\Downarrow$$

$$\dfrac{}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma, \top}\top R$$

Cut If the proof of one hypothesis ends in **Cut***, then we know that it has degree less than d, by the hypothesis of this lemma. If the cut formula of the lower degree d application of **Cut*** begins with ? on the right or ! on the left, then it is considered principal (by definition) in the upper application of **Cut***, and will be handled in Section 2.6.2. Otherwise, we know the formula does not begin with ? on the right or ! on the left, and thus the lower **Cut*** must actually be **Cut**.

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Theta_1 \vdash \Sigma, A, C\end{array} \quad \begin{array}{c}\vdots\\ \Theta_2, C \vdash \Gamma\end{array}}{\Theta_1, \Theta_2 \vdash \Sigma, A, \Gamma}\,Cut* \qquad \begin{array}{c}\vdots\\ \Theta_3 \vdash \Delta, A^{\perp}\end{array}}{\Theta_1, \Theta_2, \Theta_3 \vdash \Sigma, \Delta, \Gamma}\,Cut$$

$$\Downarrow$$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Theta_1 \vdash \Sigma, A, C\end{array} \quad \begin{array}{c}\vdots\\ \Theta_3 \vdash \Delta, A^{\perp}\end{array}}{\Theta_1, \Theta_3 \vdash \Sigma, \Delta, C}\,Cut \qquad \begin{array}{c}\vdots\\ \Theta_2, C \vdash \Gamma\end{array}}{\Theta_1, \Theta_2, \Theta_3 \vdash \Sigma, \Delta, \Gamma}\,Cut*$$

Here we know that the number of symbols in the formula A is d, and the number of symbols in the formula C is less than d. Thus by induction we know that we can construct a proof of degree less than d of $\vdash \Sigma, \Delta, C$, and from that we can construct our desired proof of $\Theta_1, \Theta_2, \Theta_3 \vdash \Sigma, \Delta, \Gamma$.

2.6.2 Cut of Principal Formulas

If the proof of each hypothesis ends in a rule with the cut formula as its principal formula, then the two last rules above the cut must be one of these combinations: **I** versus any, $\otimes$ **R** versus $\otimes$ **L**, ⅋ **R** versus ⅋ **L**, & **R** versus & **L**, $\oplus$ **R** versus $\oplus$ **L**, **? W** versus **! S**, **? C** versus **! S**, **? D** versus **! S**, **! S** versus **! S**, **Cut*** versus **! S**, $\otimes$ versus **! S**, or $\perp$ versus 1. Since all formulas in the conclusion of **! S** are considered principal, the analysis of **! S** at this stage of the proof is rather complex.

In many of these cases, we know that the **Cut!** and **Cut?** rules are inapplicable, since the cut formula has just been introduced, and it does not begin with a ! or ?. When we know this, we will disambiguate the reduction, and show the applications of **Cut**, **Cut?** and **Cut!** separately.

I versus Any If the last rule applied in either hypothesis is **I** (identity), then regardless of the rule applied in the other hypothesis we may remove the cut, and the application of identity:

$$\dfrac{\dfrac{}{p_i \vdash p_i}\,I \qquad \begin{array}{c}\vdots\\ \Theta, p_i \vdash \Gamma\end{array}}{\Theta, p_i \vdash \Gamma}\,Cut \quad \Longrightarrow \quad \begin{array}{c}\vdots\\ \Theta, p_i \vdash \Gamma\end{array}$$

The symmetric case (where p_i appears on the right in the non-axiom branch of the proof) is similar. Note that the identity axiom only applies to atomic propositions, and thus we know that **Cut!** and **Cut?** are inapplicable.

$\otimes$ R versus $\otimes$ L

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \Theta_1 \vdash \Sigma, A\end{matrix} \qquad \begin{matrix}\vdots\\ \Theta_2 \vdash B, \Delta\end{matrix}}{\Theta_1, \Theta_2 \vdash \Sigma, (A \otimes B), \Delta}{\scriptstyle \otimes R} \qquad \dfrac{\begin{matrix}\vdots\\ \Theta_3, A, B \vdash \Gamma\end{matrix}}{\Theta_3, (A \otimes B) \vdash \Gamma}{\scriptstyle \otimes L}}{\Theta_1, \Theta_2, \Theta_3 \vdash \Sigma, \Gamma, \Delta}{\scriptstyle Cut}$$

$$\Downarrow$$

$$\dfrac{\begin{matrix}\vdots\\ \Theta_1 \vdash \Sigma, A\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\\ \Theta_2 \vdash B, \Delta\end{matrix} \qquad \begin{matrix}\vdots\\ \Theta_3, A, B \vdash \Gamma\end{matrix}}{\Theta_2, \Theta_3, B \vdash \Gamma, \Delta}{\scriptstyle Cut}}{\Theta_1, \Theta_2, \Theta_3 \vdash \Gamma, \Delta, \Sigma}{\scriptstyle Cut}$$

In this case, as in most of the principal formula cut-elimination steps, we need not appeal to the induction hypothesis of this lemma. We have eliminated the **Cut** of degree d, and replaced it with two applications of **Cut** of degree smaller than d.

The case of $\parr$ R versus $\parr$ L is similar to this one, and is omitted.

& R versus & L

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \Theta_1 \vdash \Sigma, A\end{matrix} \qquad \begin{matrix}\vdots\\ \Theta_1 \vdash \Sigma, B\end{matrix}}{\Theta_1 \vdash \Sigma, (A \& B)}{\scriptstyle \& R} \qquad \dfrac{\begin{matrix}\vdots\\ \Theta_2, A \vdash \Gamma\end{matrix}}{\Theta_2, (A \& B) \vdash \Gamma}{\scriptstyle \& L}}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma}{\scriptstyle Cut}$$

$$\Downarrow$$

$$\dfrac{\begin{matrix}\vdots\\ \Theta_1 \vdash \Sigma, A\end{matrix} \qquad \begin{matrix}\vdots\\ \Theta_2, A \vdash \Gamma\end{matrix}}{\Theta_1, \Theta_2 \vdash \Sigma, \Gamma}{\scriptstyle Cut}$$

The symmetric case of & L is similar, as are the two cases of $\oplus$ R versus $\oplus$ L, all of which are omitted. We need not appeal to the induction hypothesis, and the cut formula does not begin with ?, and thus we know that **Cut?** does not apply, and similarly for ! and the **Cut!** rule.

! W versus !S For this and subsequent cases involving **!S** and **?S**, 'packaging' is a useful analogy. We build packages containing a number of contractions and a single **Cut!** when we reduce principal cases involving **! C** versus **!S**. We shrink the package in cases of **! W** versus **!S**, and we actually use the contents of the package as cases of **! D** versus **!S**. We let packages pass by each other at cases of **!S** versus **!S**, and at cases of **Cut!** versus **!S** and of $\otimes$ versus **!S** we break one package into two. This same intuition applies to the dual case involving ?.

For this case, **! W** versus **!S**, there are two possibilities, depending on whether the cut in question eliminates more than one occurrence of the cut formula from the weakened sequent. Informally, the possibilities turn on whether there is only one thing in the package. If so, we don't need the package. If there are more things in the package, we shrink the package.

In the first possibility, the cut eliminates the one occurrence of the cut formula introduced by the **! W** rule, and thus this application of cut may be eliminated entirely:

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \Theta_1 \vdash \Sigma\end{matrix}}{\Theta_1, !A \vdash \Sigma}{\scriptstyle !W} \qquad \dfrac{\begin{matrix}\vdots\\ !\Theta_2 \vdash ?\Gamma, A^{\perp}\end{matrix}}{!\Theta_2 \vdash ?\Gamma, !A}{\scriptstyle !S}}{\Theta_1, !\Theta_2 \vdash \Sigma, ?\Gamma}{\scriptstyle Cut*} \quad \Longrightarrow \quad \dfrac{\begin{matrix}\vdots\\ \underline{\Theta_1 \vdash \Sigma}_{\,!W}\\ \vdots\end{matrix}}{\Theta_1, !\Theta_2 \vdash \Sigma, ?\Gamma}{\scriptstyle ?W}$$

However, the second possibility, where the **Cut*** is actually a **Cut!** and eliminates more than one occurrence of the cut formula from the weakened sequent, we perform the following reduction:

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \Theta_1(!A)^{n-1} \vdash \Sigma\end{matrix}}{\Theta_1(!A)^{n} \vdash \Sigma}{\scriptstyle !W} \qquad \dfrac{\begin{matrix}\vdots\\ !\Theta_2 \vdash ?\Gamma, A\end{matrix}}{!\Theta_2 \vdash ?\Gamma, !A}{\scriptstyle !S}}{\Theta_1, !\Theta_2 \vdash \Sigma, ?\Gamma}{\scriptstyle Cut!}$$

$$\Downarrow$$

$$\dfrac{\begin{matrix}\vdots\\ \Theta_1, (!A)^{n-1} \vdash \Sigma\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\\ !\Theta_2 \vdash ?\Gamma, A\end{matrix}}{!\Theta_2 \vdash ?\Gamma, !A}{\scriptstyle !S}}{\Theta_1, !\Theta_2 \vdash \Sigma, ?\Gamma}{\scriptstyle Cut!}$$

In the first possibility we have our result immediately, since the **Cut*** has been eliminated. In the second possibility, we appeal to the induction hypothesis.

The **? W** versus **? S** case is similar.

! C versus ! S In this case we make critical use of the **Cut!** rule. Without this extra rule of inference this reduction is especially difficult to formulate correctly, and the induction required is complicated.

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \Theta_1, !A, !A \vdash \Sigma\end{matrix}}{\Theta_1, !A \vdash \Sigma}{!C} \qquad \dfrac{\begin{matrix}\vdots\\ !\Theta_2 \vdash ?\Gamma, A\end{matrix}}{!\Theta_2 \vdash ?\Gamma, !A}{!S}}{\Theta_1, !\Theta_2 \vdash \Sigma, ?\Gamma}{Cut*}$$

$$\Downarrow$$

$$\dfrac{\begin{matrix}\vdots\\ \Theta_1, !A, !A \vdash \Sigma\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\\ !\Theta_2 \vdash ?\Gamma, A\end{matrix}}{!\Theta_2 \vdash ?\Gamma, !A}{!S}}{\Theta_1, !\Theta_2 \vdash \Sigma, ?\Gamma}{Cut!}$$

Here we know that the cut formula begins with a !, and thus **Cut!** may apply to it. We thus produce a **Cut!** regardless of whether the original **Cut*** was a **Cut** or a **Cut!**.

The **? C** versus **? S** case is similar.

! D versus ! S As for the previous **! W** versus **! S** case, here we have two cases, depending on whether the **Cut*** in question eliminates more than one occurrence of the cut formula from the derelicted (**!D**) sequent. Again, informally, the two cases turn on the size of the package. If there is only one thing in the package, we simple make use of it, and throw away the wrapping. If there are more thing in the package, we take one out, and move the smaller package along its way.

In the first case, the cut eliminates the one occurrence of the cut formula introduced by the **! D** rule, and thus the following reduction applies:

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \Theta_1, A \vdash \Sigma\end{matrix}}{\Theta_1, !A \vdash \Sigma}{!D} \qquad \dfrac{\begin{matrix}\vdots\\ !\Theta_2 \vdash ?\Gamma, A\end{matrix}}{!\Theta_2 \vdash ?\Gamma, !A}{!S}}{\Theta_1, !\Theta_2 \vdash \Sigma, ?\Gamma}{Cut*}$$

$$\Downarrow$$

$$\dfrac{\begin{matrix}\vdots\\ \Theta_1, A \vdash \Sigma\end{matrix} \qquad \begin{matrix}\vdots\\ !\Theta_2 \vdash ?\Gamma, A\end{matrix}}{\Theta_1, !\Theta_2 \vdash \Sigma, ?\Gamma}{Cut}$$

However, in the second case, where the cut is actually a **Cut!** and eliminates more than one occurrence of the cut formula from the derelicted sequent, we perform the following reduction:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Theta_1, !A^{n-1}, A \vdash \Sigma\end{array}}{\Theta_1, !A^{n} \vdash \Sigma}\,{\scriptstyle !D} \qquad \dfrac{\begin{array}{c}\vdots\\ !\Theta_2 \vdash ?\Gamma, A\end{array}}{!\Theta_2 \vdash ?\Gamma, !A}\,{\scriptstyle !S}}{\Theta_1, !\Theta_2 \vdash \Sigma, ?\Gamma}\,{\scriptstyle Cut!}$$

$$\Downarrow$$

$$\begin{array}{c}\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Theta_1, !A^{n-1}, A \vdash \Sigma\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ !\Theta_2 \vdash ?\Gamma, A\end{array}}{!\Theta_2 \vdash ?\Gamma, !A}\,{\scriptstyle !S}}{\Theta_1, !\Theta_2, A \vdash \Sigma, ?\Gamma}\,{\scriptstyle Cut!} \qquad \begin{array}{c}\vdots\\ !\Theta_2 \vdash ?\Gamma, A\end{array}}{\Theta_1, !\Theta_2, !\Theta_2 \vdash \Sigma, ?\Gamma, ?\Gamma}\,{\scriptstyle Cut}\\ \overline{}\,{\scriptstyle !C}\\ \vdots\\ \dfrac{}{\Theta_1, !\Theta_2 \vdash \Sigma, ?\Gamma}\,{\scriptstyle ?C}\end{array}$$

Note that the second case requires the duplication of the proof above the application of **!S**. Since A has fewer symbols than $!A$, the lower **Cut** in the second case is of degree smaller than d. By induction, we may assume that the upper application of **Cut!** is reducible in degree.

! S versus ! S

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ !\Theta_1, !A \vdash ?\Sigma, B\end{array}}{!\Theta_1, !A \vdash ?\Sigma, !B}\,{\scriptstyle !S} \qquad \dfrac{\begin{array}{c}\vdots\\ !\Theta_2 \vdash ?\Gamma, A\end{array}}{!\Theta_2 \vdash ?\Gamma, !A}\,{\scriptstyle !S}}{!\Theta_1, !\Theta_2 \vdash ?\Sigma, ?\Gamma, !B}\,{\scriptstyle Cut*}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ !\Theta_1, A \vdash ?\Sigma, B\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ !\Theta_2 \vdash ?\Gamma, A\end{array}}{!\Theta_2 \vdash ?\Gamma, !A}\,{\scriptstyle !S}}{!\Theta_1, !\Theta_2 \vdash ?\Sigma, ?\Gamma, B}\,{\scriptstyle Cut*}}{!\Theta_1, !\Theta_2 \vdash ?\Sigma, ?\Gamma, !B}\,{\scriptstyle !S}$$

Here we appeal to the induction hypothesis to produce a proof degree less than d of $!\Theta_1, !\Theta_2 \vdash ?\Sigma, ?\Gamma, B$, and then construct the desired proof from that.

Cut* versus ! S There are two possibilities here, which correspond to whether it is necessary to split a package into two pieces. The case where the package needs to be split is one of the most tricky aspects of the entire cut-elimination procedure.

If the lower application of **Cut*** is applied to formulas which may all be found in one hypothesis of the upper application of **Cut***, then we apply the same reduction as in the non-principal **Cut** case (Section 2.6.1):

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Theta_1 \vdash \Sigma, A\end{array} \qquad \begin{array}{c}\vdots\\ \Theta_2, A, !C^n \vdash \Delta\end{array}}{\Theta_1, \Theta_2, !C^n \vdash \Sigma, \Delta}{\scriptstyle Cut*} \qquad \dfrac{\begin{array}{c}\vdots\\ !\Theta_3 \vdash ?\Gamma, C\end{array}}{!\Theta_3 \vdash ?\Gamma, !C}{\scriptstyle !S}}{\Theta_1, \Theta_2, !\Theta_3 \vdash \Sigma, \Delta, ?\Gamma}{\scriptstyle Cut*}$$

$$\Downarrow$$

$$\dfrac{\begin{array}{c}\vdots\\ \Theta_1 \vdash \Sigma, A\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ \Theta_2, A, !C^n \vdash \Delta\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ !\Theta_3 \vdash ?\Gamma, C\end{array}}{!\Theta_3 \vdash ?\Gamma, !C}{\scriptstyle !S}}{\Theta_2, !\Theta_3, A \vdash \Delta, ?\Gamma}{\scriptstyle Cut*}}{\Theta_1, \Theta_2, !\Theta_3 \vdash \Sigma, \Delta, ?\Gamma}{\scriptstyle Cut*}$$

In the more complex case, when the cut formulas descend from both hypotheses of the upper **Cut***,we use the following reduction:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Theta_1, !C^m \vdash \Sigma, A\end{array} \qquad \begin{array}{c}\vdots\\ \Theta_2, !C^n, A \vdash \Delta\end{array}}{\Theta_1, \Theta_2, !C^{n+m} \vdash \Sigma, \Delta}{\scriptstyle Cut*} \qquad \dfrac{\begin{array}{c}\vdots\\ !\Theta_3 \vdash ?\Gamma, C\end{array}}{!\Theta_3 \vdash ?\Gamma, !C}{\scriptstyle !S}}{\Theta_1, \Theta_2, !\Theta_3 \vdash \Sigma, \Delta, ?\Gamma}{\scriptstyle Cut!}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Theta_1, !C^m \vdash \Sigma, A\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ !\Theta_3 \vdash ?\Gamma, C\end{array}}{!\Theta_3 \vdash ?\Gamma, !C}{\scriptstyle !S}}{\Theta_1, !\Theta_3 \vdash \Sigma, ?\Gamma, A}{\scriptstyle Cut!} \qquad \dfrac{\begin{array}{c}\vdots\\ \Theta_2, !C^n \vdash \Delta, A\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ !\Theta_3 \vdash ?\Gamma, C\end{array}}{!\Theta_3 \vdash ?\Gamma, !C}{\scriptstyle !S}}{\Theta_2, !\Theta_3 \vdash \Delta, ?\Gamma, A}{\scriptstyle Cut!}}{\Theta_1, !\Theta_3, \Theta_2, !\Theta_3 \vdash \Sigma, ?\Gamma, \Delta, ?\Gamma}{\scriptstyle Cut*}}{\begin{array}{c}\vdots\end{array}}{\scriptstyle !C}$$

$$\dfrac{}{\Theta_1, \Theta_2, !\Theta_3 \vdash \Sigma, \Delta, ?\Gamma,}{\scriptstyle ?C}$$

$\otimes$ **R versus ! S** There are two possibilities here, which correspond to whether it is necessary to split a package into two pieces. The case where the package needs to be split is again one of the most tricky aspects of the entire cut-elimination procedure.

If **Cut*** is applied to formulas which may all be found in one hypothesis of $\otimes$, then we apply the same reduction as in the non-principal $\otimes$ case (Section 2.6.1):

$$
\begin{prooftree}
\AxiomC{$\vdots$}\noLine\UnaryInfC{$\Theta_1 \vdash \Sigma, A$}
\AxiomC{$\vdots$}\noLine\UnaryInfC{$\Theta_2, !C^n \vdash B, \Delta$}
\RightLabel{$\otimes R$}\BinaryInfC{$\Theta_1, \Theta_2, !C^n \vdash \Sigma, (A \otimes B), \Delta$}
\AxiomC{$\vdots$}\noLine\UnaryInfC{$!\Theta_3 \vdash ?\Gamma, C$}
\RightLabel{$!S$}\UnaryInfC{$!\Theta_3 \vdash ?\Gamma, !C$}
\RightLabel{$Cut*$}\BinaryInfC{$\Theta_1, \Theta_2, !\Theta_3 \vdash \Sigma, (A \otimes B), \Delta, ?\Gamma$}
\end{prooftree}
$$

$$\Downarrow$$

$$
\begin{prooftree}
\AxiomC{$\vdots$}\noLine\UnaryInfC{$\Theta_1 \vdash \Sigma, A$}
\AxiomC{$\vdots$}\noLine\UnaryInfC{$\Theta_2, !C^n \vdash B, \Delta$}
\AxiomC{$\vdots$}\noLine\UnaryInfC{$!\Theta_3 \vdash ?\Gamma, C$}
\RightLabel{$!S$}\UnaryInfC{$!\Theta_3 \vdash ?\Gamma, !C$}
\RightLabel{$Cut*$}\BinaryInfC{$\Theta_2, !\Theta_3 \vdash B, \Delta, ?\Gamma$}
\RightLabel{$\otimes R$}\BinaryInfC{$\Theta_1, \Theta_2, !\Theta_3 \vdash \Sigma, (A \otimes B), \Delta, ?\Gamma$}
\end{prooftree}
$$

In the more complex case, when the cut formulas descend from both hypotheses of $\otimes$, we use the following reduction to push the cut above the $\otimes$ rule.

$$
\begin{prooftree}
\AxiomC{$\vdots$}\noLine\UnaryInfC{$\Theta_1, !C^m \vdash \Sigma, A$}
\AxiomC{$\vdots$}\noLine\UnaryInfC{$\Theta_2, !C^n \vdash B, \Delta$}
\RightLabel{$\otimes R$}\BinaryInfC{$\Theta_1, \Theta_2, !C^{n+m} \vdash \Sigma, (A \otimes B), \Delta$}
\AxiomC{$\vdots$}\noLine\UnaryInfC{$!\Theta_3 \vdash ?\Gamma, C$}
\RightLabel{$!S$}\UnaryInfC{$!\Theta_3 \vdash ?\Gamma, !C$}
\RightLabel{$Cut!$}\BinaryInfC{$\Theta_1, \Theta_2, !\Theta_3 \vdash \Sigma, (A \otimes B), \Delta, ?\Gamma$}
\end{prooftree}
$$

$$\Downarrow$$

$$
\begin{prooftree}
\AxiomC{$\vdots$}\noLine\UnaryInfC{$\Theta_1, !C^m \vdash \Sigma, A$}
\AxiomC{$\vdots$}\noLine\UnaryInfC{$!\Theta_3 \vdash ?\Gamma, C$}
\RightLabel{$!S$}\UnaryInfC{$!\Theta_3 \vdash ?\Gamma, !C$}
\RightLabel{$Cut!$}\BinaryInfC{$\Theta_1, !\Theta_3 \vdash \Sigma, ?\Gamma, A$}
\AxiomC{$\vdots$}\noLine\UnaryInfC{$\Theta_2, !C^n \vdash B, \Delta$}
\AxiomC{$\vdots$}\noLine\UnaryInfC{$!\Theta_3 \vdash ?\Gamma, C$}
\RightLabel{$!S$}\UnaryInfC{$!\Theta_3 \vdash ?\Gamma, !C$}
\RightLabel{$Cut!$}\BinaryInfC{$\Theta_2, !\Theta_3 \vdash B, \Delta, ?\Gamma$}
\RightLabel{$\otimes R$}\BinaryInfC{$\Theta_1, !\Theta_3, \Theta_2, !\Theta_3 \vdash \Sigma, ?\Gamma, (A \otimes B), \Delta, ?\Gamma$}
\RightLabel{$!C$}\UnaryInfC{$\vdots$}
\RightLabel{$?C$}\UnaryInfC{$\Theta_1, \Theta_2, !\Theta_3 \vdash \Sigma, (A \otimes B), \Delta, ?\Gamma$}
\end{prooftree}
$$

1 R versus 1 L

$$\dfrac{\dfrac{}{\vdash 1}{1R} \qquad \dfrac{\begin{array}{c}\vdots\\ \Theta \vdash \Gamma\end{array}}{\Theta, 1 \vdash \Gamma}{1L}}{\Theta \vdash \Gamma}{Cut} \quad \Longrightarrow \quad \begin{array}{c}\vdots\\ \Theta \vdash \Gamma\end{array}$$

Again, we know that the **Cut*** involved here is **Cut**, since the formula 1 was just introduced, and does not begin with ! or ?.

This exhausts all the cases.

Thus, we have a procedure which given a proof which ends in **Cut*** of degree d, and which has no applications of **Cut*** in the proof of either hypothesis of degree greater than or equal to d, produces a proof of degree less than d.

LEMMA 2 (LOWER-DEGREE-CUTS) If a sequent is provable in linear logic with a proof of degree $d > 0$, then it is provable in linear logic with a proof of degree less than d.

Proof 2 By induction on the height of the derivation tree of the conclusion, we show that given any proof of degree d of $\Theta \vdash \Gamma$ in propositional linear logic, we may find a (possibly much larger) proof of $\Theta \vdash \Gamma$ in linear logic of degree less than d.

We examine the proof of $\Theta \vdash \Gamma$. Since the degree of this proof is greater than zero, there must be some **Cut*** in the proof. If the last rule is not **Cut***, then by induction we may form proofs of its hypotheses of degree less than d. Applying the same rule to the resulting reduced degree hypotheses produces the desired proof of degree less than d.

In the case that the last rule is **Cut***, we have the following situation for some Θ_1 and Θ_2 which together (in multiset union) make up Θ, and similarly for Γ_1, Γ_2 which make up Γ: (where at most one of n, m is greater than one:

$$\dfrac{\begin{array}{c}\vdots\\ \Theta_1 \vdash \Gamma_1, A^n\end{array} \qquad \begin{array}{c}\vdots\\ \Theta_2, A^m \vdash \Gamma_2\end{array}}{\Theta \vdash \Gamma}{Cut*}$$

By induction, we can produce proofs of $\Theta_1 \vdash \Gamma_1, A$ and $\Theta_2, A \vdash \Gamma_2$ of degree less than d. By a single application of Lemma 1 to the resulting proof constructed from the modified hypotheses, we obtain a proof of $\Theta \vdash \Gamma$ of degree less than d.

THEOREM 2 (CUT-ELIMINATION) If a sequent is provable in linear logic, then it is provable in linear logic without using the **Cut** rule.

Proof 3 By induction on the degree of the assumed proof. We may apply Lemma 2 at each inductive step, and at the base case the degree of the proof is zero, so therefore by definition of proof degree there are no cuts, and we have our desired cut-free proof.

Note that the proof can explode hyperexponentially in size during the cut-elimination process.

2.7 Subformula Property

We have demonstrated that all cuts may be eliminated from a proof, at the possible expense of increasing the size of the proof hyperexponentially. This normalization is worthwhile, however, since cut-free proofs have useful properties. One such property is the subformula property.

COROLLARY 2.7.1 (SUBFORMULA PROPERTY) Any formula of any sequent in any cut-free proof of $\Theta \vdash \Gamma$ is a subformula of Θ or Γ.

Proof 4 Each rule of linear logic except **Cut** has the property that every subformula of the hypotheses is also a subformula of the conclusion. For example, in the $\otimes$ R rule, any subformula of either hypothesis is either a subformula of $\Gamma_1, \Gamma_2, \Sigma_1, \Sigma_2, A$, or B. However, any such subformula is also a subformula of the conclusion. In fact, we may have "added" a subformula: $(A \otimes B)$ is a subformula of the conclusion, but might not be a subformula of the hypotheses.

Therefore, by induction on the size of proofs, we have that any subformula of any step of a cut-free proof of a sequent is a subformula of the original sequent.

It is easy to see that the subformula property is not true of proofs with cut: the formula A in the hypotheses of **Cut** might not appear in the conclusion. The main historical interest in the subformula property is that it implies consistency of the logic.

2.8 Polarity

Another useful property of cut-free proofs is stated using *polarity.* We define the polarity of a formula to be the number of (implicit) negations that surround it. Formally, we define the polarity of a formula based on the following sets of transformations, beginning with $[\Sigma]- \vdash [\Delta]^+$.

$$
\begin{array}{rcl}
[A \multimap B]^+ & = & [A]^- \multimap [B]^+ \\
[A \otimes B]^+ & = & [A]^+ \otimes [B]^+ \\
[\Sigma, A]^+ & = & [\Sigma]^+, [A]^+ \\
[A^\perp]^+ & = & ([A]^-)^\perp \\
[A \multimap B]^- & = & [A]^+ \multimap [B]^- \\
[A \otimes B]^- & = & [A]^- \otimes [B]^- \\
[\Sigma, A]^- & = & [\Sigma]^-, [A]^- \\
[A^\perp]^- & = & ([A]^+)^\perp
\end{array}
$$

The polarity of an instance of a formula A in a sequent $\Sigma \vdash \Delta$ is given by the sign of the superscript on A in $[\Sigma]^-$ or $[\Delta]^+$. That is, if an instance of formula A ends up as $[A]^+$, then it is of *positive* polarity. If an instance of formula A ends up as $[A]^-$, then it is of *negative* polarity.

Polarity is preserved throughout cut-free proofs, as stated formally below.

LEMMA 3 If a formula A has polarity p in an occurrence in a sequent in a cut-free proof of $\Gamma \vdash \Delta$, then A has polarity p in $\Gamma \vdash \Delta$.

2.9 Permutabilities of Linear Logic

In the previous section, the cut-elimination theorem was proved for linear logic. One may view a cut-free proof as the normal form of an infinite class of proofs with cut of the same formula. Thus the process of cut-elimination is often referred to as "proof normalization." In this section we turn our attention to a set of more specific normal form theorems for linear logic proofs. Specifically, we present a set of permutability theorems which may be used to convert proofs into various normal forms. Work along these lines has been fruitful in the study of other proof-theoretic properties of logics and has had an impact on the design of logic programming languages [90, 156, 122, 9, 108, 28, 50, 140, 51].

Formally, the following is a list of some permutabilities and impermutabilities of linear logic. The table lists only the permutabilities for connectives as they appear on the right hand side of a sequent. From this one may easily extrapolate all possible permutabilities of two-sided linear logic, using the definition of negation to consider the behavior of the dual operator as it appears on the right.

A rule R for the main connective of a formula C is permutable below another rule R' for the main connective of formula C' (C not a subformula of C') if for any proof with C analyzed immediately above C', then there is a proof with C analyzed immediately below C'. Using other terminology (see Bellin [28, 51]), an inference R permutes below an inference R' if and only if for any proof where R occurs immediately above R', and the principal formula of R is not active in R', there is a proof where R occurs immediately below R'. It is easiest to consider cut-free proofs, although one may also study permutabilities involving cut if one is careful with the definitions.

A stricter definition of permutability could require that the set of hypotheses occuring above the upper rule and above the lower rule but not including the conclusion of the upper rule be identical before and after the permutation. This disallows, for example, the proof modifications needed to ensure the permutability of & with itself. In the context of non-logical theories described in the next section, the stricter form of definition is often useful.

Any numeral in the following table should be read as "the connective of this column cannot always permute below the connective of this row." For example, number 9 shows that $\exists$ cannot always permute below $\forall$. Example 9 is essentially the same example that shows quantifier impermutability in classical logic. That is, the $\forall$ must be instantiated lower in the proof tree than the $\exists$ in sequent 9 in classical as well as linear logic. However, classical logic enjoys all other possible permutabilities, while intuitionistic and linear logic have many impermutabilities.

Any blank in the table indicates the rules in question are always permutable. A $\circ$ indicates cases permutable under the looser definition of permutability, but which require certain modifications or extensions to the proofs of some hypothesis, and thus do not always permute by the stricter definition of permutation. A $-$ indicates the impermutable cases involving the cut rule.

Note that I, 1, and $\top$ are trivial cases, since they have no hypotheses. Also, $!S$ is only rarely permutable, since it requires all other formulas to be prefixed with ?, (although here only the !'d formula is considered principal in an application of !S). Thus in all permutable cases except $?W$ and $?C$, the permutability of $!S$ is trivial. For example, there simply are no proofs with $\otimes$ immediately above $!S$ where the principal formulas are not subformula of one another.

	$\otimes$	$\parr$	$\&$	$\oplus$	$?W$	$?C$	$?D$	$!S$	$\bot$	$\forall$	$\exists$	Cut
$\otimes$								0				
$\parr$	1							0				$-$
$\&$	2	$\circ$	$\circ$	3	4	$\circ$	5	0	$\circ$	$\circ$	6	$-$
$\oplus$								0				
$?W$												
$?C$	7											$-$
$?D$												
$!S$							8					$-$
$\bot$								0				
$\forall$								0			9	$-$
$\exists$								0				
Cut								$-$				

0 various examples
1 $\vdash (A \parr B), (A^\bot \otimes B^\bot)$
2 $\vdash (A\&B), (\bot \otimes \top), A^\bot, B^\bot$
3 $\vdash (A\&B), (A^\bot \oplus B^\bot)$
4 $\vdash (1\&A), ?A^\bot$
5 $\vdash ((!A)\&A), ?A^\bot$
6 $\vdash (A(t)^\bot \& A(u)^\bot), \exists x.A(x)$
7 $\vdash ?A, (A^\bot \otimes A^\bot)$
8 $\vdash !(A^\bot \oplus B), ?A$
9 $\vdash \forall y.(A(y) \oplus B), \exists x.A(x)^\bot$

Note that the impermutabilities involving Cut (labeled $-$) are sensitive to the exact definitions used, and if one is using a study of permutabilies to design a linear logic programming system, then cut is of limitted interest.

This set of permutability results may be proved by applying Girard's sequentialization theorem from proof nets to the sequent calculus backward and forward [57], or by direct proof theoretic argument in the sequent calculus [51]. For full proofs see [51].

As specific applications of the above table of permutabilities, consider the following "invertability" property of the $\parr$ R rule, and the closest analog for &.

PROPOSITION 1 A sequent $\Gamma \vdash \Delta, (A \parr B)$ is provable in linear logic if and only if $\Gamma \vdash \Delta, A, B$ is.

The proof of this property is immediate from emptiness of the $\wp$ column in the array of permutabilities.

PROPOSITION 2 A sequent $\Gamma \vdash \Delta, (A \& B)$ is provable in linear logic if and only if both $\Gamma \vdash \Delta, A$ and $\Gamma \vdash \Delta, B$ are.

These and similar properties for $\perp$, $\forall$, and $\top$ can be used to direct proof search in a linear logic programming system [8, 7]. In searching for a proof of a sequent containing any formulas with main connective $\wp$, one need only consider those proofs ending in $\wp$. By immediately analyzing all top-level $\wp$, $\perp$, $\forall$, and $\top$ connectives, one can reduce the space of proof search and the expected time to find a proof dramatically.

These proof normalization properties are used extensively in later chapters to simplify reasoning about the soundness of certain encodings. For example, in the demonstration of Lemma 25 on page 121, the elimination of cut and the permutability of $\wp$ are used to limit the set of possible proofs that must be considered. Stronger forms of proof normalization greatly simplify such demonstrations.

2.10 Linear Logic Augmented with Theories

Essentially, a theory is a set of nonlogical axioms (sequents) that may occur (many times) as leaves of a proof tree. The main theorem of this section can be seen as the linear analog of the deduction theorm. The use of theories described here is an extension of earlier work on multiplicative theories [67, 115].

The point of the extension of linear logic to encompass nonlogical theories is to highlight the difference between linear (non-! or MALL) formulas and reusable (!, or theory axiom) formulas. Theory axioms can be used to capture the behavior of machine instructions such as the transitions of a Petri net, a Turing machine's finite control, or the transitions of a counter machine. MALL formulas appearing in a sequent can be used to represent the current state of a machine.

This section shows that nonlogical MALL theories may be efficiently encoded in full propositional linear logic using the modal !. Thus one may study the computational significance of propositional linear logic by considering MALL augmented with nonlogical theories, knowing that any result for this system can be carried over to full propositional linear logic.

2.10.1 Encoding mall Theories

We define the translation $[T]$ of a theory T with k axioms into a multiset of pure linear logic formulas by

$[\{t_1, t_2, \cdots, t_k\}] = [t_1], [t_2], \cdots, [t_k]$

where $[t_i]$ is defined for each axiom t_i as follows:

$$\begin{array}{c}[\, G_1, \cdots, G_n \vdash F_1, F_2, \cdots, F_m \,] \\ \triangleq \\ !((G_1 \otimes \cdots \otimes G_n) \multimap (F_1 \parr F_2 \parr \cdots \parr F_m))\end{array}$$

LEMMA 4 (THEORY $\Rightarrow$) For any finite set of axioms T, $\Gamma \vdash \Delta$ is provable in theory T only if $\Gamma, [T] \vdash \Delta$ is provable without nonlogical axioms.

Proof 5 Given some proof of $\Gamma \vdash \Delta$ in theory T, we have a linear logic proof tree with axioms of T at some leaves. For each leaf of the proof tree of the form $\Theta \vdash \Sigma$, where $\Theta \vdash \Sigma$ is some axiom t_i, we replace that leaf with a small proof of $\Theta, [T] \vdash \Sigma$ which does not use theory axioms (see below). For each leaf sequent which was originally an application of identity or 1 R, or $\bot L$, we extend the proof by adding weakenings for all the $![t_i]$ formulas. For each leaf sequent which consists of an application of $\top$ R, or 0 L, we simply add the $[T]$ formulas to the sequent. We then add $[T]$ to every sequent in the body of the entire proof tree. At every application of $\otimes R$, $\parr\, L$, $\multimap L$, and **Cut**, we extend the proof tree with contractions on each formula in $[T]$.

The first mentioned proof tree, of $\Theta, [T] \vdash \Sigma$, will be constructed from the proof tree for $\Theta, [t_i] \vdash \Sigma$. Since each formula in $[T]$ begins with !, we may weaken in the remainder of $[T]$, and thus with some number of weakening steps we have $\Theta, [T] \vdash \Sigma$. For example, if there are k axioms, and $\Theta \vdash \Sigma$ is the axiom $t_1 = q_1 \vdash (q_2 \otimes a)$, then we know $[t_1] = !(q_1 \multimap (q_2 \otimes a))$. We then perform the following transformation:

$$\dfrac{}{q_1 \vdash (q_2 \otimes a)}\,T \quad \Longrightarrow \quad \dfrac{\dfrac{\dfrac{\vdots}{\dfrac{\dfrac{\dfrac{\dfrac{}{q_1 \vdash q_1}\,I \quad \dfrac{\dfrac{\dfrac{}{q_2 \vdash q_2}\,I \quad \dfrac{}{a \vdash a}\,I}{q_2, a \vdash (q_2 \otimes a)}\,\otimes R}{(q_2 \otimes a) \vdash (q_2 \otimes a)}\,\otimes L}{q_1, (q_1 \multimap (q_2 \otimes a)) \vdash (q_2 \otimes a)}\,\multimap L}{q_1, [t_1] \vdash (q_2 \otimes a)}\,!D}{[t_1], [t_2], q_1 \vdash (q_2 \otimes a)}\,!W}\,!W}{[t_1], [t_2], \cdots, [t_{k-1}], q_1 \vdash (q_2 \otimes a)}\,!W}{[T], q_1 \vdash (q_2 \otimes a)}\,!W$$

For each leaf sequent which was originally an application of identity or 1 R, or $\bot L$, we weaken in all the $[t_i]$ formulas:

$$\dfrac{}{p_i \vdash p_i}{\scriptstyle I} \quad \Longrightarrow \quad \dfrac{\dfrac{\dfrac{\vdots}{[t_1],[t_2],\cdots,[t_{k-1}],p_i \vdash p_i}{\scriptstyle !W}}{[T],p_i \vdash p_i}{\scriptstyle !W}}{} \qquad \text{where the top is } \dfrac{\dfrac{\dfrac{\dfrac{}{p_i \vdash p_i}{\scriptstyle I}}{[t_1],p_i \vdash p_i}{\scriptstyle !W}}{[t_1],[t_2],p_i \vdash p_i}{\scriptstyle !W}}{\vdots}{\scriptstyle !W}$$

We then continue by adding $[T]$ to every sequent in the entire proof tree. At every application of $\otimes R$, $\parr L$, $\multimap L$, and **Cut**, we extend the proof tree with an extra copy of the conclusion sequent of the binary rule, to which we add an extra copy of $[T]$. Then we extend the proof further, adding one contraction step for each $[t_i]$ between that sequent and the original conclusion of that binary rule.

$$\dfrac{\begin{matrix}\vdots\\ \Theta \vdash A,\Sigma\end{matrix} \qquad \begin{matrix}\vdots\\ \Gamma \vdash B,\Delta\end{matrix}}{\Theta,\Gamma \vdash \Sigma,(A\otimes B),\Delta}{\scriptstyle \otimes}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\begin{matrix}\vdots\\ \Theta,[T] \vdash A,\Sigma\end{matrix} \qquad \begin{matrix}\vdots\\ \Gamma,[T] \vdash B,\Delta\end{matrix}}{\Theta,[T],\Gamma,[T] \vdash (A\otimes B),\Sigma,\Delta}{\scriptstyle \otimes}}{\vdots}{\scriptstyle !C}}{\Theta,[T],\Gamma \vdash (A\otimes B),\Sigma,\Delta}{\scriptstyle !C}$$

Thus we have given a construction which builds a proof of $\Theta,[T] \vdash \Sigma$ without any nonlogical axioms from a given proof of $\Theta \vdash \Sigma$ using axioms from T.

LEMMA 5 (THEORY $\Leftarrow$) For any finite set of axioms T, $\Gamma \vdash \Delta$ is provable in theory T if $\Gamma,[T] \vdash \Delta$ is provable without nonlogical axioms.

Proof 6 Assuming we have a proof of $\Gamma,[T] \vdash \Delta$, we immediately have a proof of $\Gamma,[T] \vdash \Delta$ in theory T, since any proof in pure linear logic is also a proof in the logic extended with axioms. We can also build proofs of $\vdash [t_i]$ in the theory T for each axiom t_i. By cutting these proofs against the given proof of $\Gamma, [T] \vdash \Delta$, we obtain a proof of $\Gamma \vdash \Delta$ in theory T.

For each axiom $t_i \equiv A_1, A_2, \cdots, A_n \vdash B_1, B_2, \cdots, B_m$, we may prove $\vdash [t_i] \equiv \vdash\, !((A_1 \otimes A_2 \otimes \cdots \otimes A_n) \multimap (B_1 \parr B_2 \parr \cdots \parr B_m))$ by several applications of $\otimes$ L, $\parr$ R, one application of $\multimap$ R, and one application of **!S**, as follows.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A_1, A_2, \cdots, A_n \vdash B_1, B_2, \cdots, B_m}T}{\vdots}\otimes L}{A_1 \otimes A_2 \otimes \cdots \otimes A_n \vdash B_1, B_2, \cdots, B_m}\otimes L}{\vdots}\parr R}{A_1 \otimes A_2 \otimes \cdots \otimes A_n \vdash B_1 \parr B_2 \parr \cdots \parr B_m}\parr R}{\vdash (A_1 \otimes A_2 \otimes \cdots \otimes A_n) \multimap (B_1 \parr B_2 \parr \cdots \parr B_m)}\multimap R}{\vdash !((A_1 \otimes A_2 \otimes \cdots \otimes A_n) \multimap (B_1 \parr B_2 \parr \cdots \parr B_m))}!S
$$

By cutting this proof against the given proof of $\Gamma, [T] \vdash \Delta$, we obtain a proof of $\Gamma, [T - \{t_i\}] \vdash \Delta$, where $T - \{t_i\}$ is the multiset difference of T and $\{t_i\}$. Thus by induction on the number of axioms, cutting each one out in turn, we can derive $\Gamma \vdash \Delta$ in theory T.

Immediately from the preceding two lemmas, we achieve the soundness and completeness of the translation.

THEOREM 3 (DEDUCTION) For any finite set of axioms T, $\Gamma \vdash \Delta$ is provable in theory T if and only if $\Gamma, [T] \vdash \Delta$ is provable without nonlogical axioms.

Despite the fact that one can encode arbitrary theories in linear logic, as demonstrated above, the remainder of this section will focus on a specific subclass of propositional, nonmodal theories. The theories we will define below are sufficient to encode recursively unsolvable problems, but yet allow a great deal of control over the shape of proofs.

2.10.2 Cut Standardization

We now focus on a special class of linear theories, where every theory axiom contains at most one formula at top level which is not a negative literal. Recall that a positive literal is one of the given propositional symbols p_i or $p_i^\perp$ which occurs with positive polarity, and a negative literal is one of these which occurs with negative polarity. Thus the class of theories of interest here contains linear logic sequents of the form $q_1, q_2, \cdots q_n \vdash C, p_1^\perp, p_2^\perp, \cdots, p_n^\perp$ or of the form $q_1, q_2, \cdots q_n, C \vdash p_1^\perp, p_2^\perp, \cdots, p_n^\perp$. For example, the sequents $q, r \vdash s, p^{\perp}$, $q, p \vdash (p \otimes q), p^{\perp}$, $(p \otimes p), q \vdash p^{\perp}$, and $\vdash p^{\perp}, q^{\perp}$ are all axioms in this class. However, $\vdash p_1, p_1$ and $(p_1 \otimes p_2), p_3^{\perp} \vdash$ are not.

A *directed* cut is one where at least one premise is an axiom in T, and the cut-formula in the axiom is not an atomic literal of negative polarity. We call any axiom premise of a directed cut where the cut-formula in that axiom is not a negative literal a *principal axiom* of that directed cut. By definition, all directed cuts have at least one principal axiom. A cut between two axioms is always directed, and if the cut-formula of such a cut is non-atomic, that cut has two principal axioms. A *directed* or *standardized* proof is a proof with all cuts directed.

When theories are added to linear logic the cut-elimination Theorem 2 no longer holds, due to the added axioms which may participate directly in cuts. Such cuts may never be removed: consider the nonlogical MALL theory consisting of two axioms: $p_1 \vdash p_2$ and $p_2 \vdash p_3$:

$$\dfrac{\dfrac{}{p_1 \vdash p_2}T \qquad \dfrac{}{p_2 \vdash p_3}T}{p_1 \vdash p_3}Cut$$

By case analysis, it may be seen that there is no proof of this conclusion sequent that doesn't contain cut. However, we do obtain the following result:

LEMMA 6 (CUT STANDARDIZATION) If there is a proof of $\Theta \vdash \Delta$ in theory T, then there is a directed proof of $\Theta \vdash \Delta$ in theory T.

Proof 7 We modify the cut-elimination procedure defined in Section 2.6 in two ways. First we alter the definition of degree to ignore the measure of directed cuts. Formally, we say that if a cut is directed, its degree is zero. Second, we modify the procedure given in Lemma 1 to handle the extra cases brought about by the presence of axioms. We must allow all the reductions as stated in Sectoin 2.6 to apply to the case when one of the premises is an axiom, but we need not introduce any truly novel reductions.

We will follow the notation used in Section 2.6, where **Cut*** is used to ambiguously refer to the **Cut** rule or the extra rule of inference called **Cut!**. Also, we will define all the formulas which appear in an axiom to be principal in that application of the axiom.

In Section 2.6 most of the reductions are given for some specific derivation versus any possible derivation. For example, all the non-principal cases are stated for any derivation of the "other" hypothesis of **Cut***. Similarly, the Identity and $\top$ R rules are stated for any derivation of the "other" hypothesis. We simply now state that even if the other derivation involves an axiom, the reduction still applies.

For example, if the last rule applied in the left hypothesis is $\otimes$, and the last rule in the right hypothesis is an axiom, we apply the following transformation:

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \Theta_1 \vdash \Sigma, C\end{matrix} \qquad \begin{matrix}\vdots\\ \Theta_2 \vdash B, \Delta, p_i\end{matrix}}{\Theta_1, \Theta_2 \vdash \Sigma, (C \otimes B), \Delta, p_i}\otimes \qquad \dfrac{}{\Theta_3 \vdash \Gamma, p_i^{\perp}}T}{\Theta_1, \Theta_2, \Theta_3 \vdash \Sigma, (C \otimes B), \Delta, \Gamma}Cut$$

$$\Downarrow$$

$$\dfrac{\begin{matrix}\vdots\\ \Theta_1 \vdash \Sigma, C\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\\ \Theta_2 \vdash B, \Delta, p_i\end{matrix} \qquad \dfrac{}{\Theta_3 \vdash \Gamma, p_i^{\perp}}T}{\Theta_2, \Theta_3 \vdash B, \Delta, \Gamma}Cut}{\Theta_1, \Theta_2, \Theta_3 \vdash \Sigma, (C \otimes B), \Delta, \Gamma}\otimes$$

This is simply a special case of the reduction given in case 2.6.1 in Section 2.6.

Also, as a second example, the reduction given for Identity is applicable even to the axiom case:

$$\dfrac{\dfrac{}{p_i \vdash p_i}I \qquad \dfrac{}{\Theta, p_i \vdash \Gamma}T}{\Theta, p_i \vdash \Gamma}Cut \quad \Longrightarrow \quad \dfrac{}{\Theta, p_i \vdash \Gamma}T$$

Again, this is simply a special case of the reduction given in Section 2.6.

As a third and final example of specializations of reductions, the $\top$ R rule also applies to axioms:

$$\dfrac{\dfrac{}{\Theta_1 \vdash \top, \Sigma, p_i}\top \qquad \dfrac{}{\Theta_2, p_i \vdash \Gamma}T}{\Theta_1, \Theta_2 \vdash \top, \Sigma, \Gamma}Cut \quad \Longrightarrow \quad \dfrac{}{\Theta_1, \Theta_2 \vdash \top, \Sigma, \Gamma}\top$$

This is also a special case of the reduction given earlier.

Now, some simple analysis is required to show that there are no new cases of principal cuts involving axioms. If the cut in question is already directed, the cut has degree zero, by our modified definition, and thus we are done. Otherwise, by definition of axiom we know that the cut-formula is a negative atomic literal. There are only three rules where an atomic literal may be principal: Identity, 0 L, and $\top$ **R**. However, all of these cases are handled by existent reductions (two of which are restated above). One should also note that since any cut involving two axioms must be directed, we needn't provide a reduction for that case.

This completes the discussion of the modifications to Lemma 1 necessary to handle nonlogical axioms. Fortunately, Lemma 2 and Theorem 2 then follow without modification (although the definition of degree has changed somewhat).

Therefore, given any proof of a sequent $\Theta \vdash \Gamma$ in theory T, we can construct a directed proof of $\Theta \vdash \Gamma$ in theory T.

The cut-elimination procedure in Section 2.6 introduces new rules of inference called **Cut!** and **Cut?**. If we generalized axioms to allow ? and ! formulas in axioms, we would have to generalize the notion of directed proof to include cases involving **Cut!** and **Cut?**, and a post processing step would be required to transform all directed **Cut!** and **Cut?**s into a sequence of contractions followed by a single directed **Cut**, or perhaps simply into a sequence of **Cut**s. In any event, our axioms are restricted to MALL formulas so that any cut involving an axiom is always an application of **Cut**, never of **Cut!** nor of **Cut?**.

2.11 Ponderables

1. Using $((A \multimap B) \multimap 0) \multimap (A \otimes \top)$ rigorously prove that CLL is not conservative over ILL.
2. Why is $(A \otimes B) \parr (A \otimes B^{\perp}) \parr (A^{\perp} \otimes B) \parr (A^{\perp} \otimes B^{\perp})$ not provable in CLL, even though all resources add up?
3. Give an example where cut fails to permute with $\forall$. That is, give a proof that ends in the use of $\forall$ with the use of cut ending the subproof of one hypothesis that cannot be permuted.
4. Rigorously prove the (loose) permutability of $\parr$ with &.
5. Linear logic is not the only logic with several impermutabilities. Give the complete set of impermutabilities of LJ with an example of each. (see [90]).

2.12 Summary of Chapter

In this chapter, we have presented several proof theoretic results. These results are relatively minor extensions of theorems already in the literature of linear logic. However, the exact form of some of these theorems, such as cut-elimination, are used to develop richer proof theoretic tools, such as cut standardization (Lemma 6) in the presence of theory axioms. These results will be used as the basis for later theorems, such as the undecidability of full propositional linear logic.

3 Propositional Linear Logic Is Undecidable

> Decided only to be undecided, resolved to be irresoute, adament for drift, solid for fluidity, all powerful to be impotent.
> SIR WINSTON SPENCER CHURCHILL 1874-1965
>
> *While England Slept* 1936

In this chapter we show that unlike most propositional logics, the decision problem for propositional linear logic is not recursively solvable. We study this problem through the use of nonlogical axioms in the fragment of linear logic without modal operators (MALL). Since this class of axioms may be encoded in full propositional linear logic, as shown in Chapter 2, undecidability of MALL with nonlogical axioms implies the undecidability of full propositional linear logic. One can also prove this result directly, without the use of nonlogical theories. The undecidability result was first obtained by the author in such a direct fashion, and recently Kanovich has simplified this approach [83]. Such direct encodings may ease certain semantic embeddings [96, 97], but the proof using nonlogical axioms factors out many logical concerns from the computational machinery.

The proof of undecidability of MALL with nonlogical axioms consists of a reduction from the halting problem for a form of counter machine to a decision problem in linear logic. In more detail, we begin by extending propositional linear logic with theories whose (nonlogical) axioms may be used any number of times in a proof. We then describe a form of *and*-branching two-counter machine with an undecidable halting problem and show how to encode these machines in propositional linear logic with theories. Since the axioms of our theories must have a special form, we are able to show the faithfulness of this encoding using a natural form of cut-elimination in the presence of nonlogical axioms. To illustrate the encoding of two-counter machines, we present an example simulation of a simple computation in Section 3.3. On first reading, the reader may wish to jump ahead to that section since it demonstrates the basic mechanism used in the undecidability proof.

It should be noted that linear logic is not the only known undecidable propositional logic — propositional relevance logic is also undecidable [146]. However, the undecidability of these logics appear to arise in very different ways. Linear logic fails to have a critical distributivity property used in the proof of undecidability of relevance logic. Without this distributivity relevance logic is decidable. Further, relevance logic fails to exhibit the two different kinds of conjunction ($\otimes$ and $\&$) used in the proof of undecidability of linear logic.

The encoding of an undecidable problem in linear logic hinges on the combination of three powerful mechanisms: resource accumulation, arbitrary reuse, and and-branching. In linear logic, A, A is very different from A, and this allows us to represent counters in unary. Indefinitely reusable formulas such as $!(B \multimap C)$, (or axioms of the form $B \vdash C$) may be used to implement machine instructions. The additive connectives & and $\oplus$ may be used to test a conjunction of properties of the simulated machine, such as whether a counter is zero, and if the rest of the computation can continue normally. Together this machinery is enough to encode recursively unsolvable problems in linear sequents.

3.1 And-Branching Two Counter Machines

In this section we describe nondeterministic two counter machines with *and*-branching but without a zero-test instruction. We show that these machines have a recursively unsolvable halting problem, and then we will show how the halting problem for these machines may be encoded as a decision problem in MALL, with nonlogical axioms corresponding to the machine instructions.

The machines described here are similar to standard two counter machines except for the lack of an explicit zero test transition, and the addition of "fork" transitions. Intuitively, Q_i *Fork* Q_j, Q_k is an instruction which allows a machine in state Q_i to continue computation from both states Q_j and Q_k, each computation continuing with the current counter values. For brevity in the following proofs, we emphasize *two* counter machines. However, there is no intrinsic reason to restrict the machines to two counters. All of our arguments and results generalize easily to N counters, for $N \geq 2$. Formally, an *And-Branching Two Counter Machine Without Zero-Test*, or ACM for short, is given by a finite set Q of states, a finite set δ of transitions, and distinguished initial and final states, Q_I and Q_F, as described below.

An *instantaneous description*, or *ID*, of an ACM M is a finite list of ordered triples $\langle Q_i, A, B\rangle$, where $Q_i \in Q$, and A and B are natural numbers, each corresponding to a *counter* of the machine. Intuitively, a list of triples represents a set of machine configurations. One may think of an ACM state as some sort of parallel computation which terminates successfully only if all its concurrent computation fragments terminate successfully.

We define the *accepting* triple as $\langle Q_F, 0, 0\rangle$. We also define an *accepting* ID as any ID where every element of the ID is the accepting triple. That is, every and-branch of the computation has reached an accepting triple. We say that an ACM M *accepts from* an ID s if and only if there is some computation from s to an accepting ID. It is essential for our encoding in linear logic that both counters be zero in all elements of an accepting ID.

The set δ may contain transitions of the following form:

$(Q_i$ **Increment** A $Q_j)$ taking
$\langle Q_i, A, B\rangle$ to $\langle Q_j, A+1, B\rangle$
$(Q_i$ **Increment** B $Q_j)$ taking
$\langle Q_i, A, B\rangle$ to $\langle Q_j, A, B+1\rangle$
$(Q_i$ **Decrement** A $Q_j)$ taking
$\langle Q_i, A+1, B\rangle$ to $\langle Q_j, A, B\rangle$
$(Q_i$ **Decrement** B $Q_j)$ taking
$\langle Q_i, A, B+1\rangle$ to $\langle Q_j, A, B\rangle$
$(Q_i$ **Fork** $Q_j, Q_k)$ taking
$\langle Q_i, A, B\rangle$ to $(\langle Q_j, A, B\rangle, \quad \langle Q_k, A, B\rangle)$

where Q_i, Q_j, and Q_k are states in Q. The **Decrement** instructions only apply if the appropriate counter is not zero, while the **Increment** and **Fork** instructions are always enabled from the proper state.

For example, the single transition Q_i **Increment** A Q_j takes an ACM from ID: $\{\cdots, \langle Q_i, A, B\rangle, \cdots\}$ to ID: $\{\cdots, \langle Q_j, A+1, B\rangle, \cdots\}$

3.1.1 Standard Two Counter Machines

Standard two counter machines have a finite set of states, Q, a finite set of transitions, δ, a distinguished initial state Q_I, and a set of final states F [120, 74]. An instantaneous description of the state of a two counter machine is given by a triple $\langle Q_i, A, B\rangle$, which consists of the current state, and the values of two counters, A and B. The transitions in δ are of four kinds:

$(Q_i$ **Increment** A $Q_j)$	taking	$\langle Q_i, A, B\rangle$ to $\langle Q_j, A+1, B\rangle$
$(Q_i$ **Increment** B $Q_j)$	taking	$\langle Q_i, A, B\rangle$ to $\langle Q_j, A, B+1\rangle$
$(Q_i$ **Decrement** A $Q_j)$	taking	$\langle Q_i, A+1, B\rangle$ to $\langle Q_j, A+1, B\rangle$
$(Q_i$ **Decrement** B $Q_j)$	taking	$\langle Q_i, A, B+1\rangle$ to $\langle Q_j, A, B+1\rangle$
$(Q_i$ **Zero-Test** A $Q_j)$	taking	$\langle Q_i, 0, B\rangle$ to $\langle Q_j, 0, B\rangle$
$(Q_i$ **Zero-Test** B $Q_j)$	taking	$\langle Q_i, A, 0\rangle$ to $\langle Q_j, A, 0\rangle$

A two counter machine accepts if it is able to reach any one of the final states in the set F with both counters at zero. It is important that these machines have a **Zero-Test** instruction since the halting problem becomes decidable otherwise, by obvious reduction to the word problem in commutative semi-Thue systems, which is decidable [116]. Since **Zero-Test** is the most difficult to encode in linear logic, we concentrate on a machine with and-branching, which provides a basic mechanism powerful enough to simulate **Zero-Test**, but which is more easily simulated in linear logic.

Using two counter machines, we show that ACM's have an undecidable halting problem.

LEMMA 7 It is undecidable whether an and-branching two counter machine without zero-test accepts from ID $\{\langle Q_I, 0, 0\rangle\}$. This remains so if the transition relation of the machine is restricted so that there are no outgoing transitions from the final state.

Proof 8 Since ACM's may simulate zero-test with and-branching, ACM's are sufficiently powerful to simulate two counter machines, for which the halting problem is known to be recursively unsolvable [120, 100]. We will give a construction from standard two counter machines to ACMs, and argue that the construction is sound and faithful. This construction and the proof of its soundness is routine, and its steps should be familiar to anyone versed in automata theory. In our simulation of the test for zero instruction of two counter machines, we make essential use of the fact that all branches of computation terminate with both counters set to zero.

Given a nondeterministic two counter machine M we first construct an equivalent two counter machine M' with a unique final state Q_F which has no outgoing transitions. One simply adds two new states, Q_D and Q_F to M', and for each $Q_f \in F$ of M, one adds the instructions (Q_f **Increment** A Q_D) and (Q_D **Decrement** A Q_F). Note that one may simply look at these new transitions as a single nondeterministic step from each old final state to the new (unique) final state, which has no outgoing transitions. However, since there is no general "silent" move, we make the transition in two steps.

We claim without proof that M and M' accept the same set of input values, and are therefore equivalent machines.

From a nondeterministic two counter machine M' with unique final state without outgoing transitions, we construct an ACM M'' as follows. The ACM M'' will have the same set of states, and same initial and final states as M'. The transition function of M'' is built by first taking all the **Increment** and **Decrement** instructions from the transition function of M'. We then add two new states to M'', Z_A and Z_B, which are used to test for zero in each of the two counters. For Z_A, we add two instructions, (Z_A **Decrement** B Z_A), and (Z_A **Fork** Q_F, Q_F), to the transition function of M''. Similarly for Z_B, we add (Z_B **Decrement** A Z_B), and (Z_B **Fork** Q_F, Q_F). Then for each **Zero-Test** instruction of M' of the form

(Q_i **Zero-Test** A Q_j)

we add one instruction to M'':

$(Q_i$ **Fork** $Q_j,\ Z_A)$.

An important feature of M'' is that once a zero testing or final state is entered, no configuration of that branch of computation may ever leave that set of states. More specifically, where M' would test for zero, M'' will fork into two "parallel" computations. One continues in the "same" state as M' would have if the **Zero-Test** had succeeded, and the other branch "verifies" that the counter is indeed zero. While the second branch may change the value of one of the counters (the counter which is not being tested), this cannot affect the values of the counters in the "main" branch of computation. Further, the zero-testing branch of computation never enters any states other than zero-test states or the final state. This holds because there are no outgoing transitions from the final state whatsoever, and the only transitions from the two zero testing states either loop back to that state or move directly to the final state. Also note that any branch of an ACM M'' computation which arrives at the state Z_A may be part of a terminating computation if and only if the counter A is zero when the machine reaches that state. This can be seen by observing that once arriving in Z_A, there is no possibility of modifications to the counter A. The **Decrement** B transition from Z_A to itself allows M'' to loop and decrement the counter B arbitrarily. In particular it is possible for B to be decremented to the value 0. Since Q_F has no outgoing transitions, the **Fork** instruction which moves from Z_A to Q_F and Q_F allows this branch of computation to terminate correctly if and only if both counters are zero when it is executed. Since we are considering nondeterministic ACMs, it is possible for a branch of computation which reaches Z_A to terminate if and only if the A counter is zero when it reaches Z_A. Similarly, any branch of computation reaching Z_B reaches an accepting ID if and only if the B counter is zero.

We claim that there is a halting computation for the given two counter machine M' if and only if there is one for the constructed ACM M''. This is proven by two simulations.

The and-branching machine M'' may mimic the original two counter machine in the performance of any instruction, by following any **Increment** of M' with the corresponding **Increment** instruction, and a **Decrement** with the corresponding **Decrement**. When M' executes a **Zero-Test** A instruction, M'' forks off an *and* branch which verifies that the counter A is in fact zero, and the other branch continues to follow the computation of M'.

For the converse simulation, there is always at most one and-branch of any M'' computation which corresponds to a nonfinal, non-zero-testing state in the original machine. There may be many *and* branches of the computation which are in states Z_A, Z_B, and Q_F, but at most one *and* branch is in any other state. Thus, M' may mimic M'' by following the branch of ACM computation which does not enter Z_A, Z_B, or Q_F until the final step of computation, when it enters Q_F. For every **Increment** and **Decrement** instruction in the accepting computation of M'', M' may performs the corresponding instruction. Every **Fork** instruction executed by M'' from a nonfinal, non-zero-testing state corresponds to a **Zero-Test** instruction in M', and by the above observation, if M'' forks into state Z_A, then M'' accepts only if the counter A is zero (and similarly for Z_B and the counter B). Since we are assuming an accepting M'' computation, we know that M' may execute the corresponding **Zero-Test** instruction successfully.

3.2 From Machines to Logic

We give a translation from ACMs to linear logic with theories and show that our sequent translation of a machine in a particular state is provable in linear logic if and only if the ACM halts from that state. In fact, our translation uses only MALL formulas and theories, thus with the use of the linear deduction theorem, Theorem 3, we will have our result for propositional linear logic without nonlogical axioms. Since an instantaneous description of an ACM is given by a list of triples, it is somewhat delicate to state the induction we will use to prove soundness.

The main idea of this encoding is to use the linear connective & to simulate and-branching behavior, along with previously described techniques for encoding petri-net like tokens using the multiplicative connectives. These machine parts may be combined to build an ACM.

Given an ACM $M = \langle Q, \delta, Q_I, Q_F \rangle$ we first define a set of propositions:

$$\{q_i | Q_i \in Q\} \bigcup \{a, , b\}$$

We then define the linear logic theory corresponding to the transition relation δ as the set of axioms determined as follows:

$$\begin{array}{rcl}
Q_i \ \textbf{Increment} \ \mathrm{A} \ Q_j & \mapsto & q_i \vdash (q_j \otimes a) \\
Q_i \ \textbf{Increment} \ \mathrm{B} \ Q_j & \mapsto & q_i \vdash (q_j \otimes b) \\
Q_i \ \textbf{Decrement} \ \mathrm{A} \ Q_j & \mapsto & q_i, a \vdash q_j \\
Q_i \ \textbf{Decrement} \ \mathrm{B} \ Q_j & \mapsto & q_i, b \vdash q_j \\
Q_i \ \textbf{Fork} \ Q_j, Q_k & \mapsto & q_i \vdash (q_j \oplus q_k)
\end{array}$$

Given a triple $\langle Q_i, x, y \rangle$ of an ACM, we define the translation $\theta(\langle Q_i, x, y \rangle)$ by:

$\theta(\langle Q_i, x, y\rangle) \triangleq q_i, a^x, b^y, \vdash q_F$

The notation a^x (and similarly b^y) is used here to indicate a sequence of x a's, separated by commas, as follows.

$a^x \triangleq \overbrace{a, a, \cdots, a}^{x}$

Thus all sequents which correspond to ACM triples have exactly one positive literal, q_F, some number of as, and bs, the multiplicity of which correspond to the values of the two counters of the ACM in unary, and exactly one other negative literal, q_i, which corresponds to the state of the ACM.

The translation of an ACM ID is simply the set of translations of the elements of the ID:

$\theta(\{E_1, E_2, \cdots, E_m\}) = \{\theta(E_1), \theta(E_2), \cdots, \theta(E_m)\}$

We claim that an ACM M accepts from ID s if and only if every element of $\theta(s)$ is provable in the theory corresponding to the transition function of the machine. We prove each half of this equivalence in separate lemmas.

LEMMA 8 (MACHINE $\Rightarrow$) An and-branching counter machine M accepts from ID s only if every sequent in $\theta(s)$ is provable in the theory derived from M.

Proof 9 Given a halting computation of an ACM machine M from s we claim we can build a proof of every sequent in $\theta(s)$ in the theory derived from M.

M accepts from s only if there is some finite sequence of transitions from this ID to an accepting ID. We proceed by induction on the length of that sequence of transitions.

If there are no transitions in the sequence, then by the definition of accepting ID, s consists entirely of $\langle Q_F, 0, 0\rangle$. We must show that the sequent

$q_F, a^0, b^0, \vdash q_F$

is provable in linear logic. This is immediate: we have 0 A's and 0 B's, that is, none at all. Thus by one application of identity (per sequent) $q_F \vdash q_F$, we have our (set of) proof.

If there is at least one transition in the sequence, we have to show that $\theta(s)$ is provable. Since M accepts from ID $\{\cdots\langle Q_i, A, B\rangle\cdots\}$, and there is at least one transition in the sequence, we know that there is some transition in M such that

$ID \to ID'$, and M accepts from ID'. We assume by induction that there is a linear logic proof which corresponds to the accepting computation for ID'.

We now perform case analysis on the type of transition. There are five different types of instructions: **Increment** A or B, **Decrement** A or B, and **Fork**. Since the two increment and two decrement instructions are nearly identical, we will concentrate only on the cases concerning the counter A.

Q_i **Increment A** Q_j**:** In this case, the first step in the halting computation has the form

$$\{\cdots\langle Q_i, A, B\rangle\cdots\} \to \{\cdots\langle Q_j, A+1, B\rangle\cdots\}$$

We assume by induction that we have a proof of $\theta(\langle Q_j, A+1, B\rangle) = q_j, a^{A+1}, b^B \vdash q_F$. We extend this proof into a proof of $\theta(\langle Q_i, A, B\rangle) = q_i, a^A, b^B \vdash q_F$ by adding a cut with an axiom, as follows.

$$\dfrac{\dfrac{}{q_i \vdash (q_j \otimes a)}T \qquad \dfrac{\begin{array}{c}\vdots\\ q_j, a^{A+1}, b^B \vdash q_F\end{array}}{(q_j \otimes a), a^A, b^B \vdash q_F}\otimes L}{q_i, a^A, b^B \vdash q_F}Cut$$

Note that the axiom $q_i \vdash (q_j \otimes a)$ is precisely the translation of the transition taken by the machine, and therefore is an axiom of the theory.

Q_i **Increment B** Q_j**:** Analogous to above.

Q_i **Decrement A** Q_j**:** Since the A counter of the machine must be positive for this instruction to apply, we know that the halting computation begins with the transition

$$\{\cdots\langle Q_i, A+1, B\rangle\cdots\} \to \{\cdots\langle Q_j, A, B\rangle\cdots\}$$

We assume by induction that we have a proof of $q_j, a^A, b^B \vdash q_F$. As in the **Increment** A case, we extend this to a proof of $q_i, a^{A+1}, b^B \vdash q_F$ by adding a cut with the axiom corresponding to the transition taken by the machine.

$$\dfrac{\dfrac{}{q_i, a \vdash q_j}T \qquad \begin{array}{c}\vdots\\ q_j, a^A, b^B \vdash q_F\end{array}}{q_i, a^{A+1}, b^B \vdash q_F}Cut$$

Q_i **Decrement B** Q_j**:** Analogous to above.

Q_i **Fork** Q_j, Q_k**:** Here, the halting computation begins with the step

$\{\cdots \langle Q_i, A, B\rangle \cdots\} \rightarrow \{\cdots \langle Q_j, A, B\rangle, \langle Q_k, A, B\rangle \cdots\}$

We assume by induction that we have a proof of $q_j, a^A, b^B \vdash q_F$,
and of $q_k, a^A, b^B \vdash q_F$, and we extend those proofs into a proof of $q_i, a^A, b^B \vdash q_F$.

$$\dfrac{\dfrac{}{q_i \vdash (q_j \oplus q_k)}T \qquad \dfrac{\begin{matrix}\vdots \\ q_j, a^A, b^B \vdash q_F\end{matrix} \qquad \begin{matrix}\vdots \\ q_k, a^A, b^B \vdash q_F\end{matrix}}{(q_j \oplus q_k), a^A, b^B \vdash q_F}\oplus L}{q_i, a^A, b^B \vdash q_F}Cut$$

Here $q_i \vdash (q_j \oplus q_k)$ is the axiom which corresponds to the fork instruction.

LEMMA 9 (MACHINE $\Leftarrow$) An and-branching counter machine M accepts from ID s if every sequent in the set $\theta(s)$ is provable in the theory derived from M.

Proof 10 There are two main approaches to the proof of this lemma. The approach taken here is a proof-theoretic one, yielding a direct reading of a normalized proof as a computation. The second approach explored in Section 3.4 is to use semantics of linear logic formulas to circumvent the need for normalization.

Given a set of proofs of the elements of $\theta(s)$ in the theory derived from M, we claim that a halting computation of the ACM M from state s can be extracted from those proofs. We achieve this with the aid of the cut standardization Lemma 6, which in this case leaves cuts in the proof only where they correspond to applications of ACM instructions. We may thus simply read the description of the computation from the standardized proof.

By Lemma 6, it suffices to consider standardized proofs. We show that a set of standardized proofs of $\theta(s)$ may be mimicked by the ACM M to produce an accepting computation from state s.

This proof is by induction on the sum of the sizes (number of proof rules applied) of standardized proofs. Since an ACM state is given by a finite set of triples, and all proofs are finite, we know that this measure is well founded. We assume that any smaller set of proofs which all end in conclusions which correspond to a triple $\langle Q_i, A, B\rangle$ can be simulated by machine M.

We consider the proof of a single element of $\theta(s)$ at a time.

If $s = \{\cdots \langle Q_i, x, y\rangle \cdots\}$, then $\theta(s) \;=\; \{\cdots q_i, a^x, b^y \vdash q_F \;\cdots\}$.

We assume that we are given a proof of each element of the set $\theta(s)$, and we analyze one of the proofs, all of which end in a conclusion corresponding to a machine triple $\langle Q_i, x, y\rangle$.

$$\begin{array}{c}\vdots\\ q_i, a^x, b^y \vdash q_F\end{array}$$

Since this sequent is simply a list of atomic propositions, the only linear logic rules which can apply to any such sequent are identity, some axiom, and cut.

Identity is only applicable when both x and y are zero, and $q_i = q_F$. In this case, $q_F \vdash q_F$ already corresponds to the accepting triple $\langle Q_F, 0, 0\rangle$.

The only axioms which are identical to a sequent in $\theta(s)$ are those which correspond to some δ which is a decrement instruction that ends in q_F. In this case, since each decrement axiom in $[\delta]$ contains exactly one occurrence of a or b, $x = 1$ and $y = 0$, or $x = 0$ and $y = 1$. In either case, the ACM machine M need only perform the decrement instruction δ, and this branch of computation reaches an accepting triple.

The final possibility is cut, and by our standardization procedure, we know that one hypothesis of that cut is an axiom from the theory derived from M, and furthermore that the cut-formula in that axiom is not a negative literal.

Since there are only five types of instructions in an ACM; **Increment** A or B, **Decrement** A or B, and **Fork**, there are only five different types of axioms in a theory derived from any ACM M. We now perform case analysis on the type of axiom that was last applied in a proof.

$q_i \vdash (q_j \otimes a)$**:** If the last axiom applied is of the form $q_i \vdash (q_j \otimes a)$, then it corresponds to an **Increment** A instruction, and by standardization, we know the cut-formula must be $(q_j \otimes a)$ in the axiom, and that the proof must look like

$$\dfrac{\dfrac{}{q_i \vdash (q_j \otimes a)}T \qquad \begin{array}{c}\vdots\\ (q_j \otimes a), a^x, b^y \vdash q_F\end{array}}{q_i, a^x, b^y \vdash q_F}Cut$$

Since each other linear logic rule besides $\otimes$ L, cut, identity, or axiom introduces some symbol which does not occur in $(q_j \otimes a), a^x, b^y \vdash q_F$, the derivation of this sequent must end in one of these rules. Furthermore, there are two formulas in this sequent which are not negative literals, so this sequent is not derivable using only an axiom. Identity could not lead to this sequent, since the sequent contains a non-atomic formula. By our standardization procedure, we know that each cut must involve an axiom from the theory, and the cut-formula in the axiom is not a negative literal. Inspecting the various types of axioms in the theory derived from M, we see that all axioms contain one top level negative atomic formula q_i for some i. Since q_i cannot be a directed cut-formula in a principal axiom, it must appear in the conclusion of that application of cut. However, there is no such top level q_i in the sequent in question. Thus this sequent may only be derived by the application of the $\otimes$ L rule. Therefore, we know the derivation must have the form:

$$\dfrac{\dfrac{}{q_i \vdash (q_j \otimes a)}T \qquad \dfrac{\begin{array}{c}\vdots\\ q_j, a^{x+1}, b^y \vdash q_F\end{array}}{(q_j \otimes a), a^x, b^y \vdash q_F}\otimes L}{q_i, a^x, b^y \vdash q_F}Cut$$

We know that the proof of $q_j, a^{x+1}, b^y \vdash q_F$ may be simulated by the ACM by induction, since it is the sequent $\theta(\langle Q_j, x+1, y\rangle)$, which corresponds to the triple $\langle Q_j, x+1, y\rangle$, and has a proof in linear logic of smaller size.

Therefore the machine M may emulate this proof by performing the ACM instruction corresponding to the axiom used (in this case an **Increment** A instruction), and then continuing as dictated by the inductive case.

$q_i \vdash (q_j \otimes a)$**:** Analogous arguments apply.

$q_i, a \vdash q_j$**:** If the last axiom applied is $q_i, a \vdash q_j$, which corresponds to a **Decrement** A instruction, then by standardization, we know the cut-formula must be q_j in the axiom, and that the proof must be of the form

$$\dfrac{\dfrac{}{q_i, a \vdash q_j}T \qquad \begin{array}{c}\vdots\\ q_j, a^x, b^y \vdash q_F\end{array}}{q_i, a^{x+1}, b^y \vdash q_F}Cut$$

By induction, the proof of $q_j, a^x, b^y \vdash q_F$ can be simulated, since it is the sequent $\theta(\langle Q_j, x, y\rangle)$, which corresponds to the triple $\langle Q_j, x, y\rangle$, and has a shorter proof in linear logic.

Therefore the machine M may emulate this proof by performing the ACM instruction corresponding to the axiom used (in this case a **Decrement** A instruction), and then continuing as dictated by the inductive case.

$q_i \vdash a, q_j$**:** Analogous arguments apply.

$q_i \vdash (q_j \oplus q_k)$**:** If the last axiom applied is $q_i \vdash (q_j \oplus q_k)$, which corresponds to a **Fork** instruction, then by standardization, we know the cut-formula must be $(q_j \oplus q_k)$ in the axiom, and that the proof must look like

$$\dfrac{\dfrac{}{q_i \vdash (q_j \oplus q_k)}T \qquad \dfrac{\vdots}{(q_j \oplus q_k), a^x, b^y \vdash q_F}}{q_i, a^x, b^y \vdash q_F}Cut$$

Since each other linear logic rule besides $\oplus$ L, cut, identity, or axiom introduces some symbol which does not occur in $(q_j \oplus q_k), a^x, b^y \vdash q_F$, the derivation of this sequent must end in one of these rules. Furthermore, there are two formulas in the sequent which are not negative literals, so this sequent is not derivable using only an axiom. Identity could not lead to this sequent, since the sequent contains a non-atomic formula. By our standardization procedure, we know that each cut must involve an axiom from the theory, and the cut-formula in the axiom is not a negative literal. Inspecting the various types of axioms in the theory derived from M, we see that all axioms contain one top level negative atomic formula q_i for some i. Since q_i cannot be the cut-formula in a principal axiom of a directed cut, it must appear in the conclusion of that application of cut. However, there is no such top level q_i in the sequent in question. Thus this sequent may only be derived by the application of the $\oplus$ L rule. Thus we know the derivation to be of the form:

$$\dfrac{\dfrac{}{q_i \vdash (q_j \oplus q_k)}T \qquad \dfrac{\dfrac{\vdots}{q_j, a^x, b^y \vdash q_F} \qquad \dfrac{\vdots}{q_k, a^x, b^y \vdash q_F}}{(q_j \oplus q_k), a^x, b^y \vdash q_F}\oplus L}{q_i, a^x, b^y \vdash q_F}Cut$$

The proofs of $q_j, a^x, b^y \vdash q_F$ and $q_k, a^x, b^y \vdash q_F$ can be simulated on the machine by induction, since one is a sequent which corresponds to the triple $\langle Q_j, x, y\rangle$, the other corresponds to $\langle Q_k, x, y\rangle$, and each has a proof in linear logic of smaller size.

Therefore the machine M may emulate this proof by performing the ACM instruction corresponding to the axiom used (in this case a **Fork** instruction), and then continuing as dictated by the two inductive cases.

From Theorem 3 and Lemmas 7, 8, and 9 of this section, we easily obtain our main result:

THEOREM 4 The provability problem for propositional linear logic is recursively unsolvable.

As mentioned earlier, linear logic, like classical logic, has an intuitionistic fragment. Briefly, the intuitionistic fragment is restricted so that there is only one positive formula in any sequent. In fact, the entire construction above was carried out in intuitionistic linear logic, and thus the undecidability result also holds for this logic.

In any theory derived from an ACM M, there is only one positive formula in any theory axiom. Also, throughout a directed proof of $\theta(s)$ in such a theory, the only positive atom which appears outside a theory axiom is q_F. Thus any directed proof of $\theta(s)$ in a theory derived from M is in the intuitionistic fragment of linear logic, and along with a conservativity result not proven here, we have the following:

COROLLARY 3.2.1 The provability problem for propositional intuitionistic linear logic is recursively unsolvable.

In the proof of this corollary we make use of the conservativity property of full linear logic over the intuitionistic fragment for any sequents occurring in a directed proof of a translation of an ACM machine configuration. This conservativity is a weaker property than full conservativity since sequents in such a directed proof have a special form. In particular, they have no constants, and the right hand side is always a single formula.

3.3 Example Computation

This section is intended to give an overview of the mechanisms we have defined above, and lend some insight into our undecidability result, stated above. We present a simple computation of an ordinary two counter machine *with* zero-test instruction, a corresponding ACM computation, and a corresponding linear logic proof.

Repeating from the introduction, a key insight is that searching for a directed proof of a linear logic sequent in a theory is analogous to searching for an accepting ACM computation. The product of a successful search is an accepting computation.

Suppose the transition relation δ of a standard two counter machine with zero-test consists of the following:

δ_1 ::= Q_I **Increment** A Q_2
δ_2 ::= Q_3 **Decrement** A Q_F
δ_3 ::= Q_2 **Zero-Test** B Q_3

The machine may perform the following transitions, where an instantaneous description of a two counter machine is given by the triple consisting of Q_j, the current state, and the values of counters A and B.

$\langle Q_I, 0, 0\rangle \xrightarrow{\delta_1} \langle Q_2, 1, 0\rangle \xrightarrow{\delta_3} \langle Q_3, 1, 0\rangle \xrightarrow{\delta_2} \langle Q_F, 0, 0\rangle$

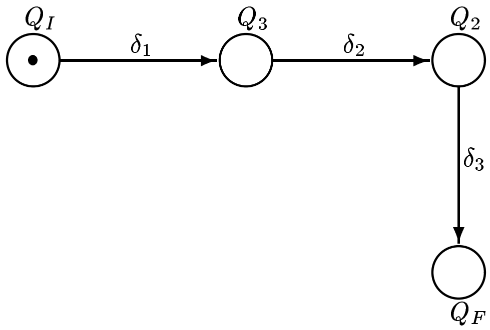


Figure 3.1
Oringal Machine

This computation starts in state Q_I, increments the A counter and steps to state Q_2. Then it tests the B counter for zero, and moves to Q_3, where it then decrements the A counter, moves to Q_F, and accepts. The transition relation can be represented as a finite state machine with four states, as depicted in Figure 3.1.

The transition relation δ may be transformed into a transition relation δ' for an equivalent and-branching two counter machine *without* zero-test. The modified relation δ' (shown on the left below), may then be encoded as a linear logic theory (shown on the right):

		Transitions	*Theory Axioms*
δ'_1	::=	Q_I **Increment** A Q_2	$q_I \vdash (q_2 \otimes a)$
δ'_2	::=	Q_3 **Decrement** A Q_F	$q_3 \vdash a, q_F$
δ'_3	::=	Q_2 **Fork** Z_B, Q_3	$q_2 \vdash (z_B \oplus q_3)$
δ'_4	::=	Z_B **Decrement** A Z_B	$z_B \vdash a, z_B$
δ'_5	::=	Z_B **Fork** Q_F, Q_F	$z_B \vdash (q_F \oplus q_F)$

Notice how the first two transitions (δ_1 and δ_2) of the standard two counter machine are preserved in the translation from δ to δ'. Also, the Zero-Test instruction δ_3 is encoded as three ACM transitions — δ'_3, δ'_4, and δ'_5. The transition δ'_3 is a fork to a special state Z_B, and one other state, Q_3. The two extra transitions, δ'_4 and δ'_5, force the computation branch starting in state Z_B to verify that counter B is

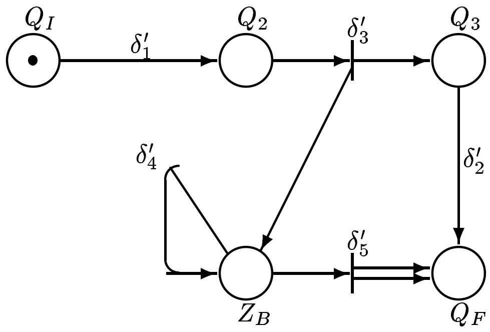


Figure 3.2
ACM machine

zero. In larger examples the transitions δ'_4, and δ'_5 and the state Z_B can be shared among all zero-test-B instructions. Given the above transitions, the and-branching machine without zero-test may then perform these moves:

$$
\begin{array}{ll}
 & \{\langle Q_I, 0, 0\rangle\} \\
\xrightarrow{\delta'_1} & \{\langle Q_2, 1, 0\rangle\} \\
\xrightarrow{\delta'_3} & \{\langle Z_B, 1, 0\rangle, \langle Q_3, 1, 0\rangle\} \\
\xrightarrow{\delta'_4} & \{\langle Z_B, 0, 0\rangle, \langle Q_3, 1, 0\rangle\} \\
\xrightarrow{\delta'_5} & \{\langle Q_F, 0, 0\rangle, \langle Q_F, 0, 0\rangle, \langle Q_3, 1, 0\rangle\} \\
\xrightarrow{\delta'_2} & \{\langle Q_F, 0, 0\rangle, \langle Q_F, 0, 0\rangle, \langle Q_F, 0, 0\rangle\}
\end{array}
$$

Note that an instantaneous description of this and-branching machine is a list of triples, and the machine accepts if and only if it is able to reach $\langle Q_F, 0, 0\rangle$ in *all* branches of its computation. This particular computation starts in state Q_I, increments the A counter and steps to state Q_2. Then it forks into two separate computations; one which verifies that the B counter is zero, and the other which proceeds to state Q_3. The B counter is zero, so the proof of that branch proceeds by decrementing the A counter to zero, and jumping to the final state Q_F. The other branch from state Q_3 simply decrements A and moves to Q_F. Thus all branches of the computation terminate in the final state with both counters at zero, resulting in an accepting computation. As before, the transition relation of the ACM can be represented as a finite state machine with five reachable states, as depicted in Figure 3.2.

The linear logic proof corresponding to this computation is displayed in Figures 3.3 and 3.4, and is explained in the following paragraphs. In these proofs, each application of a theory axiom corresponds to one step of ACM computation. We represent the values of the ACM counters in unary by copies of the formulas a and b. In this example the B counter is always zero, so there are no occurrences of b.

The proof shown in Figure 3.3 of $z_B, a \vdash q_F$ in the above linear logic theory corresponds to the ACM verifying that the B counter is zero. Reading the proof bottom up, it begins with a directed cut. The sequent $z_B \vdash q_F$ is left as an intermediate step. The next step is to use another directed cut, and after application of the & rule, we have two sequents left to prove: $q_F \vdash q_F$ and $q_F \vdash q_F$. Both of these correspond to the ACM triple $\langle Q_F, 0, 0\rangle$ which is the accepting triple, and are provable by the identity rule. If we had attempted to prove this sequent with some occurrences of b, we would be unable to complete the proof.

$$\dfrac{\dfrac{}{z_B, a \vdash z_B}\,\delta'_4 \qquad \dfrac{\dfrac{}{z_B \vdash (q_F \oplus q_F)}\,\delta'_5 \qquad \dfrac{\dfrac{}{q_F \vdash q_F}\,I \quad \dfrac{}{q_F \vdash q_F}\,I}{(q_F \oplus q_F) \vdash q_F}\,\oplus L}{z_B \vdash q_F}\,Cut}{z_B, a \vdash q_F}\,Cut$$

Figure 3.3
Zero-test proof

The proof shown in Figure 3.4 of $q_I \vdash q_F$ in the same theory demonstrates the remainder of the ACM machinery. The lowermost introduction of a theory axiom, **Cut**, and $\otimes$ **L** together correspond to the application of the increment instruction δ'_1. That is, the q_I has been "traded in" for q_2 along with a. The application of a directed cut and $\oplus$ **L** correspond to the fork instruction, δ'_3 which requires that both branches of the proof be successful in the same way that and-branching machines require all branches to reach an accepting configuration. The elided proof of $z_B, a \vdash q_F$ appears in Figure 3.3, and corresponds to the verification that the B counter is zero. The application of **Cut**, theory axiom, and identity correspond to the final decrement instruction of the computation, and complete the proof.

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{δ_1'}
\UnaryInfC{$q_I \vdash (q_2 \otimes a)$}
\AxiomC{}
\RightLabel{δ_3'}
\UnaryInfC{$q_2 \vdash (z_B \oplus q_3)$}
\AxiomC{$\vdots$}
\noLine
\UnaryInfC{$z_B, a \vdash q_F$}
\AxiomC{}
\RightLabel{δ_2'}
\UnaryInfC{$q_3, a \vdash q_F$}
\AxiomC{}
\RightLabel{I}
\UnaryInfC{$q_F \vdash q_F$}
\RightLabel{Cut}
\BinaryInfC{$q_3, a \vdash q_F$}
\RightLabel{$\oplus L$}
\BinaryInfC{$(z_B \oplus q_3), a \vdash q_F$}
\RightLabel{Cut}
\BinaryInfC{$q_2, a \vdash q_F$}
\RightLabel{$\otimes L$}
\UnaryInfC{$(q_2 \otimes a) \vdash q_F$}
\RightLabel{Cut}
\BinaryInfC{$q_I \vdash q_F$}
\end{prooftree}
$$

Figure 3.4
Proof corresponding to computation

3.4 Semantic Treatment

Here we present an alternative proof of Lemma 9 based on a semantic treatment due to Lafont [96].

A *phase space* is a commutative monoid M with an identified subset of M called $\bot$. We define $\mathbf{1} = (\{1\}^\bot)^\bot = \bot^\bot$, and I is the set of idempotents of $\mathbf{1}$. Then the following defintions are given for X and Y any subsets of M.

$$
\begin{aligned}
XY &\triangleq \{xy; x \in X \text{ and } y \in Y\} \\
X \multimap Y &\triangleq \{z \in M; \forall x \in X : xz \in Y\} \\
!X &\triangleq ((X \cup I)^\bot)^\bot \\
X^\bot &\triangleq X \multimap \bot
\end{aligned}
$$

A *fact* is some subset of M that is unchanged by double (linear) negation. That is, for any fact X it is the case that $(X^\bot)^\bot = X$. In particular, $\bot = \{1\}^\bot$ is a fact. Any intersection of facts is a fact, and if Y is a fact, then $X \multimap Y$ is a fact. Also, $(X^\bot)^\bot$ is the intersection of all facts containing X, which is also the smallest fact containing X. For any facts X and Y, the following definitions are made.

$$
\begin{aligned}
X \parr Y &\triangleq (X^\bot Y^\bot)^\bot \\
X \otimes Y &\triangleq (X^\bot \parr Y^\bot)^\bot = ((XY)^\bot)^\bot \\
X \& Y &\triangleq X \bigcap Y \\
X \oplus Y &\triangleq (X^\bot \& Y^\bot)^\bot = ((X \bigcup Y)^\bot)^\bot
\end{aligned}
$$

From these it follows that $X^\bot \parr Y = X \multimap Y$.

Next a mapping from linear logic formulas to M is given. If some fact $\alpha^\bullet$ is given for each atom α occuring in a (propositional) linear logic formula A, then the above definitions define the fact $A^\bullet$. We now get to the point of all these definitions. A phase space M *satisfies* a propositional linear logic formula A when $1 \in A^\bullet$.

THEOREM 5 (GIRARD) If A is provable in propositional linear logic then every phase space M satisfies A.

This is the main soundness theorem of the premier semantic treatement of linear logic. There is also a completeness theorem, however difficult it may be to use to derive provability results.

THEOREM 6 (GIRARD) If A is satisfied by all phase spaces for all interpretations of propositional atoms as facts, then A is provable in propositional linear logic.

Both these theorems can be extended to first- and second-order linear logic.

3.4.1 An Interesting Phase Space

The ACM construction can be finessed by directly considering the standard two counter machine constructed in the proof of Lemma 7. A very interesting phase space M is constructed from the free commutative monoid generated by the atoms $A, B, Q_0 \cdots Q_n$, and $\perp$ is the following set.

$$\{Q_i A^p B^q : \langle i, p, q\rangle \text{ is accepted by the machine}\} \bigcup \{Z_A B^p : p \in N\} \bigcup \{Z_B B^q : q \in N\}$$

Informally, this infinite phase space is constructed so that all and only accepting configurations are in $\perp$. The use of this phase space is to interpret provable sequents or formulas (which are interpreted in $\perp$) as evidence of an accepting computation.

We then define an interpretation of propositional atoms as $q_i^\bullet = (\{Q_i\}^\perp)^\perp$, $a^\bullet = (\{A\}^\perp)^\perp$, $b^\bullet = (\{B\}^\perp)^\perp$. Then since there are no outgoing arcs from the final state, $Q_F x \in \perp$ if and only if $x = A^0 B^0 = 1$, $\{Q_F\}^\perp = \{1\}$, $Q_F^\bullet = \{1\}^\perp = \perp$, and $\mathbf{1} = \perp^\perp = \{1\}$. Also, for any fact X, since in this phase space $I = \mathbf{1}$, $!X = ((X \bigcap I)^\perp)^\perp = X \& \mathbf{1}$.

The phase space M satisfies all the formulas encoding machine instructions. For an instruction Q_i **Increment** A Q_j encoded as $!(q_i \multimap q_j \otimes a)$, then the interpretation of this instruction is $[[(\{Q_i\} \multimap (\{Q_j A\}^\perp)^\perp) \bigcup I]^\perp]^\perp$ By the above remark regarding $!X$, this is equal to $(\{Q_i\} \multimap (\{Q_j A\}^\perp)^\perp)$, so we must show that $Q_i \in (\{Q_j A\}^\perp)^\perp)$. Consider an arbitrary $x \in \{Q_j A\}^\perp$. It must be the case that $Q_j A x \in \perp$, which means that x is of the form $A^p B^q$ and $\langle Q_j, p+1, q\rangle$ is accepted by the machine. Thus, configuration $\langle Q_i, p, q\rangle$ is accepted by the machine and $Q_i A^p B^q = Q_i x \in \perp$.

3.5 Recent Undecidability Results

Kanovich has shown that even without propositional symbols, linear logic is undecidable [81, 82]. His encoding follows this one quite closely. Where propositional symbols are used in the encoding of this chapter, Kanovich uses complex formulas involving the multiplicative connectives and multiplicative constants $(\otimes, ⅋, 1, \perp)$. In Section 5.2.3 it is shown that the multiplicative constants can be used to replace propositional symbols in an encoding of 3-Partition. In that case there are only two propositions, while in the undecidability proof the number of propositions used is unbounded (although the number of states plus the number of counters in some fixed universal counter machine plus three would suffice).

Using semantic techniques and building on the direct embedding of counter machines in linear logic [107, 83] and encoding of contraction and weakening in second-order linear logic [105] (see Section 6.4), Lafont has shown that one can encode reusable counter machine instructions as second-order formulas in second-order MALL, MALL2 [96]. Lafont and Scedrov refined that argument to apply to the smaller fragment MLL2 [97]. Recently Kopylov has demonstrated that these recent results can be extended even into logics that validiate weakening [92]. In particular, he showed that linear affine logic is undecidable. These semantic treatment of second-order quantifiers overcomes the dearth of proof-theoretic tools for higher-order logic.

Emms has developed a proof that the polymorphic lambek calculus (intuitionistic noncommutative multiplicative linear logic) is undecidable [47]. This result is based on some of the techniques of Section 6.4 and the key idea that one can encode the structural rule of exchange using second-order quantification, and thus one can obtain a direct embedding of LJ2 in noncommutative intuitionistic linear logic. This technique also demonstrates the undecidability of larger intuitionistic fragments containing the additive connectives. Kanovich has shown that in the same way that Lafont lifted the results in Section 6.4 from the intuitionistic setting to the classical setting in commutative linear logic, one can lift Emms result above from the noncommutative intuitionistic setting to noncommutative (classical) setting. That is, Kanovich showed that second-order noncommutative multiplicative logic is undecidable [84].

3.6 Summary of Chapter

In this section it was shown that propositional linear logic is undecidable. This fact follows from a computational reading of some linear logic proofs as (ACM) computations. The reduction from the halting problem for ACMs to a decision problem in linear logic is factored through decision problems in linear logic augmented with nonlogical theory axioms.

This undecidability result is perhaps somewhat surprising, but should not be viewed as a negative result: linear logic is an extremely expressive logic, and finding that the full (propositional) logic is undecidable merely implies that linear logic embodies notions closer to computing machinery than previously envisioned.

4 Circular Logic

> Beauty is the First test; there is no permanent place in the world for ugly mathematics GODFRY
>
> HAROLD HARDY 1877-1947
> *A mathematicians appology*, 1941

The following is called the *unrestricted exchange* rule:

$$\textbf{Exch. R} \quad \frac{\Theta \vdash \Sigma, A, B\Gamma}{\Theta \vdash \Sigma, B, A, \Gamma}$$

Since this rule is present in full linear logic, sequents are often treated as *multisets* of formulas, and the exchange rules are often considered implicit in the display of sequent proofs. This structural rule allows sequents to be permuted arbitrarily, making linear logic a commutative logic. More specifically, $\vdash (A \otimes B) \multimap (B \otimes A)$ is derivable in linear logic using exchange, as are the analogous sequents for all the other binary connectives of linear logic ($\invamp$, $\oplus$, $\&$). However, the absence of the **E** rule (treating sequents as *sequences* of formulas, without any rule of exchange) drasticly alters the set of provable sequents in linear logic. In fact, without the exchange rule, $\vdash (A \otimes B) \multimap (B \otimes A)$ is not derivable.

Noncommutative propositional linear logic is linear logic where the unrestricted exchange rule is omitted, or equivalently, where sequents are treated as being lists instead of multisets. One particular formulation of noncommutative linear logic allows sequents to be rotated, but not permuted, thus leading to the name *circular logic*.

This family of logics is somewhat speculative, though all are very closely related to work by Lambek [99],Yetter [157], and others. However, the immediate resulting system is unsatisfying in that the reusable formulas (those marked by ?) are exactly the ones which can be contracted and weakened in linear logic, and thus should be permitted the freedom of exchange, even in the noncommutative versions of linear logic. The present formulation therefore allows reusable formulas to permute.

There are a whole family of logics which could result from various additions of restricted exchange to noncommutative linear logic. The main point of difference within this family is the exact formulation of the rules of inference. However, as is shown by example below, most members of this family of logics have an undecidable validity problem.

In fact, the multiplicative and reuse operators are sufficient to encode undecidable problems in most of these logics. In other words, to simulate a Turing machine in noncommutative linear logic the constants and additive connectives are not necessary, although they appear to be necessary for this purpose in commutative linear logic. Below we present the detailed proof of undecidability for a particular logic we will call FICL, which is the multiplicative and exponential fragment of ICL, a member of the noncommutative linear logic family.

David Yetter [157] has also studied a variant of noncommutative linear logic. In his work, he considered a system with two new modalities, k and K, which are related to ? and !. The k modality essentially marks those formulas which are free to be permuted both left and right through a sequent, despite the noncommutativity of the logic in general. The reusable formulas (marked with ! on the left, or ? on the right) are allowed to permute, but are also allowed the freedom of contraction and weakening, while the k and K formulas are not. The undecidability result of this chapter therefore holds for Yetter's formulation of noncommutative linear logic as well.

More recently Kanovich has studied a "modalized" version of the Lambek calculus [81], which essentially amounts to noncommutative propositional linear logic with only two forms of implication ($\multimap$ and $\circ\!-$ which differ in this logic) and !. Kanovich claims that this logic is undecidable as well, amounting to a strengthening of the results presented below, since the multiplicative conjunction $\otimes$ is not needed. Kanovich claims some linguistic interpretation of ! in the context of the Lambek calculus, but this is beyond the scope of this book.

We now focus on FICL, a fragment of ICL sufficient to encode Turing machines. This logic includes only the multiplicative and exponential connectives of linear logic, excluding the additives and constants, but including restricted versions of the exchange rule which only applies to ! formulas.

4.1 Noncommutative Rules

The logic we work with in this chapter is intuitionistic noncommutative linear logic, where the modal ! is assumed to be allowed to commute.

Although ICL is undecidable, a more intriguing result is that a small fragment of ICL is also undecidable. We call this multiplicative-exponential fragment FICL.

However, one may simplify this system further, by considering the logic with nonlogical theories, but without the explicit modal operators ! and ?. One may encode such a logic in FICL using the ! and ?.

Identity	$\dfrac{}{A \vdash A}$	$\dfrac{\Gamma_1 \vdash A \quad \Gamma_2, A, \Gamma_3 \vdash C}{\Gamma_2, \Gamma_1, \Gamma_3 \vdash C}$	**Cut**
$\otimes$ **Left**	$\dfrac{\Gamma_1, A, B, \Gamma_2 \vdash C}{\Gamma, (A \otimes B), \Gamma_2 \vdash C}$	$\dfrac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash (A \otimes B)}$	$\otimes$ **Right**
$\multimap$ **Left**	$\dfrac{\Gamma_1 \vdash A \quad B, \Gamma_2 \vdash C}{\Gamma_1, (A \multimap B), \Gamma_2 \vdash C}$	$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash (A \multimap B)}$	$\multimap$ **Right**
$\multimapinv$ **Left**	$\dfrac{\Gamma_2 \vdash B \quad \Gamma_1, A \vdash C}{\Gamma_1, (A \multimapinv B), \Gamma_2 \vdash C}$	$\dfrac{\Gamma, A \vdash B}{\Gamma \vdash (B \multimapinv A)}$	$\multimap$ **Right**
$\oplus$ **Left**	$\dfrac{\Gamma, A \vdash \Sigma \quad \Gamma, B \vdash \Sigma}{\Gamma, (A \oplus B) \vdash \Sigma}$	$\dfrac{\Gamma \vdash A, \Sigma \quad \Gamma \vdash B, \Sigma}{\Gamma \vdash (A \& B), \Sigma}$	& **Right**
& **Left1**	$\dfrac{\Gamma, A \vdash \Sigma}{\Gamma, (A \& B) \vdash \Sigma}$	$\dfrac{\Gamma \vdash A, \Sigma}{\Gamma \vdash (A \oplus B), \Sigma}$	$\oplus$ **Right1**
& **Left2**	$\dfrac{\Gamma, B \vdash \Sigma}{\Gamma, (A \& B) \vdash \Sigma}$	$\dfrac{\Gamma \vdash B, \Sigma}{\Gamma \vdash (A \oplus B), \Sigma}$	$\oplus$ **Right2**
! W	$\dfrac{\Gamma_1, \Gamma_2 \vdash C}{\Gamma_1, !A, \Gamma_2 \vdash C}$	$\dfrac{\Gamma_1, !A, !A, \Gamma_2 \vdash C}{\Gamma_1, !A, \Gamma_2 \vdash C}$	**! C**
! D	$\dfrac{\Gamma_1, A, \Gamma_2 \vdash C}{\Gamma_1, !A, \Gamma_2 \vdash C}$	$\dfrac{!\Gamma \vdash A}{!\Gamma \vdash !A}$	**! S**
? D	$\dfrac{\Gamma \vdash A}{\Gamma \vdash ?A}$	$\dfrac{!\Gamma, A \vdash ?C}{!\Gamma, ?A \vdash ?C}$	**? S**
? W	$\dfrac{\Gamma \vdash}{\Gamma \vdash ?A}$		
! E1	$\dfrac{\Gamma_1, A, !B, \Gamma_2 \vdash C}{\Gamma_1, !B, A, \Gamma_2 \vdash C}$	$\dfrac{\Gamma_1, !B, A, \Gamma_2 \vdash C}{\Gamma_1, A, !B, \Gamma_2 \vdash C}$	**! E2**
0 **Left**	$\Gamma, 0 \vdash C$	$\Gamma \vdash \top$	$\top$ **Right**
1 **Left**	$\dfrac{\Gamma \vdash C}{\Gamma, 1 \vdash C}$	$\vdash 1$	1 **Right**

Figure 4.1
Sequent Rules for Circular Logic (ICL)

$$
\begin{array}{lccl}
\textbf{Identity} & \dfrac{}{A \vdash A} & \dfrac{\Gamma_1 \vdash A \quad \Gamma_2, A, \Gamma_3 \vdash C}{\Gamma_2, \Gamma_1, \Gamma_3 \vdash C} & \textbf{Cut} \\[2ex]
\otimes \textbf{ Left} & \dfrac{\Gamma_1, A, B, \Gamma_2 \vdash C}{\Gamma, (A \otimes B), \Gamma_2 \vdash C} & \dfrac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash (A \otimes B)} & \otimes \textbf{ Right} \\[2ex]
\multimap \textbf{ Left} & \dfrac{\Gamma_1 \vdash A \quad B, \Gamma_2 \vdash C}{\Gamma_1, (A \multimap B), \Gamma_2 \vdash C} & \dfrac{\Gamma, A \vdash B}{\Gamma \vdash (A \multimap B)} & \multimap \textbf{ Right} \\[2ex]
\textbf{! W} & \dfrac{\Gamma_1, \Gamma_2 \vdash C}{\Gamma_1, !A, \Gamma_2 \vdash C} & \dfrac{\Gamma_1, !A, !A, \Gamma_2 \vdash C}{\Gamma_1, !A, \Gamma_2 \vdash C} & \textbf{! C} \\[2ex]
\textbf{! D} & \dfrac{\Gamma_1, A, \Gamma_2 \vdash C}{\Gamma_1, !A, \Gamma_2 \vdash C} & \dfrac{!\Gamma \vdash A}{!\Gamma \vdash !A} & \textbf{! S} \\[2ex]
\textbf{! E1} & \dfrac{\Gamma_1, A, !B, \Gamma_2 \vdash C}{\Gamma_1, !B, A, \Gamma_2 \vdash C} & \dfrac{\Gamma_1, !B, A, \Gamma_2 \vdash C}{\Gamma_1, A, !B, \Gamma_2 \vdash C} & \textbf{! E2}
\end{array}
$$

Figure 4.2
Sequent Rules for FICL

4.2 FICL is Undecidable

We will show the word problem for semi-Thue systems has a straightforward encoding in FICL. Since we have already shown that full linear logic is undecidable, the fact that full noncommutative linear logic is undecidable is not too surprising. But since FICL is a fragment of noncommutative linear logic which does not contain the additive connectives, the earlier construction of and-branching two-counter machines in full linear logic would fail in FICL. However, the and-branching used in that construction was required in order to encode zero-test in a commutative setting. In a noncommutative setting a zero test operation may be encoded easily without any sort of and-branching. This situation is analogous to that for commutative versus noncommutative semi-Thue systems, where the noncommutative version allows the encoding of a zero test leading to undecidability, whereas the commutative version is unable to simulate zero test and has been shown to be decidable [118, 93]. In fact, since FICL closely resembles semi-Thue systems, we will demonstrate undecidability of FICL by a reduction from semi-Thue systems.

Although the reduction is intuitively simple, the proof of its correctness requires some elaborate machinery. In particular, a cut-elimination theorem is required.

LEMMA 10 (CUT ELIMINATION REVISITED) If there is a proof of sequent $\Gamma \vdash C$

in FICL, then there is a cut-free proof of $\Gamma \vdash C$ in FICL.

This lemma may be proven in the same manner as in the full logic 2.

COROLLARY 4.2.1 (SUBFORMULA PROPERTY REVISITED) Any formula in any cut-free FICL proof of $\Gamma \vdash C$ is a subformula of Γ or of C.

4.3 FICL Theories

We will define theories as for the commutative case (see Section 2.10), and show that cut-standardization (see Lemma 6) again holds in this logic.

Formally, a FICL *axiom* may be any FICL sequent of the form $p_{i_1}, p_{i_2}, \cdots p_{i_n} \vdash C$, where C is any FICL formula not including modal operators (? or !), and the remainder of the sequent is made up of negative literals. Any finite set of FICL axioms is a FICL *theory*. For any theory T, we say that a sequent $\Gamma \vdash C$ is provable in T exactly when we are able to derive $\Gamma \vdash C$ using the standard set of FICL proof rules and FICL axioms from T. Thus each axiom of T is treated as a reusable sequent which may occur as a leaf of a proof tree. As before we will write $\overline{\Gamma \vdash C}^{T}$ for a leaf sequent which is a member of the theory T.

We recall the definition of a *directed* cut:

A *directed* cut is one where at least one premise is an axiom in T, and the cut-formula in the axiom is not an atomic literal of negative polarity. We call any axiom premise of a directed cut where the cut-formula in that axiom is not a negative literal a *principal axiom* of that directed cut. By definition, all directed cuts have at least one principal axiom. A cut between two axioms is always directed, and if the cut-formula of such a cut is non-atomic, that cut has two principal axioms. A *directed* or *standardized* proof is a proof with all cuts directed.

LEMMA 11 (FICL CUT STANDARDIZATION) If there is a proof of $\Gamma \vdash C$ in theory T in FICL, then there is a directed proof of $\Gamma \vdash C$ in theory T in FICL.

This lemma may be proven in nearly the same way as cut-standardization in full linear logic (Lemma 6). In fact there are fewer cases here since the constants and additive connectives are not present in FICL.

An $\otimes$ **L** normalized directed proof is a directed proof where each application of $\otimes$ **L** has been permuted as far down the proof tree as possible.

LEMMA 12 ($\otimes$ NORMALIZATION) If there is a proof of $\Gamma \vdash C$ in theory T in FICL, then there is a $\otimes$ **L** normalized directed proof of $\Gamma \vdash C$ in theory T in FICL.

This lemma may be proven by appealing to Lemma 11 and a permutability lemma, which is not proven here, but which is directly analogous to the permutability of $\otimes$ **L** (or $\parr$ **R**) down in a sequent proof (see Section 2.9).

4.3.1 Theories into ficl

It happens that the translation of the theories into pure (commutative) linear logic may also be employed to encode FICL theories in pure FICL. We recall the definition of the translation $[T]$ of a theory T with k axioms into a multiset of formulas by

$$[\{t_1, t_2, \cdots, t_k\}] = [t_1], [t_2], \cdots, [t_k]$$

where $[t_i]$ is defined for each axiom t_i as follows:

$$\begin{array}{c}[\, G_1, G_2, \cdots, G_n \vdash F_1, F_2, \cdots, F_m \,] \\ \triangleq \\ !((G_1 \otimes G_2 \otimes \cdots \otimes G_n) \multimap (F_1 \parr F_2 \parr \cdots \parr F_m))\end{array}$$

The following theorem is directly analogous to Theorem 4, but is restated and proved here for the FICL case.

THEOREM 7 (THEORY $\Rightarrow$) For any finite set of FICL axioms T, $\Gamma \vdash C$ is provable in theory T only if $\Gamma, [T] \vdash C$ is provable in FICL without nonlogical axioms.

Proof 11 Given some proof of $\Gamma \vdash C$ in theory T, we have a linear logic proof tree with axioms of T at some leaves. For each leaf of the proof tree of the form $\Theta \vdash D$, where $\Theta \vdash D$ is some axiom t_i, we replace that leaf with a small proof of $\Theta, [T] \vdash D$ which does not use theory axioms (see below). For each leaf sequent which was originally an application of identity we extend the proof by adding weakenings for all the $![t_i]$ formulas. We then add $[T]$ to every sequent in the body of the entire proof tree. At every application of $\otimes R$, $\multimap L$, and **Cut**, we extend the proof tree with contractions on each formula in $[T]$.

The first mentioned proof tree, of $\Theta, [T] \vdash D$, will be constructed from the proof tree for $\Theta, [t_i] \vdash D$. Since each formula in $[T]$ begins with !, we may weaken in the remainder of $[T]$, and thus with some number of weakening steps we have $\Theta, [T] \vdash D$. For example, if there are k axioms, and $\Theta \vdash D$ is the axiom $t_1 = q_1 \vdash (q_2 \otimes a)$, then we know $[t_1] = !(q_1 \multimap (q_2 \otimes a))$. We then perform the following transformation:

$$\overline{q_1 \vdash (q_2 \otimes a)}^{\,T} \quad \Longrightarrow \quad \dfrac{\dfrac{\vdots}{\dfrac{\dfrac{\dfrac{\dfrac{\overline{q_1 \vdash q_1}^{\,I} \qquad \dfrac{\dfrac{\overline{q_2 \vdash q_2}^{\,I} \quad \overline{a \vdash a}^{\,I}}{q_2, a \vdash (q_2 \otimes a)}\otimes R}{(q_2 \otimes a) \vdash (q_2 \otimes a)}\otimes L}{q_1, (q_1 \multimap (q_2 \otimes a)) \vdash (q_2 \otimes a)}\multimap L}{q_1, [t_1] \vdash (q_2 \otimes a)}!D}{q_1, [t_1], [t_2] \vdash (q_2 \otimes a)}!W}{q_1, [t_1], [t_2], \cdots, [t_{k-1}] \vdash (q_2 \otimes a)}!W}{q_1, [T] \vdash (q_2 \otimes a)}!W$$

For each leaf sequent which was originally an application of identity we weaken in all the $[t_i]$ formulas:

$$\overline{p_i \vdash p_i}^{\,I} \quad \Longrightarrow \quad \dfrac{\dfrac{\vdots}{\dfrac{\dfrac{\dfrac{\overline{p_i \vdash p_i}^{\,I}}{p_i, [t_1] \vdash p_i}!W}{p_i, [t_1], [t_2] \vdash p_i}!W}{p_i, [t_1], [t_2], \cdots, [t_{k-1}] \vdash p_i}!W}}{p_i, [T] \vdash p_i}!W$$

We then continue by adding $[T]$ to every sequent in the entire proof tree. At every application of $\otimes R$, $\multimap L$, and **Cut**, we extend the proof tree with an extra copy of the conclusion sequent of the binary rule, to which we add an extra copy of $[T]$. Then we extend the proof further, adding one contraction step for each $[t_i]$ between that sequent and the original conclusion of that binary rule.

$$\dfrac{\dfrac{\vdots}{\Theta \vdash A} \qquad \dfrac{\vdots}{\Gamma \vdash B}}{\Theta, \Gamma \vdash (A \otimes B)}\otimes \quad \Longrightarrow \quad \dfrac{\dfrac{\vdots}{\dfrac{\dfrac{\dfrac{\vdots}{\Theta, [T] \vdash A} \qquad \dfrac{\vdots}{\Gamma, [T] \vdash B}}{\Theta, [T], \Gamma, [T] \vdash (A \otimes B)}\otimes}{}!C}}{\Theta, \Gamma, [T] \vdash (A \otimes B)}!C$$

Thus we have given a construction which builds a proof of $\Theta, [T] \vdash \Sigma$ without any nonlogical axioms from a given proof of $\Theta \vdash \Sigma$ using axioms from T.

The following theorem closely corresponds to Theorem 5.

THEOREM 8 (THEORY $\Leftarrow$) For any finite set of axioms T, $\Gamma \vdash C$ is provable in theory T if $\Gamma, [T] \vdash C$ is provable without nonlogical axioms.

Proof 12 Assuming we have a proof of $\Gamma, [T] \vdash C$, we immediately have a proof of $\Gamma, [T] \vdash C$ in theory T, since any proof in pure linear logic is also a proof in the logic extended with axioms. We can also build proofs of $\vdash [t_i]$ in the theory T for each axiom t_i. By cutting these proofs against the given proof of $\Gamma[T] \vdash C$, we obtain a proof of $\Gamma \vdash C$ in theory T.

For each axiom $t_i \equiv A_1, A_2, \cdots, A_n \vdash B$, we may prove $\vdash [t_i] \equiv \vdash !((A_1 \otimes A_2 \otimes \cdots \otimes A_n) \multimap B)$ by several applications of $\otimes$ **L**, one application of $\multimap$ **R**, and one application of **!S**, as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A_1, A_2, \cdots, A_n \vdash B}T}{\vdots}\otimes L}{A_1 \otimes A_2 \otimes \cdots \otimes A_n \vdash B}\otimes L}{\vdash (A_1 \otimes A_2 \otimes \cdots \otimes A_n) \multimap B}\multimap R}{\vdash !((A_1 \otimes A_2 \otimes \cdots \otimes A_n) \multimap B)}!S$$

By cutting this proof against the given proof of $\Gamma, [T] \vdash C$, we obtain a proof of $\Gamma, [T - \{t_i\}] \vdash C$, where $T - \{t_i\}$ is the multiset difference of T and $\{t_i\}$. Thus by induction on the number of axioms, cutting each one out in turn, we can derive $\Gamma \vdash C$ in theory T.

4.3.2 Semi-Thue Systems

A *semi-Thue system* $\mathcal{T}$ over alphabet Σ is a set of pairs $\langle x \rightarrow y \rangle$, where x and y are strings over Σ. Each pair in $\mathcal{T}$ is called a *production*, and we use them as rewrite rules. We call x the *left hand side* and y the *right hand side* of a production $\langle x \rightarrow y \rangle$. U *rewrites* to V in system $\mathcal{T}$ with a production $\langle g \rightarrow g' \rangle$ if U and V are words over alphabet Σ, and there exist possibly null words r and s such that $U = rgs$ and $V = rg's$. We write

$U \Longrightarrow V$

if U rewrites to V using some production. We use the notation

$U \Longrightarrow^* V$

if there exists a (possibly empty) sequence of words $U_1, \cdots, U_n$ such that

$U \Longrightarrow U_1 \Longrightarrow U_2 \Longrightarrow \cdots \Longrightarrow U_n \Longrightarrow V$

The *word problem* for a semi-Thue system $\mathcal{T}$ the problem of determining, for a given pair of words U and V, whether or not $U \Longrightarrow^* V$ in system $\mathcal{T}$. This problem is known to be undecidable [129]. The problem remains undecidable if we add the condition that V be a singleton (word of length one) n such that n does not occur in U or in the right hand side of any production, and only appears as a singleton on the left hand side of productions. This restriction is analogous to requiring that a Turing machine have a unique final state without any outgoing transitions.

To show that the above restrictions preserve undecidability it suffices to give a transformation from a general word problem to a word problem of the restricted class. Specifically, given a word problem from U to V, with set of productions $\mathcal{T}$, we may add the production $V \rightarrow n$ where n is a new symbol which is added to the alphabet Σ. We then ask the new word problem from U to n (with the new and the original productions), in the alphabet $\Sigma \cup \{n\}$. This problem is solvable if and only if the original problem is. However the new problem is of the restricted class defined above.

4.4 From Semi-Thue Systems to FICL

We overload the definition of translation $[\ ]$ to include the case of words — the translation of a word $[ab \cdots z]$ is the list of FICL formulas $p_a, p_b, \cdots, p_z$. We also define $[ab \cdots z]^2$ to be the FICL formula $p_a \otimes p_b, \cdots \otimes p_z$. Finally, as a notational convenience, we let $\gamma(\Gamma)$ designate ambiguously any formula which could be derived from Γ by applications of the $\otimes$ **L** rule. In other words, $\gamma(\Gamma)$ is the result of replacing some number of commas separating formulas in Γ by $\otimes$.

Given a Semi-Thue system $\mathcal{T} = \{\langle a_1 \rightarrow b_1\rangle, \langle a_2 \rightarrow b_2\rangle \cdots \langle a_k \rightarrow b_k\rangle\}$, we define the FICL theory derived from $\mathcal{T}$ as the following set of sequents:

$$\begin{array}{l} [a_1] \vdash [b_1]^2 \\ [a_2] \vdash [b_2]^2 \\ \quad\vdots \\ [a_k] \vdash [b_k]^2 \end{array}$$

For a word problem P consisting of the pair U, V we define the translation $\tau(P)$ of this problem into a FICL sequent as:

$$[U] \vdash [V]^2$$

Now, we show that the word problem P is solvable within system $\mathcal{T}$ if and only if the translation $\tau(P)$ is provable in the theory derived from $\mathcal{T}$. We state the two parts of the equivalence as Lemmas 13 and 14.

Lemma 13 The word problem P is solvable in the semi-Thue system $\mathcal{T}$ only if $\tau(P)$ is provable in the theory derived from $\mathcal{T}$.

Proof 13 We proceed by induction on the length of the derivation of $U \Longrightarrow^* V$. If the derivation is trivial, that is $U \equiv V$, then we must show that the sequent

$$\vdash [V], [V]^{\perp}$$

is provable. Since we assume the word problem has the restricted form with V a singleton, this sequent actually has the form $\vdash [n], [n]^{\perp}$, which by definition of notation is $\vdash p_n^{\perp}, p_n$. This is provable by identity.

Suppose the derivation of $U \Longrightarrow^* V$ is a nonempty sequence:

$$U \Longrightarrow U_1 \Longrightarrow U_2 \Longrightarrow \cdots \Longrightarrow U_n \Longrightarrow V.$$

Since $U \Longrightarrow U_1$, there is some rule $\langle g \rightarrow g' \rangle$ in $\mathcal{T}$, and possibly null words r and s such that $[U] = [rgs]$ and $[U_1] = [rg's]$. By induction we assume that we have a proof of $[U_1] \vdash [V]$, and construct from this a proof of the following form:

$$\dfrac{\dfrac{}{[g] \vdash [g']}\,{\scriptstyle T} \qquad \dfrac{\begin{array}{c} \vdots \\ \dfrac{\dfrac{[r], [g'], [s] \vdash [V]}{[r], \gamma([g']), [s] \vdash [V]}\,{\scriptstyle \otimes L}}{\vdots}\,{\scriptstyle \otimes L} \end{array}}{[r], [g']^2, [s] \vdash [V]}\,{\scriptstyle ⅋}}{[r], [g], [s] \vdash [V]}\,{\scriptstyle Cut}$$

In this partial deduction there are as many applications of the $\otimes$ **L** rule as there are separate formulas in $[g']$.

Thus, by induction, given a sequence of reductions which solves a word problem, we may simulate the solution in FICL.

The following concrete example illustrates the intended simulation. Assume that the first rule applied in the sequence of reductions is $\langle cd \rightarrow xy \rangle$. Then g in the above schema is cd, and g' is xy. Also, $[g]$ is p_c, p_d, $[g']$ is p_x, p_y, $[g']^2$ is $(p_x \otimes p_y)$, and $\gamma([g'])$ is $(p_x \otimes p_y)$. Also assume that r is $p_a p_b$, s is $p_e p_f$, and $[V]$ is p_n.

$$\dfrac{\dfrac{}{p_c, p_d \vdash (p_y \otimes p_x)}\,{\scriptstyle T} \qquad \dfrac{\begin{array}{c} \vdots \\ p_a, p_b, p_x, p_y, p_e, p_f \vdash p_n \end{array}}{p_a, p_b, (p_x \otimes p_y), p_e, p_f \vdash p_n}\,{\scriptstyle \otimes L}}{p_a, p_b, p_c, p_d, p_e, p_f \vdash p_n}\,{\scriptstyle Cut}$$

LEMMA 14 The word problem P is solvable with productions $\mathcal{T}$ if $\tau(P)$ is provable in the theory derived from $\mathcal{T}$.

Proof 14 We prove this lemma by induction on the size of the normalized directed proof of $\tau(P)$.

Inspecting the rules of FICL, we see that most rules are inapplicable, since the conclusion sequent, which is a translation $\tau(P)$ of a word problem, contains only occurrences propositions.

In the case that $\Gamma \vdash C$ is equal to $[V] \vdash [V]$, for some V, (Recall that the target word V of the word problem is a singleton — this proof may be a single application of the identity rule) then the solution to the word problem is trivial, i.e., no productions are required, since $U \equiv V$.

In the induction case where $\Gamma \vdash C$ is not proven by identity, the only remaining case is **Cut**. By cut-standardization, we know that one hypothesis is an axiom, and the cut-formula in that axiom is not a negative literal. Inspecting the axioms in the theory corresponding to the productions of a semi-Thue system, we see that the cut-formula must always be a formula $[g']^2$. This formula is built up from positive literals connected by $\otimes$. By the normalization property, we know that we may now apply $\otimes$ **L** until we are left again with the inductive case of some $\tau(P')$.

Thus the semi-Thue system may mimic the FICL proof by applying the production corresponding to the theory axiom applied, and we have our result by induction.

THEOREM 9 The provability problem for FICL is recursively unsolvable.

COROLLARY 4.4.1 The provability problem for ICL is recursively unsolvable.

This corollary follows from Theorem 9 by a conservativity result which is easily derived from the cut-elimination and subformula properties of FICL.

4.5 Other Noncommutative Logics

As mentioned previously, there is a family of logics which share a strong resemblance to FICL. All of the ones we can sensibly imagine have undecidable decision problems.

The first generalization that one might make is to loosen the intuitionistic restriction of FICL, allowing multiple formulas to appear on the right hand side of a sequent.

We will now consider various possibilities of formalizations of such a system, where all formulas appear on the right of the $\vdash$. When specific rules are mentioned, it is their use on the right that is meant. For example, in the remainder of this chapter, the $\otimes$ **R** rule will be referred to as the $\otimes$ rule.

4.5.1 One-Sided Noncommutative Linear Logic: cl

The one-sided rules of noncommutative linear logic (CL are given in Figure 4.3.

These rules contain the two novel rules of rotation (**R**) and restricted exchange (**!E**). This system is very close to that studied by Yetter [157], although Yetter's extra modalities k and K have been omitted.

The one-sided calculus CL is a conservative extension over the two-sided intuitionistic calculus FICL. In order to make this precise, we require some definitions. The notation $[\Gamma]^{\perp}$ stands for the negation of a sequence of formulas, which is defined to the sequence of negations of formulas in Γ, in reverse order.

LEMMA 15 Any well-formed FICL sequent $\Gamma \vdash C$, is provable in FICL if and only if $\vdash [\Gamma]^{\perp}, C$ is provable in CL.

A broader class of conservativity results are available, going beyond the small fragment FICL, but the logic CL is not conservative over the whole two-sided intuitionistic logic ICL.

4.6 Summary of Chapter

In this chapter the undecidability of a small fragment of propositional noncommutative linear logic was demonstrated. This logic contains fewer connectives than those required for the proof of undecidability of (commutative) linear logic. The proof uses semi-Thue systems, although perhaps a more intuitive proof could use standard Turing machines, viewing the left of the turnstile as a direct representation of the Turing tape. The instructions must be reusable, and so are marked by !, and here we assume that all three structural rules (exchange, contraction, and weakening) are allowed for ! formulas.

The interest in this logic stems from work by Lambek [99] predating the introduction of linear logic. Lambek's work considered one of the possible variants of noncommutative linear logic (and did not contain exponentials, and so is decidable). The later sections of this chapter consider some other alternative formulations of noncommutative linear logic. Although these proof-theoretic investigations are suggestive, perhaps the best source of insight into the correct axiomatization would be semantic considerations and applications such as Lambek's.

$$\mathbf{I} \qquad \frac{}{\vdash p_i, p_i{}^{\perp}}$$

$$\mathbf{Cut} \qquad \frac{\vdash \Sigma, A \qquad \vdash \Gamma, A^{\perp}}{\vdash \Sigma, \Gamma}$$

$$\otimes \qquad \frac{\vdash \Sigma, A \qquad \vdash B, \Gamma}{\vdash \Sigma, (A \otimes B), \Gamma}$$

$$\parr \qquad \frac{\vdash \Sigma, A, B}{\vdash \Sigma, (A \parr B)}$$

$$\oplus \qquad \frac{\vdash \Sigma, A}{\vdash \Sigma, (A \oplus B)} \qquad \frac{\vdash \Sigma, B}{\vdash \Sigma, (A \oplus B)}$$

$$\& \qquad \frac{\vdash \Sigma, A \qquad \vdash \Sigma, B}{\vdash \Sigma, (A \mathbin{\&} B)}$$

$$\mathbf{?W} \qquad \frac{\vdash \Sigma}{\vdash \Sigma, ?A}$$

$$\mathbf{?C} \qquad \frac{\vdash \Sigma, ?A, ?A}{\vdash \Sigma, ?A}$$

$$\mathbf{?D} \qquad \frac{\vdash \Sigma, A}{\vdash \Sigma, ?A}$$

$$\mathbf{!S} \qquad \frac{\vdash ?\Sigma, A}{\vdash ?\Sigma, !A}$$

$$\perp \qquad \frac{\vdash \Sigma}{\vdash \Sigma, \perp}$$

$$1 \qquad \frac{}{\vdash 1}$$

$$\top \qquad \frac{}{\vdash \Sigma, \top}$$

$$\mathbf{R} \qquad \frac{\vdash \Gamma, A}{\vdash A, \Gamma}$$

$$\mathbf{!E} \qquad \frac{\vdash \Gamma, ?A, \Sigma}{\vdash \Gamma, \Sigma, ?A}$$

Figure 4.3
Sequent Rules for One-Sided Circular Logic (CL)

5 Decidable Fragments of Linear Logic

This chapter briefly covers decidable fragments of linear logic for which complexity results are known. The main results are that without exponentials or non-logical theory axioms, linear logic is PSPACE-complete, and the multiplicative fragment of linear logic is NP-complete.

5.1 MALL is PSPACE-complete

In this section, we analyze the complexity of the fragment of propositional linear logic without the modal storage operator ! and its dual ?, but including all the remaining connectives and constants of linear logic. Earlier it was shown that with the addition of nonlogical theories to MALL the decision problem is recursively unsolvable. Here we study the complexity of the decision problem for MALL in the absence of theory axioms.

In this section we restrict our attention to the right hand side of the sequent arrow $\vdash$. The results of this section immediately carry over to the two sided version of the calculus at the expense of greater case analysis at many steps in the following proofs. Technically, we must restate the **Cut** rule as follows, essentially incorporating the negation rule into the **Cut** rule.

$$\textbf{Cut} \quad \frac{\vdash \Sigma, A \qquad \vdash \Gamma, A^{\perp}}{\vdash \Sigma, \Gamma}$$

An important property of the sequent calculus formulation of MALL is cut-elimination. This property follows from Theorem 2, but since we are restricting our attention to the one sided case, we restate the theorem explicitly.

THEOREM 10 Any sequent provable in MALL is provable without the cut rule.

Proof 15 Since MALL is a fragment of linear logic, we may use the cut-elimination procedure from Theorem 2 to convert a MALL proof to a cut-free proof in linear logic. By the subformula property (Theorem 2.7.1), such a cut-free proof of a MALL conclusion sequent contains only MALL formulas. Thus any cut-free proof of a MALL sequent is already a MALL proof.

Membership in PSPACE is straightforward given cut elimination, but we include a short sketch to illustrate the importance of Theorem 10. The proof of PSPACE-hardness is more involved and is presented below. Proof search in the cut-free sequent calculus is crucial to the proof. The primitive step in proof search is a ɪre-duction, namely the application of an inference rule to transform a sequent matching the conclusion of the rule to the collection of sequents given by the corresponding premises of the rule. A reduction is the inverse of an inference rule, and drives conclusions to premises. Proof search is the process of constructing a cut-free proof in a bottom-up manner by nondeterministically applying reductions starting from the conclusion sequent.

5.1.1 Membership in PSPACE

THEOREM 11 The provability in MALL of a given sequent can be decided by a polynomial-space bounded Turing machine.

Proof 16 By Theorem 10, a provable MALL sequent has a cut-free MALL proof. In a cut-free MALL proof, there are at most two premises to each rule, and each premise is strictly smaller than the consequent. Therefore, the depth of a cut-free MALL proof tree is at most linear in the length of the final sequent of the proof. An alternating Turing machine [39] may guess and check a cut-free proof in linear time, using OR-branching to nondeterministically guess a reduction in the cut-free proof, and AND-branching to generate and check the proofs of both premises of a two premise rule in parallel.

Membership in PSPACE can also be proved without reference to alternation. A nondeterministic Turing machine can be defined to generate and check a cut-free sequent proof in a depth-first manner. Given the linear bound on the depth of any cut-free proof with respect to the size of the conclusion sequent, the search stack need contain no more than a linear number of sequents. Since each sequent in a cut-free proof is no larger than the conclusion sequent, we get a quadratic bound on the stack size.

5.1.2 Informal Outline of PSPACE-hardness of MALL

Since there are a number of technical details to the proof of PSPACE-hardness, we will illustrate the key intuitions by means of an example; the details of the proof are given in Subsection 5.1.6.

The PSPACE-hardness of MALL provability is demonstrated by a transformation from the validity problem for quantified Boolean formulas (QBF). A *quantified Boolean formula* has the (prenex) form $Q_m X_m \ldots Q_1 X_1 : M$, where

1. each Q_i is either $\forall$ or $\exists$,
2. M is a quantifier-free *Boolean matrix* containing only the connectives $\neg$ and $\wedge$, and *Boolean variables*.

A *closed* QBF contains no free variables. Our conventions in this section are that G and H range over quantified Boolean formulas; M and N range over quantifier-free Boolean formulas; U, V, X, Y, Z range over Boolean variables; and I ranges over truth value assignments. For expository convenience, we refer to quantifier-free Boolean formulas simply as Boolean formulas.

An *assignment* I for a set of Boolean variables $\{X_1, \ldots, X_n\}$ maps each X_i to a truth value from $\{T, F\}$. An assignment is represented by a sequence of Boolean variables and negated Boolean variables. For example, the assignment $X_1, \neg X_2, X_3$ maps X_1 to T, X_2 to F, and X_3 to T. The assignment I, X assigns T to X, but behaves like I, otherwise. If I is an assignment for the free variables in G, we use the standard notation $I \models G$ to indicate that G is valid under I, and write $I \not\models G$ if I falsifies G. Note that

$$I \models \forall X : G \text{ iff } I, X \models G \quad \text{and} \quad I, \neg X \models G$$
$$I \models \exists X : G \text{ iff } I, X \models G \quad \text{or} \quad I, \neg X \models G$$

If G is a QBF and I is an assignment for the free variables in G, we say *G is valid under I* exactly if $I \models G$. If G is a closed QBF, then G is said to be valid if it is valid under the empty assignment. The validity of a closed QBF G is represented as $\models G$. The QBF validity problem is: *Given a closed QBF G, is G valid?*

We demonstrate the PSPACE-hardness of MALL provability by defining a succinct encoding of a QBF as a MALL sequent that is provable *exactly* when the given QBF is valid.

The transformation of the QBF validity problem to MALL provability takes place in two steps: 1. Given a quantifier-free Boolean formula M and an assignment I for the free variables in M, we show that there is a MALL sequent encoding M and I which is provable exactly when M is valid under I. This essentially demonstrates that the process of evaluating Boolean functions can be represented by the process of cut-free proof search in the MALL sequent calculus. 2. Given a QBF G and an assignment I for the free variables in G, there exists a MALL sequent encoding the quantifier prefix and the Boolean matrix of G so that the MALL sequent is provable exactly when G is valid under I. The idea here is to simulate the Boolean quantifiers $\exists$ and $\forall$ by using the additive connectives $\oplus$ and $\&$.

Two-sided vs. one-sided sequents. We use only one-sided MALL sequents in the following encodings to reduce the number of cases in some of the proofs. In linear logic, a two-sided sequent $A_1, \ldots, A_m \vdash B_1, \ldots, B_n$ has the one-sided form $\vdash A_1^{\perp}, \ldots, A_m^{\perp}, B_1, \ldots, B_n$. Thus, a formula $A \multimap B$ on the left of a two-sided sequent becomes $A \otimes B^{\perp}$ in a one-sided sequent. Similarly, the provable two-sided sequent $A, A \multimap B \vdash B$ becomes $\vdash A^{\perp}, A \otimes B^{\perp}, B$. While one-sided sequents simplify the technical arguments considerably, the reader might gain further insight by rewriting parts of our encoding in a two-sided form.

5.1.3 Encoding Boolean Evaluation

The encoding of the Boolean connectives and quantifiers in MALL is described here by means of an example. The full definition of the encoding appears in Section 5.1.5. The encoding from QBF validity to MALL provability makes no use of the MALL constants. Consider the *valid* QBF G given by

$$\forall X_2 \exists X_1 : \neg(\neg X_1 \wedge X_2) \wedge \neg(\neg X_2 \wedge X_1).$$

The matrix M of G is essentially a restatement of $(X_1 \iff X_2)$. Let H be the falsifiable formula $\exists X_1 \forall X_2 : M$ that is obtained from G by reversing the order of the quantifiers. It is crucial that the encodings of G and H in MALL respect the ordering of quantifiers so that the encoding of G is provable but the encoding of H is not.

The encoding of the Boolean matrix describes the formula as a circuit with signals labeled by MALL literals. Let the assignment I be encoded by a sequence of MALL formulas $\langle I \rangle$, and $[M]_a$ be the MALL formula encoding M with output labeled by the literal a. Then $I \models M$ is encoded by the sequent

$$\vdash \langle I \rangle, [M]_a, a$$

whereas $I \not\models M$ is encoded by

$$\vdash \langle I \rangle, [M]_a, a^{\perp}.$$

Since we are using one-sided sequents, we encode the assignment $X_1, \neg X_2$ by $x_1^{\perp}, x_2$. The MALL literals encoding the assignment are to be seen as the input signals to the encoding of the Boolean formula.

We first consider the Boolean connectives $\neg$ and $\wedge$, then construct the full encoding of M. The encoding $[\neg X_1]_a$ of $\neg X_1$ with output labeled a is the formula $\mathrm{NOT}(x_1, a)$. For literals x and y, the definition of $\mathrm{NOT}(x, y)$ is just the representation of the truth table for negation within MALL, as shown below:

$$\mathrm{NOT}(x, y) = (x \otimes y) \oplus (x^{\perp} \otimes y^{\perp}). \tag{5.1.1}$$

$\mathrm{NOT}(x_1, a)$ is simply the linear negation of the formula

$$(x_1 \multimap a^{\perp}) \& (x_1^{\perp} \multimap a)$$

which is more perspicuous in describing a as the Boolean negation of x_1. The sequent

$$\vdash x_1, \text{NOT}(x_1, a), a \tag{5.1.2}$$

encodes the situation where the input X_1 is F, and asserts (correctly) that the output $\neg X_1$ is T.

The sequent (5.1.2) is easily seen to have the MALLproof

$$\dfrac{\dfrac{\dfrac{}{\vdash x_1, x_1^{\perp}}I \qquad \dfrac{}{\vdash a^{\perp}, a}I}{\vdash x_1, (x_1^{\perp} \otimes a^{\perp}), a}\otimes}{\vdash x_1, (x_1 \otimes a) \oplus (x_1^{\perp} \otimes a^{\perp}), a}\oplus$$

Similarly, the sequent (5.1.3) representing $\{X_1 \leftarrow T\} \not\models \neg X_1$ is also provable.

$$\vdash x_1^{\perp}, \text{NOT}(x_1, a), a^{\perp}. \tag{5.1.3}$$

On the other hand, the sequent

$$\vdash x_1^{\perp}, \text{NOT}(x_1, a), a \tag{5.1.4}$$

asserts (falsely) that $\{X_1 \leftarrow T\} \models \neg X_1$. To see why sequent (5.1.4) is not provable, we observe that MALL is a refinement of classical logic in which no classically falsifiable sequents are provable. The sequent $\vdash x_1^{\perp}, \text{NOT}(x_1, a), a$ is falsified by assigning T to x_1 and F to a, while interpreting $\otimes$ and $\&$ as classical conjunction and $\oplus$ and $\invamp$ as classical disjunction. A sequent is interpreted classically as the disjunction of the sequence of formulas that it contains.

The encoding for conjunction, $[X \wedge Y]_b$ is given by $\text{AND}(x, y, b)$ as defined below.

$$\text{AND}(x, y, b) = \begin{bmatrix} (x \otimes y \otimes b^{\perp}) & \oplus \\ (x^{\perp} \otimes y^{\perp} \otimes b) & \oplus \\ (x \otimes y^{\perp} \otimes b) & \oplus \\ (x^{\perp} \otimes y \otimes b) & \end{bmatrix} \tag{5.1.5}$$

Sequent (5.1.6) represents $X, Y \models (X \wedge Y)$:

$$\vdash x^{\perp}, y^{\perp}, \text{AND}(x, y, b), b. \tag{5.1.6}$$

Sequent (5.1.6) has the proof

$$\dfrac{\dfrac{\dfrac{}{\vdash x^{\perp}, x}I \qquad \dfrac{\dfrac{}{\vdash y^{\perp}, y}I \qquad \dfrac{}{\vdash b^{\perp}, b}I}{\vdash y^{\perp}, (y \otimes b^{\perp}), b}\otimes}{\vdash x^{\perp}, y^{\perp}, (x \otimes y \otimes b^{\perp}), b}\otimes}{\vdash x^{\perp}, y^{\perp}, \text{AND}(x, y, b), b}\oplus$$

As with sequent (5.1.4), the MALL sequent representing the false assertion $\neg X, Y \models (X \wedge Y)$ is given by

$$\vdash x, y^{\perp}, \textsc{and}(x, y, b), b$$

and is not provable since it can be falsified by the classical interpretation assigning F to x and b, and T to y.

The next step is to construct the encoding of the Boolean formula M given at the beginning of this section, from the encodings of the Boolean connectives. The formula M is thought of as a Boolean circuit with the distinctly labeled signals. The encoding $[(\neg X_1 \wedge X_2)]_b$ is given by the formula $\textsc{and}(a, x_2, b) \parr \textsc{not}(x_1, a)$. Let $\textsc{imp}(x, y, u, v, w)$ represent the formula

$$\textsc{not}(v, w) \parr \textsc{and}(u, y, v) \parr \textsc{not}(x, u),$$

then $\textsc{imp}(x_1, x_2, a, b, c)$ is the encoding $[\neg(\neg X_1 \wedge X_2)]_c$. The literals a, b and c are the distinct literals labeling the output signals of the Boolean gates.

We now consider the problem that the input signals in M have a fanout greater than one. An almost correct encoding in MALL of the Boolean formula M is given by the formula

$$\textsc{and}(c, f, g) \parr \textsc{imp}(x_1, x_2, a, b, c) \parr \textsc{imp}(x_2, x_1, d, e, f).$$

The validity of M under the assignment $\{X_1 \leftarrow T, X_2 \leftarrow T\}$ would then be represented by

$$\begin{array}{ll} \vdash & \textsc{and}(c, f, g) \parr \textsc{imp}(x_1, x_2, a, b, c) \parr \textsc{imp}(x_2, x_1, d, e, f), \\ & x_1^{\perp}, x_2^{\perp}, g \end{array} \tag{5.1.7}$$

The following deduction represents one attempt to prove sequent (5.1.7).

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash g^{\perp}, g}\,I \qquad \dfrac{\begin{array}{c}\vdots\\ \vdash x_1^{\perp}, x_2^{\perp}, \textsc{imp}(x_1, x_2, a, b, c), c\end{array} \quad \begin{array}{c}\vdots\\ \vdash \textsc{imp}(x_2, x_1, d, e, f), f\end{array}}{\vdash x_1^{\perp}, x_2^{\perp}, (c \otimes f), \textsc{imp}(x_1, x_2, a, b, c), \textsc{imp}(x_2, x_1, d, e, f)}\,\otimes}{\vdash x_1^{\perp}, x_2^{\perp}, (c \otimes f \otimes g^{\perp}), \textsc{imp}(x_1, x_2, a, b, c), \textsc{imp}(x_2, x_1, d, e, f), g}\,\otimes}{\vdash x_1^{\perp}, x_2^{\perp}, \textsc{and}(c, f, g), \textsc{imp}(x_1, x_2, a, b, c), \textsc{imp}(x_2, x_1, d, e, f), g}\,\oplus}{\vdash x_1^{\perp}, x_2^{\perp}, \textsc{and}(c, f, g) \parr \textsc{imp}(x_1, x_2, a, b, c) \parr \textsc{imp}(x_2, x_1, d, e, f), g}\,\parr$$

Since MALL lacks a rule of contraction, each of the assignment literals, $x_1^{\perp}$ and $x_2^{\perp}$, can appear in only one premise of a $\otimes$ rule. As a result, one of the remaining subgoals in the above deduction lacks the required input literals. We therefore need to be able to explicitly duplicate the assignment literals in the sequent (5.1.7) to match the number of duplicate occurrences of X_1 and X_2 in M. The formula $\textsc{copy}(x_1)$ defined as

$$(x_1 \otimes (x_1^\perp \parr x_1^\perp)) \oplus (x_1^\perp \otimes (x_1 \parr x_1))$$

serves to duplicate an instance of x_1 or $x_1^\perp$. If M is now encoded as

$$\textsc{and}(c,f,g) \parr \textsc{copy}(x_1) \parr \textsc{copy}(x_2) \parr \Gamma_1 \parr \Gamma_2$$

where Γ_1 abbreviates $\textsc{imp}(x_1,x_2,a,b,c)$, and Γ_2 abbreviates $\textsc{imp}(x_2,x_1,d,e,f)$, the desired deduction of (5.1.7) can then be constructed.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash x_1, x_1^\perp}I \quad \dfrac{\dfrac{}{\vdash x_2, x_2^\perp}I \quad \dfrac{\dfrac{}{\vdash g, g^\perp}I \quad \dfrac{\begin{matrix}\vdots \\ \vdash x_1^\perp, x_2^\perp, \Gamma_1, c\end{matrix} \quad \begin{matrix}\vdots \\ \vdash x_1^\perp, x_2^\perp, \Gamma_2, f\end{matrix}}{\vdash x_1^\perp, x_1^\perp, x_2^\perp, x_2^\perp, (c\otimes f), \Gamma_1, \Gamma_2}\otimes}{\vdash x_1^\perp, x_1^\perp, x_2^\perp, x_2^\perp, (c\otimes f \otimes g^\perp), \Gamma_1, \Gamma_2, g}\otimes}{\vdash x_1^\perp, x_1^\perp, x_2^\perp, \textsc{and}(c,f,g), x_2 \otimes (x_2^\perp \parr x_2^\perp), \Gamma_1, \Gamma_2, g}\otimes}{\vdash x_1^\perp, x_2^\perp, \textsc{and}(c,f,g), x_1 \otimes (x_1^\perp \parr x_1^\perp), \textsc{copy}(x_2), \Gamma_1, \Gamma_2, g}\otimes}{\vdash x_1^\perp, x_2^\perp, \textsc{and}(c,f,g), \textsc{copy}(x_1), \textsc{copy}(x_2), \Gamma_1, \Gamma_2, g}\oplus}{\vdash x_1^\perp, x_2^\perp, \textsc{and}(c,f,g) \parr \textsc{copy}(x_1) \parr \textsc{copy}(x_2) \parr \Gamma_1 \parr \Gamma_2, g}\parr$$

In summary, we have informally described the encoding in MALL of the evaluation of Boolean formulas under an assignment. The connectives $\parr$, $\otimes$, and $\oplus$ were used to represent the truth tables of $\neg$ and $\wedge$, and MALL literals were used to represent the "signals" in the Boolean formula. The duplication of input signals forms a crucial part of the encoding since MALL lacks a rule of contraction.

5.1.4 Encoding Boolean Quantification

Recall that G is the formula $\forall X_2 \exists X_1 : M$, and H is the formula $\exists X_1 \forall X_2 : M$, where M is $\neg(\neg X_1 \wedge X_2) \wedge \neg(\neg X_2 \wedge X_1)$. Intuitively, it is useful to separate the encoding of the Boolean quantifier prefix as separately encoding the individual quantifiers and the dependencies between quantifiers. Given the above encoding for assignments and Boolean formulas, an almost correct way to encode Boolean quantifiers would be to encode $\exists X_1$ as the formula $(x_1 \oplus x_1^\perp)$, and $\forall X_2$ as $(x_2 \& x_2^\perp)$. The encoding of G would then be given by the sequent

$$\vdash (x_2 \& x_2^\perp), (x_1 \oplus x_1^\perp), [M]_g, g.$$

The formula $(x_1 \oplus x_1^\perp)$ behaves like existential quantification in proof search since a nondeterministic choice can be made between

$$\dfrac{\begin{array}{c}\vdots\\ \vdash x_1^\perp, \Gamma\end{array}}{\vdash (x_1 \oplus x_1^\perp), \Gamma}\oplus \qquad \text{and} \qquad \dfrac{\begin{array}{c}\vdots\\ \vdash x_1, \Gamma\end{array}}{\vdash (x_1 \oplus x_1^\perp), \Gamma}\oplus$$

according to the assignment (T or F, respectively) to X_1 which makes $\exists X_1 : M$ valid. Similarly, the rule for reducing $(x_2 \& x_2^\perp)$ in a proof behaves like universal quantification requiring proofs of both $\vdash x_2^\perp, \Gamma$ and $\vdash x_2, \Gamma$.

$$\dfrac{\begin{array}{c}\vdots\\ \vdash x_2^\perp, \Gamma\end{array} \quad \begin{array}{c}\vdots\\ \vdash x_2, \Gamma\end{array}}{\vdash (x_2 \& x_2^\perp), \Gamma}\&$$

However, with this mapping of quantifiers, the MALL encoding of G and H would be identical and provable, but H is not a valid QBF.

A correct encoding of $\exists X_1 \forall X_2 : M$ should ensure that if the encoding is provable in MALL, then there is a proof in which the choice of a truth value for X_1 is independent of whether X_2 is T or F. The order of reductions below show how the choice of a truth value for $\exists X_1$ in a proof of the MALL encoding can depend on the quantifier $\forall X_2$.

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \vdash x_1, x_2, \Gamma\end{array}}{\vdash (x_1 \oplus x_1^\perp), x_2, \Gamma}\oplus \quad \dfrac{\begin{array}{c}\vdots\\ \vdash x_1^\perp, x_2^\perp, \Gamma\end{array}}{\vdash (x_1 \oplus x_1^\perp), x_2^\perp, \Gamma}\oplus}{\vdash (x_1 \oplus x_1^\perp), (x_2 \& x_2^\perp), \Gamma}\&$$

In this ordering of the reductions, $(x_1 \oplus x_1^\perp)$ is reduced differently on the x_2 and $x_2^\perp$ branches of the proof leading to distinct witnesses for X_1 according to whether X_2 is T or F. The solution to this quantifier order problem is to encode the quantifier dependencies in the MALL formula so that if there is any proof, then there is some proof of the encoding in which $(x_1 \oplus x_1^\perp)$ is reduced *below* $(x_2 \& x_2^\perp)$, thus ensuring that the truth value of X_1 has been chosen independently of the truth value for X_2. For this purpose, we introduce new MALL atoms q_0, q_1, q_2, and encode $\exists X_1 \forall X_2 : M$ as

$$\begin{array}{ll}\vdash & q_2,\\ & q_2^\perp \otimes ((q_1 \parr x_1) \oplus (q_1 \parr x_1^\perp)),\\ & q_1^\perp \otimes ((q_0 \parr x_2) \& (q_0 \parr x_2^\perp)),\\ & q_0^\perp \otimes [M]_g, g\end{array}$$

The idea here is that the quantifier encoding for $\exists X_1$ hides the "key" q_1 that is needed to unlock the quantifier encoding for $\forall X_2$. If we now attempt to violate the quantifier dependencies, the following would be one possible deduction.

$$
\dfrac{
  \dfrac{
    \dfrac{}{\vdash q_2, q_2^\perp}\,I
    \qquad
    \dfrac{
      \dfrac{
        \begin{array}{c} ? \\ \vdash q_1^\perp, q_1, x_1 \end{array}
      }{\vdash q_1^\perp, q_1 \parr x_1}\,\parr
    }{\vdash q_1^\perp, ((q_1 \parr x_1) \oplus (q_1 \parr x_1^\perp))}\,\oplus
  }{\vdash q_2, q_1^\perp, q_2^\perp \otimes ((q_1 \parr x_1) \oplus (q_1 \parr x_1^\perp))}\,\otimes
  \qquad \vdots
}{
  \begin{array}{ll}
  \vdash & q_2, q_2^\perp \otimes ((q_1 \parr x_1) \oplus (q_1 \parr x_1^\perp)), \\
  & q_1^\perp \otimes ((q_0 \parr x_2) \& (q_0 \parr x_2^\perp)), q_0^\perp \otimes [M]_g, g
  \end{array}
}\,\otimes
$$

In the above deduction, we are left with a subgoal of the form $\vdash q_1^\perp, q_1, x_1$, and since x_1 is not a constant, we cannot reduce this sequent to a MALL axiom. (Recall that MALL lacks an unrestricted weakening rule.) Other deductions attempting to violate the quantifier ordering also fail. On the other hand, the deduction which does respect the order of the quantifier encodings can be performed as shown below. The quantifier encoding for $\exists X_1$ provides the key q_1 for unlocking the quantifier encoding of $\forall X_2$.

$$
\dfrac{
  \dfrac{}{\vdash q_2, q_2^\perp}\,I
  \qquad
  \dfrac{
    \dfrac{
      \begin{array}{ll}
      & \vdots \\
      \vdash & q_1, x_1, \\
      & q_1^\perp \otimes ((q_0 \parr x_2) \& (q_0 \parr x_2^\perp)), \\
      & q_0^\perp \otimes [M]_g, g
      \end{array}
    }{
      \begin{array}{ll}
      \vdash & q_1 \parr x_1, \\
      & q_1^\perp \otimes ((q_0 \parr x_2) \& (q_0 \parr x_2^\perp)), \\
      & q_0^\perp \otimes [M]_g, g
      \end{array}
    }\,\parr
  }{
    \begin{array}{ll}
    \vdash & ((q_1 \parr x_1) \oplus (q_1 \parr x_1^\perp)), \\
    & q_1^\perp \otimes ((q_0 \parr x_2) \& (q_0 \parr x_2^\perp)), \\
    & q_0^\perp \otimes [M]_g, g
    \end{array}
  }\,\oplus
}{
  \begin{array}{ll}
  \vdash & q_2, q_2^\perp \otimes ((q_1 \parr x_1) \oplus (q_1 \parr x_1^\perp)), \\
  & q_1^\perp \otimes ((q_0 \parr x_2) \& (q_0 \parr x_2^\perp)), q_0^\perp \otimes [M]_g, g
  \end{array}
}\,\otimes
$$

The formal definition of the polynomial time encoding of QBF validity in terms of MALL provability is given in Section 5.1.5. In Section 5.1.6, we demonstrate the correctness of the encoding.

5.1.5 Formal Definition of the Encoding

For our purpose, a Boolean formula is constructed from Boolean variables using the Boolean connectives $\neg$ and $\wedge$. All quantified variables are assumed to occur in the matrix. $Var(M)$ is the set of variables occurring in the Boolean formula M. Overlined syntactic variables such as $\overline{X}$ and $\overline{Y}$ range over sets of Boolean variables.

The mall sequent encoding a QBF G is represented by $\sigma(G)$. We need to be careful about keeping literals distinct. The annotation "a new" in the definition indicates that the literal a is a freshly chosen one that has not been used elsewhere in the encoding.

The sequent $\sigma(G)$ consists of the encoding of the QBF $[\![G]\!]_g$, where g labels the output signal, the key q_n, and the output value g. The definition of $[\![G]\!]_g$ constructs the quantifier encodings by induction on the length of the quantifier prefix. The definition of $[M]_g$ is by induction on the structure of M, so that $[N \wedge P]_g$ is constructed by 1. choosing the fresh labels a and b for the outputs of subformulas N and P, respectively 2. defining the relation between a, b, and g by $\textsc{and}(a,b,g)$ 3. if needed, providing a copying formula for each Boolean variable common to both N and P, and 4. recursively constructing $[N]_a$ and $[P]_b$

To be precise, we provide the following definition of the encoding.

Definition 5.1.1

$$
\begin{array}{rcll}
\sigma(G) & = & \vdash q_n, [\![G]\!]_g, g & q_n, g \text{ new} \\
[\![(\forall X_{i+1} : G)]\!]_g & = & (q_{i+1}^{\perp} \otimes ((x_{i+1} \parr q_i) \& (x_{i+1}^{\perp} \parr q_i))), [\![G]\!]_g & q_{i+1} \text{ new} \\
[\![(\exists X_{i+1} : G)]\!]_g & = & (q_{i+1}^{\perp} \otimes ((x_{i+1} \parr q_i) \oplus (x_{i+1}^{\perp} \parr q_i))), [\![G]\!]_g & q_{i+1} \text{ new} \\
[\![M]\!]_g & = & (q_0^{\perp} \otimes [M]_g) & q_0 \text{ new} \\
[X]_g & = & (x^{\perp} \otimes g) \oplus (x \otimes g^{\perp}) & \\
[\neg N]_g & = & \textsc{not}(a,g) \parr [N]_a & a \text{ new} \\
[N \wedge P]_g & = & \left\{ \begin{array}{rl} \textsc{and}(a,b,g) & \parr \\ \textsc{copyall}(Var(N) \cap Var(P)) & \parr \\ [N]_a & \parr \\ [P]_b & \end{array} \right. & a, b \text{ new} \\
[N \wedge P]_g & = & \textsc{and}(a,b,g) \parr [N]_a \parr [P]_b & a, b \text{ new} \\
\textsc{copyall}(\overline{X}) & = & \parr_{X_i \in \overline{X}} \textsc{copy}(x_i) &
\end{array}
$$

The lower (simpler) definition of $[N \wedge P]_g$ is only applicable in the case that $Var(N) \cap Var(P) = \emptyset$. In the other case, where $Var(N) \cap Var(P) \neq \emptyset$, the $\textsc{copyall}$ definition must be used.

Note that the sequent $\sigma(G)$ contains no $\textsc{mall}$ constants. The complexity of computing $\sigma(G)$ is at most quadratic in the size of G since the encoding function is defined inductively over the structure of the formula, and the intersection operation can be performed in linear time with a bit-vector representation of sets, where the length of each bit-vector is the number of distinct Boolean variables occurring in G. The cost of constructing the COPY formulas at each step in the recursion is also linear in the size of G. The cost of each NOT and AND formula is fixed with respect to the representation of the literals, and the literals can be represented with a cost that is logarithmic in the size of G.

The encoding may be computed in log-space, although the algorithm described above uses more than log-space, because of the work space required to save the set of variables that must be copied when encoding a conjunction. The encoding algorithm could be modified to make a number of passes over the input to determine the number of occurrences of each variable and generate the required number of COPY formulas. Each pass would use only log-space, and the remainder of the algorithm may be performed in log-space.

5.1.6 Proof of pspace-hardness of mall

The main theorem is that for any closed QBF G, G is valid if and only if $\sigma(G)$ is a provable MALL sequent. The first set of lemmas demonstrates that the encoding of Boolean formulas works correctly. The second set of lemmas demonstrates that the Boolean quantifiers have been correctly encoded.

If I is a truth value assignment for the Boolean variables $X_1, \ldots, X_n$, then I is encoded as $\langle I \rangle$, where

$$\begin{array}{rcl} \langle I \rangle & = & \langle X_1 \rangle_I, \ldots, \langle X_n \rangle_I \\ \langle X_i \rangle_I & = & \left\{ \begin{array}{cl} x_i^{\perp} & \text{if } I(X_i) = T \\ x_i, & \text{otherwise} \end{array} \right. \end{array}$$

If I is an assignment for a set of variables $\overline{Y}$, and $\overline{X} \subseteq \overline{Y}$, then $I/\overline{X}$ is the assignment I restricted to the subset $\overline{X}$, and by abuse of notation I/M is $I/Var(M)$. The following lemma is stated without proof.

LEMMA 16 Given sets of variables $\overline{X}$ and $\overline{Y}$, and an assignment I for $\overline{X} \cup \overline{Y}$, there is a deduction of the sequent $\vdash \langle I \rangle, \text{COPYALL}(\overline{X} \cap \overline{Y}), \Gamma$ from the sequent $\vdash \langle I/\overline{X} \rangle, \langle I/\overline{Y} \rangle, \Gamma$.

LEMMA 17 For any Boolean formula M and assignment I for the variables in M, 1. if $I \models M$ then $\vdash \langle I \rangle, [M]_g, g$ 2. if $I \not\models M$ then $\vdash \langle I \rangle, [M]_g, g^{\perp}$

Proof 17 By induction on the structure of M, as follows. The cases in the proof correspond closely to those in the definition of $[M]_g$.

Base case: $M \equiv X$. Suppose $I(X) = T$, then $I \models M$ and $\langle I \rangle = x^\perp$. The following proof can then be constructed, expanding the definition of $[M]_g$

$$\dfrac{\dfrac{\dfrac{}{\vdash x^\perp, x}I \quad \dfrac{}{\vdash g^\perp, g}I}{\vdash x^\perp, (x \otimes g^\perp), g}\otimes}{\vdash x^\perp, (x^\perp \otimes g) \oplus (x \otimes g^\perp), g}\oplus$$

The case when $I(X) = F$ is similarly straightforward.

Induction step: There are a number of cases here corresponding to the definition of $[M]_g$. We consider a typical case and leave the remaining ones to the reader.

Let $M \equiv N \wedge P$, and suppose that $Var(N) \cap Var(P) \neq \emptyset$. Consider the case when $I/N \models N$ and $I/P \not\models P$, so that $I \not\models N \wedge P$. Expanding $[M]_g$, $\textsc{and}(a, b, g)$, and using Lemma 16, the following deduction can be constructed.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash g, g^\perp}I \quad \dfrac{\dfrac{\vdots}{\vdash \langle I/N \rangle, [N]_a, a} \quad \dfrac{\vdots}{\vdash \langle I/P \rangle, [P]_b, b^\perp}}{\vdash \langle I/N \rangle, \langle I/P \rangle, (a \otimes b^\perp), [N]_a, [P]_b}\otimes}{\vdash \langle I/N \rangle, \langle I/P \rangle, (a \otimes b^\perp \otimes g), [N]_a, [P]_b, g^\perp}\otimes}{\vdash \langle I/N \rangle, \langle I/P \rangle, \textsc{and}(a, b, g), [N]_a, [P]_b, g^\perp}\oplus}{\vdots}}{\vdash \langle I \rangle, \textsc{and}(a, b, g), \textsc{copyall}(Var(N) \cap Var(P)), [N]_a, [P]_b, g^\perp}}{\vdash \langle I \rangle, \textsc{and}(a, b, g) \parr \textsc{copyall}(Var(N) \cap Var(P)) \parr [N]_a \parr [P]_b, g^\perp}\parr$$

Applying the induction hypothesis to I/N, N, and a, and to I/P, P, and b, we can establish that the remaining subgoals of the deduction are provable.

The remaining subcases in the evaluation of $N \wedge P$ are similar, as are the remaining cases in the induction argument.

The next step is to establish the converse of Lemma 17. The classical interpretation of the MALL connectives may be used to give a relatively easy proof. In the classical interpretation, truth values, T and F, are assigned to the MALL atoms, $A^\perp$ is read as classical negation, $A \otimes B$ and $A \& B$ are read as classical conjunction, and $A \parr B$ and $A \oplus B$ are read as classical disjunction. A sequent is interpreted as the classical disjunction of the formulas contained in it.

LEMMA 18 If $\vdash \Gamma$ is a provable MALL sequent, then for any assignment of truth values to the atoms in Γ, there exists a formula A in the sequence Γ such that A is true under the classical interpretation.

Proof 18 The proof is by a straightforward induction on cut-free MALL proofs. Clearly, for axioms $\vdash x, x^{\perp}$, one of x or $x^{\perp}$ must evaluate to T in a given truth assignment. In the induction case, suppose that the last step in the proof of Γ is a $\otimes$-rule of the form

$$\dfrac{\begin{matrix}\vdots \\ \vdash B, \Gamma_1\end{matrix} \qquad \begin{matrix}\vdots \\ \vdash C, \Gamma_2\end{matrix}}{\vdash (B \otimes C), \Gamma_1, \Gamma_2}\otimes$$

By the induction hypothesis, the sequence B, Γ_1 contains a formula A_1, and the sequence C, Γ_2 contains a formula A_2, and both A_1 and A_2 are true. If A_1 is different from B, then A_1 occurs in the conclusion sequent yielding the required A, and similarly, when A_2 is different from C. Otherwise, the formula $(B \otimes C)$ is $(A_1 \otimes A_2)$ and is hence true under the classical interpretation of $\otimes$ as conjunction. The induction arguments corresponding to the other connectives are similar.

The main intuition behind Lemma 19 is that by appropriately assigning truth values to the literals in $\langle I \rangle$ and $[M]_g$, it is possible to mimic the evaluation of the Boolean formula M under I. Due to our use of one-sided sequents and the form of our encoding, there is exactly one truth value falsifying each formula in $\langle I \rangle$ and $[M]_g$. This assignment turns out be the appropriate one, i.e., the value of g under this assignment is T exactly when $I \models M$. For example, if I is $\{X \leftarrow F\}$ and M is $\neg X$, then $\langle I \rangle$ is $x^{\perp}$ and $[M]_g$ is $(x \otimes g) \oplus (x^{\perp} \otimes g^{\perp})$. The only falsifying assignment here is $\{x \leftarrow F, g \leftarrow T\}$.

LEMMA 19 Let M be a Boolean formula and I be an assignment for the variables in M. There exists an assignment K of truth values to the atoms in $\langle I \rangle$ and $[M]_g$ such that for every formula A in the sequence $\langle I \rangle, [M]_g$, assignment K falsifies A under the classical interpretation, and $K(g) = T$ iff $I \models M$.

Proof 19 The proof is by induction on the construction of $[M]_g$. Note that the induction is parametric in I and g (I and g are universally quantified in the induction hypothesis), so that when $M \equiv (N \wedge P)$, the induction hypothesis on N has I/N replacing I and a replacing g, where a labels the output of N.

Base case: $M \equiv X$. Then $[M]_g = (x^{\perp} \otimes g) \oplus (x \otimes g^{\perp})$. If $I \models X$, then $I(X) = T$ and $\langle I \rangle = x^{\perp}$, and $\langle I \rangle$ is falsified if K assigns T to x. $[M]_g$ is falsified if K assigns T to g, and the second part of the conclusion, $K(g) = T$ also follows. If $I \not\models X$, then $I(X) = F$. Let K assign F to x and F to g to falsify both $\langle I \rangle$ and $[M]_g$. Then $K(g) = F$ as required.

Induction step: Observe first that the formula $\text{COPY}(x)$ defined as $(x \otimes (x^{\perp} \parr x^{\perp})) \oplus (x^{\perp} \otimes (x \parr x))$ is classically false.

When $M \equiv \neg N$, the encoding $[M]_g$ is $\text{NOT}(a, g), [N]_a$. By the induction hypothesis, we have an assignment K_1 falsifying $\langle I \rangle, [N]_a$ such that $K_1(a) = T$ iff $I \models N$. Suppose $K_1(a) = T$, and hence $I \models N$. The formula $\text{NOT}(a, g)$ is $(a \otimes g) \oplus (a^{\perp} \otimes g^{\perp})$. Let K be $K_1\{g \leftarrow F\}$. Since g does not occur in $\langle I \rangle$ or $[N]_a$, K agrees with K_1 on $\langle I \rangle, [N]_a$. The assignment K also falsifies $\text{NOT}(a, g)$, thus falsifying $\langle I \rangle, [M]_g$. Note that $K(g) = F$ as required, since $I \not\models M$.

If $K_1(a) = F$, then $I \not\models N$. Letting K be $K_1\{g \leftarrow T\}$ falsifies $\langle I \rangle, [M]_g$.

When $M \equiv (N \wedge P)$, then by the induction hypotheses for N and P, there exists

1. K_1 falsifying $\langle I/N \rangle, [N]_a$ such that $K_1(a) = T$ iff $I/N \models N$, and
2. K_2 falsifying $\langle I/P \rangle, [P]_b$ such that $K_2(a) = T$ iff $I/P \models P$

The encoding $\langle I \rangle$ is a sequence of literals such that no two distinct literals in $\langle I \rangle$ share a common atom. Since $\langle I/N \rangle$ and $\langle I/P \rangle$ are subsets of $\langle I \rangle$, there is no literal x such that x is in $\langle I/N \rangle$ and $x^{\perp}$ is in $\langle I/P \rangle$. Formulas $[N]_a$ and $[P]_b$ have no atoms in common outside of those in $\langle I \rangle$. Then the union of the assignments $K_1 \cup K_2$, is still an assignment, i.e., it assigns a unique truth value to each *atom* in $\langle I \rangle, [N]_a, [P]_b$.

Suppose that $K_1(a) = T$, and hence $I/N \models N$, and $K_2(b) = F$, so that $I/P \not\models P$. Let K be $(K_1 \cup K_2)\{g \leftarrow F\}$. Note that g does not occur in $\langle I \rangle$, $[N]_a$ or $[P]_b$ so that K agrees with K_1 on $[N]_a$ and with K_2 on $[P]_b$. Each disjunct in $\text{AND}(a, b, g)$ expanded as

$$\begin{array}{cc}
(a \otimes b \otimes g^{\perp}) & \oplus \\
(a^{\perp} \otimes b^{\perp} \otimes g) & \oplus \\
(a \otimes b^{\perp} \otimes g) & \oplus \\
(a^{\perp} \otimes b \otimes g) &
\end{array}$$

is falsified by K. As already observed, the COPY formulas are all classically false, and thus K falsifies $\langle I \rangle, [M]_g$. Since in this case, $I \not\models N \wedge P$, the second part of the conclusion is also satisfied.

The remaining cases are similar.

LEMMA 20 If I is an assignment for the variables in a given Boolean formula M, then

1. if $\vdash \langle I \rangle, [M]_g, g$ is provable, $I \models M$
2. if $\vdash \langle I \rangle, [M]_g, g^{\perp}$ is provable, $I \not\models M$.

Proof 20 By Lemma 18, we know that if $\vdash \langle I \rangle, [M]_g, g$ is provable, then no assignment can simultaneously falsify $\langle I \rangle$, $[M]_g$, and g under the classical interpretation.

By Lemma 19, we can find an assignment K which falsifies $\langle I \rangle$ and $[M]_g$ such that $K(g) = T$ iff $I \models M$. Since K cannot also falsify g, $K(g) = T$ and hence $I \models M$.

Similarly, when $\vdash \langle I \rangle, [M]_g, g^\perp$ is provable, we can, by Lemmas 19 and 18, find an assignment K such that $K(g) = F$ and as a consequence, $I \not\models M$.

LEMMA 21 $\vdash \langle I \rangle, [M]_g, g$ is provable iff $I \models M$.

Proof 21 Follows immediately from Lemmas 17 and 20.

So far, we have demonstrated the correctness of the encoding of the Boolean matrix of a given quantified Boolean formula. The remainder of the proof deals with the encoding of Boolean quantifiers. The next lemma states the crucial reason why the MALL encoding of quantifiers is faithful to the quantifier orderings. As observed in Subsection 5.1.4, the goal is to ensure that in any successful proof search, the ith quantifier encoding is reduced after, i.e, above, the reduction of the $(i + 1)$st quantifier encoding in any cut-free proof. To achieve this, we need to argue that the key q_i needed to unlock the ith quantifier encoding is only made available when the $(i + 1)$st quantifier encoding has been reduced. In order for the ith quantifier encoding, which has the form $q_i^\perp \otimes U_i$, to be reduced before the $(i+1)$st quantifier encoding, a subgoal of the form $\vdash q_i^\perp, \Gamma$ would have to be provable. The only occurrences of q_i are in the subformula U_{i+1} given by $(q_i \parr x_{i+1}) \circ (q_i \parr x_{i+1}^\perp)$, where $\circ$ may be either $\oplus$ or $\&$. If U_{i+1} occurs in Γ, then the only occurrences of q_i in Γ are as immediate arguments to a $\parr$. By exploiting the absence of an unrestricted weakening rule in MALL, it can be shown that in the absence of constants, $\vdash q_i^\perp, \Gamma$ is not provable when all of the occurrences of q_i in Γ appear as immediate arguments to $\parr$. Therefore, regardless of whether U_{i+1} occurs in Γ, the sequent $\vdash q_i^\perp, \Gamma$ would not be provable, thus making it impossible to reduce the ith quantifier encoding below the $(i + 1)$st quantifier encoding in a cut-free proof.

LEMMA 22 If q is a positive or negative literal and the sequent $\vdash q, \Gamma$ contains no constants, then $\vdash q, \Gamma$ is provable only if either $\Gamma \equiv q^\perp$ or Γ contains at least one occurrence of a subformula either of the form $q^\perp \circ A$, or the form $A \circ q^\perp$, where $\circ$ may be either $\oplus$, $\&$, or $\otimes$.

Proof 22 We fix $\circ$ to be either $\oplus$, $\&$, or $\otimes$ for this proof. The proof is by induction on cut-free MALL proofs of $\vdash q, \Gamma$. In the base case, for a cut-free proof of depth 0, the sequent $\vdash q, \Gamma$ must be a MALL axiom, and $\Gamma \equiv q^\perp$ holds.

In the induction step, when in the given cut-free proof of $\vdash q, \Gamma$, the conclusion sequent is derived by an application of either a $\otimes$, $\&$ or a $\oplus$ rule, then at least one premise must be of the form $\vdash q, \Delta$. We know by the induction hypothesis for the proof of $\vdash q, \Delta$, either $\Delta = q^{\perp}$ or Δ either contains a subformula of the form $q^{\perp} \circ A$, or the form $A \circ q^{\perp}$. In either case, Γ contains one of the forms, $q^{\perp} \circ A$ or $A \circ q^{\perp}$.

If in the cut-free proof of $\vdash q, \Gamma$, the conclusion sequent is derived by an application of the $\parr$ rule, the premise sequent must be of the form $\vdash q, \Delta$, where Δ is not a single formula, Then by the induction hypothesis on the proof of $\vdash q, \Delta$, the sequence Δ must contain one of the forms, $q^{\perp} \circ A$ or $A \circ q^{\perp}$. Since every subformula of Δ is a subformula of Γ as well, Γ must also contain one of the forms $q^{\perp} \circ A$ or $A \circ q^{\perp}$.

The following lemma demonstrates the correctness of the MALL encoding of Boolean quantifiers. Each Q_i in the statement below is either $\forall$ or $\exists$.

LEMMA 23 (MAIN INDUCTION) Let M be a Boolean formula in the variables $X_1, \ldots, X_n$, then for any m, $0 \leq m \leq n$, and assignment I for $X_{m+1}, \ldots, X_n$, the relation $I \models Q_m X_m \ldots Q_1 X_1 : M$ holds iff the sequent $\vdash q_m, \langle I \rangle, [\![Q_m X_m \ldots Q_1 X_1 : M]\!]_g, g$ is provable in MALL.

Proof 23 The proof is by induction on m between 0 and n. Note that I is universally quantified in the induction hypothesis.

Base case $\Rightarrow$: Here $m = 0$. Then $[\![M]\!]_g \equiv q_0^{\perp} \otimes [M]_g$, and we can easily construct the following deduction of the required conclusion $\vdash q_0, \langle I_0 \rangle, [\![M]\!]_g, g$.

$$\dfrac{\dfrac{}{\vdash q_0, q_0^{\perp}}\, I \qquad \dfrac{\vdots}{\vdash \langle I \rangle, [M]_g, g}}{\vdash \langle I \rangle, q_0, q_0^{\perp} \otimes [M]_g, g}\, \otimes$$

The proof of the remaining subgoal $\vdash \langle I \rangle, [G]_g, g$, follows from Lemma 17.

Base case $\Leftarrow$: The deduction shown above is the only possible one in a cut-free proof of $\vdash \langle I \rangle, q_0, q_0^{\perp} \otimes [M]_g, g$ since $q_0^{\perp} \otimes [G]$ is the only compound formula in the conclusion. So if $\vdash \langle I \rangle, q_0, q_0^{\perp} \otimes [M]_g, g$ is provable, by Theorem 10, it must have a cut-free proof containing a proof of $\vdash \langle I \rangle, [M]_g, g$. By Lemma 20, we get $I \models M$.

Induction step $\Rightarrow$: Assume $0 < m \leq n$. Let G abbreviate $Q_{m-1} X_{m-1} \ldots Q_1 X_1 : M$. We must prove the lemma for $Q_m X_m G$. If $Q_m \equiv \exists$, then

$$\llbracket Q_m X_m \colon G \rrbracket_g = (q_m^\perp \otimes ((x_m \parr q_{m-1}) \oplus (x_m^\perp \parr q_{m-1}))), \llbracket G \rrbracket_g$$

If $I \models \exists X_m : G$, then either $I, X_m \models G$ or $I, \neg X_m \models G$. In the former case, the following deduction of the required conclusion can be constructed.

$$\dfrac{\dfrac{}{\vdash q_m, q_m^\perp}\,I \qquad \dfrac{\begin{array}{c}\vdots\\ \vdash \langle I\rangle, x_m^\perp, q_{m-1}, \llbracket G\rrbracket_g, g\end{array}}{\vdash \langle I\rangle, ((x_m \parr q_{m-1}) \oplus (x_m^\perp \parr q_{m-1})), \llbracket G\rrbracket_g, g}\,\oplus}{\vdash \langle I\rangle, q_m, (q_m^\perp \otimes ((x_m \parr q_{m-1}) \oplus (x_m^\perp \parr q_{m-1}))), \llbracket G\rrbracket_g, g}\,\otimes$$

Since $\langle I, X_m\rangle = \langle I\rangle, x_m^\perp$, the induction hypothesis can be applied to show that the remaining subgoal of the above deduction is provable.

When $I, \neg X_m \models G$, the proof construction only differs from the above one on the $\oplus$ rule corresponding to the quantifier encoding.

If $Q_m \equiv \forall$, then

$$\llbracket \forall X_m \colon G \rrbracket_g = (q_m^\perp \otimes ((x_m \parr q_{m-1}) \& (x_m^\perp \parr q_{m-1}))), \llbracket G \rrbracket_g$$

Since $I \models \forall X_m : G$, it follows that $I, X \models G$ and $I, \neg X \models G$. The following deduction can be constructed.

$$\dfrac{\dfrac{}{\vdash q_m, q_m^\perp}\,I \qquad \dfrac{\begin{array}{c}\vdots\\ \vdash \langle I\rangle, x_m, q_{m-1}, \llbracket G\rrbracket_g, g\end{array} \quad \begin{array}{c}\vdots\\ \vdash \langle I\rangle, x_m^\perp, q_{m-1}, \llbracket G\rrbracket_g, g\end{array}}{\vdash \langle I\rangle, ((x_m \parr q_{m-1}) \& (x_m^\perp \parr q_{m-1})), \llbracket G\rrbracket_g, g}\,\&}{\vdash \langle I\rangle, q_m, (q_m^\perp \otimes ((x_m \parr q_{m-1}) \& (x_m^\perp \parr q_{m-1})), \llbracket G\rrbracket_g, g}\,\otimes$$

Since $\langle I, X_m\rangle$ is $\langle I\rangle, x_m^\perp$ and $\langle I, \neg X_m\rangle$ is $\langle I\rangle, x_m$, the two remaining subgoals in the deduction are provable by the induction hypotheses.

Induction step $\Leftarrow$: This is the critical step in the proof. We are given that $m > 0$ and that the conclusion sequent $\vdash q_m, \langle I\rangle, \llbracket Q_m X_m \ldots Q_1 X_1 : M\rrbracket_g, g$ is provable. Theorem 10 can be applied to construct a cut-free proof of $\vdash q_m, \langle I\rangle, \llbracket Q_m X_m \ldots Q_1 X_1 : M\rrbracket_g, g$. We show that this cut-free proof respects the quantifier ordering, i.e., the reduction of the encoding of $Q_m X_m$ occurs below any other step in the proof.

It is easy to see that every formula in the multiset $\llbracket Q_m X_m \ldots Q_1 X_1 : M\rrbracket_g$ is of the form $q_i^\perp \otimes A_i$, for $0 \leq i \leq m$, with $A_0 \equiv [M]_g$, and $A_{j+1} \equiv ((x_{j+1} \parr q_j) \circ (x_{j+1}^\perp \parr q_j))$. The connective written as $\circ$ can be either $\&$ or $\oplus$. The formulas $q_i^\perp \otimes A_i$ are the only compound formulas in the conclusion sequent $\vdash q_m, \langle I\rangle, \llbracket Q_m X_m \ldots Q_1 X_1 :$

$M]\!]_g, g$. From the MALL rules, it is clear that the only applicable reduction in a cut-free proof search would be an application of the $\otimes$-rule. Hence for some k, we can partition the formulas other than $q_k^\perp \otimes A_k$ in the conclusion sequent into Γ and Δ to get a deduction of the conclusion sequent of the following form.

$$\dfrac{\begin{array}{c}\vdots \\ \vdash \Gamma, q_k^\perp\end{array} \qquad \begin{array}{c}\vdots \\ \vdash A_k, \Delta\end{array}}{\vdash \Gamma, (q_k^\perp \otimes A_k), \Delta}\otimes$$

Suppose for the sake of deriving a contradiction that $k < m$. Recall that there are no constants in the encoding. The formula $q_{k+1}^\perp \otimes A_{k+1}$ must either occur in Γ or Δ, and definitely not in both. Since the only occurrences of q_k are within A_{k+1}, by Lemma 22, if $q_{k+1}^\perp \otimes A_{k+1}$ occurs in Δ, then we cannot complete the proof $\vdash q_k, \Gamma$. Thus we assume $q_{k+1}^\perp \otimes A_{k+1}$ occurs in Γ. It is easy to see by inspection of the form of A_{k+1} that the only occurrences of q_k in A_{k+1} have the form $q_k \circ B$ or the form $B \circ q_k$, where $\circ$ is either $\oplus$, $\&$, or $\otimes$. Therefore, again by Lemma 22, $\vdash q_k^\perp, \Gamma$ is not provable.

Thus it follows that $k \not< m$.

When $k = m$, we can apply Lemma 22 to infer that $\Gamma \equiv q_m$, since otherwise, Γ would not contain any occurrences of q_m as immediate arguments to $\otimes$, $\&$ or $\oplus$. If $Q_m \equiv \exists$, this yields the deduction

$$\dfrac{\dfrac{}{\vdash q_m, q_m^\perp}I \qquad \begin{array}{c}\vdots \\ \vdash \quad \langle I\rangle, ((x_m \parr q_{m-1}) \oplus (x_m^\perp \parr q_{m-1})), \\ [\![Q_{m-1}X_{m-1}\ldots Q_0X_0 : M]\!]_g, g\end{array}}{\begin{array}{c}\vdash \quad \langle I\rangle, q_m, (q_m^\perp \otimes ((x_m \parr q_{m-1}) \oplus (x_m^\perp \parr q_{m-1})), \\ [\![Q_{m-1}X_{m-1}\ldots Q_0X_0 : M]\!]_g, g\end{array}}\otimes$$

For the same reason as before, the remaining subgoal cannot be reduced by applying the $\otimes$-rule to a formula $q_i^\perp \otimes A_i$ since all of the occurrences of q_i remain as immediate arguments to $\parr$. The only possible reduction then is to "unwind" the quantifier encoding for Q_mX_m as in the $(\Leftarrow)$ direction of the proof until q_{m-1} is introduced as a sequent formula. If the left $\oplus$ reduction is applied in the given cut-free proof, we have

$$\dfrac{\dfrac{\begin{array}{c}\vdots \\ \vdash \langle I\rangle, x_m, q_{m-1}, [\![Q_{m-1}X_{m-1}\ldots Q_0X_0 : M]\!]_g, g\end{array}}{\vdash \langle I\rangle, (x_m \parr q_{m-1}), [\![Q_{m-1}X_{m-1}\ldots Q_0X_0 : M]\!]_g, g}\parr}{\vdash \langle I\rangle, ((x_m \parr q_{m-1}) \oplus (x_m^\perp \parr q_{m-1})), [\![Q_{m-1}X_{m-1}\ldots Q_0X_0 : M]\!]_g, g}\oplus$$

Then by the induction hypothesis applied to the proof of the sequent

$\vdash \langle I\rangle, x_m, q_{m-1}, [\![Q_{m-1}X_{m-1}\ldots Q_0X_0 : M]\!]_g, g$

we get $I, \neg X_m \models Q_{m-1}X_{m-1}\ldots Q_0X_0 : M$, and hence $I \models \exists X_m Q_{m-1}X_{m-1}\ldots Q_0X_0 : M$.

The argument is similar when the right $\oplus$ reduction is applied in the given cut-free proof.

The proof when $Q_m \equiv \forall$ is also similar.

When $m = n$ in Lemma 23, it follows that a closed QBF G is valid iff $\sigma(G)$ is provable in MALL. Since σ is a log-space encoding of a given QBF, the final result below follows immediately from Theorems 23 and 11.

THEOREM 12 MALL provability is PSPACE-complete.

5.1.7 IMALL is PSPACE-Complete

With two-sided sequents, the intuitionistic fragment of MALL constrains the right-hand side of the sequent to contain at most one formula. A two-sided reformulation of the above proof could be carried out entirely within the intuitionistic fragment of MALL, showing that intuitionistic MALL is also PSPACE-complete.

COROLLARY 5.1.2 IMALL provability is PSPACE-complete.

5.2 MLL is NP-Complete

In this section it is shown that the decision problem for propositional multiplicative linear logic is NP-complete. An argument for the NP-hardness of this fragment was first sketched by Max Kanovich in electronic mail [80]. Together with the earlier result [107] that the multiplicatives are in NP, Kanovich's result showed that this decision problem is NP-complete. Kanovich later updated his argument to show that the "Horn fragment" of the multiplicatives is also NP-complete [79, 85], using a novel computational interpretation of this fragment of linear logic. This section continues this trend by providing a proof that evaluating expressions in *and*, *or*, *true*, and *false* in multiplicative linear logic is NP-complete. That is, even without propositions, multiplicative linear logic is NP-complete.

This is a remarkable result. Cook's theorem [42] shows that it is difficult (NP-complete) to guess the truth assignments in propositional classical logic. In multiplicative linear logic, even when the truth assignments are given, it is *still* just as difficult (NP-complete) to decide if a formula is provable. In fact, the idea of replacing propositions by expressions involving constants has recently been extended by Kanovich to cover other fragments of linear logic [81, 82], showing that in full linear logic even when there are no quantifiers, and no propositions, (the entire truth assignment is given), it is undecidable to determine if a formula is provable.

It should be noted that the complexity of the decision problem for the multiplicatives varies depending on the presence of structural rules. In the classical and intuitionistic cases (where additive and multiplicative connectives coincide) the problem is co-NP-complete. The multiplicative fragment of direct logic (linear logic with unrestricted weakening) is NP-complete [107], and the Lambek calculus (multiplicative linear logic without exchange) is related to context-free grammars [126].

This section begins with a proof that intuitionistic and classical Multiplicative Linear Logic are in NP. Then it is shown that IMLL is NP-hard. Then it is demonstrated that the multiplicative circuit evaluation problem (constant only multiplicative linear logic) is NP-complete.

5.2.1 imll and cmll Are In np

Informally, the argument showing membership in NP is simply that every connective in a multiplicative linear logic formula is analyzed exactly once in any cut-free proof. Thus an entire proof, if one exists, can be guessed and checked in nondeterministic polynomial time.

THEOREM 13 (SMALL-PROOFS) Every connective is analyzed exactly once in any cut-free MLL or IMLL proof.

From Theorem 2 and Theorem 13, we know that given a MLL or IMLL sequent of size n, if there is any proof of this sequent, then there is a proof with exactly n total applications of inference rules. Since each application of an inference rule may be represented in space linear in n, we may simply guess and check an entire n^2 representation of a proof tree in nondeterministic polynomial time.

5.2.2 imll is np-Hard

In this section we analyze the inherent difficulty of deciding IMLL sequents. We show that one can encode a 3-Partition problem in IMLL using unary notation. Since 3-Partition is NP-complete in the strong sense, we thus have the result that IMLL is NP-hard. We show that our encoding is correct by relying on a property of IMLL (which also holds of MLL) which greatly restricts the space of possible IMLL proof trees. This “balanced” property is related to the tremendously useful “lock and key” mechanism used in [107, 104] for MALL and IMALL.

We begin with a few definitions. We recall the definition of *polarity* of a formula from Section 2.8

$$
\begin{array}{ccc}
[A \multimap B]^{+} & = & [A]^{-} \multimap [B]^{+} \\
[A \otimes B]^{+} & = & [A]^{+} \otimes [B]^{+} \\
[\Sigma, A]^{+} & = & [\Sigma]^{+}, [A]^{+} \\
[A^{\perp}]^{+} & = & ([A]^{-})^{\perp} \\
[A \multimap B]^{-} & = & [A]^{+} \multimap [B]^{-} \\
[A \otimes B]^{-} & = & [A]^{-} \otimes [B]^{-} \\
[\Sigma, A]^{-} & = & [\Sigma]^{-}, [A]^{-} \\
[A^{\perp}]^{-} & = & ([A]^{+})^{\perp}
\end{array}
$$

The polarity of an instance of a formula A in a sequent $\Sigma \vdash \Delta$ is given by the sign of the superscript on A in $[\Sigma]^{-}$ or $[\Delta]^{+}$. That is, if an instance of formula A ends up as $[A]^{+}$, then it is of *positive* polarity. If an instance of formula A ends up as $[A]^{-}$, then it is of *negative* polarity.

We define a sequent to be *balanced* if the number of occurrences of p_i with positive polarity and negative polarity are equal. Otherwise, we say a sequent is *unbalanced*. We show that all provable IMLL and MLL sequents are balanced, and therefore all unbalanced sequents are not provable. This property was previously discussed in [79, 15], and many other places.

PROPOSITION 3 (PROOFS ARE BALANCED) An IMLL or MLL sequent is provable only if it is balanced.

Proof 24 We proceed by induction on the depth of assumed proof. In the base case, all axioms are balanced, as are the axioms for the multiplicative constants. Inductively, by case analysis, we find that if all hypotheses of an IMLL or MLL inference rule are balanced, then the conclusion must also be balanced.

Note that this property fails for other fragments of linear logic such as IMALL which include the additive connectives and constants: $\&$, $\oplus$, $\top$, and 0, and also fails to hold in the presence of exponential connectives ! and ?.

Also note that this property follows immediately from Girard's proof net conditions on MLL formulas. Axiom links in a proof net must connect each proposition with exactly one proposition of opposite polarity, and thus all provable conclusions must be balanced.

We will encode an NP-complete problem, 3-Partition, in MLL, and show that our encoding is sound and complete. The main idea is that the small-proof property of MLL allows us to encode "resource distribution" problems naturally. Since linear logic treats propositions as resources natively, it has been called "resource-consciousness" [26]. Note that since full linear logic is conservative over MLL, our encoding remains sound and complete even in larger fragments. This does not lead to new results, however, since the complexity of most larger linear logics have already been completely characterized [107].

3-Partition We use the NP-completeness of 3-Partition:
(as stated in Garey+Johnson page 224)

Instance:	Set A of $3m$ elements, a bound $B \in Z^+$, and a size $s(a) \in Z^+$ for each $a \in A$ such that $B/4 < s(a) < B/2$ and such that $\sum_{a \in A} s(a) = mB$
Question:	Can A be partitioned into m disjoint sets $A_1, A_2, \cdots, A_m$ such that, for $1 \leq i \leq m$, $\sum_{a \in A_i} s(a) = B$ (note that each A_i must therefore contain exactly 3 elements from A)?
Reference:	[Garey+Johnson, 1975].
Comment:	NP-complete in the strong sense.

We will write $S1$ for $S(a_1)$ to improve readability of the following discussion.

Encoding Given an instance of 3-Partition equipped with a set $A = \{a_1, \cdots, a_{3M}\}$, an integer B, and a unary function S, presented as a tuple $\langle A, M, B, S \rangle$, we define the encoding function θ as

$$\theta(\langle A, M, B, S \rangle) = [(k \multimap c^{S1}) \otimes \cdots \otimes (k \multimap c^{S3M})] \multimap (k^3 \multimap c^B)^M$$

Reminder of notation: k, c are propositions, $x^Y = \overbrace{x \otimes x \otimes \cdots \otimes x \otimes x}^{Y \text{ copies}}$

The claim is that this formula is provable in the multiplicative fragment of linear logic if and only if the 3-Partition problem is solvable.

Completeness This is the more difficult direction of the proof, and requires that we show that if there is a proof of the linear logic formula $\theta(\langle A, M, B, S \rangle)$, then the 3-Partition problem $\langle A, M, B, S \rangle$ has a solution. We make critical use of the proofs-are-balanced property of MLL, and use two simple permutability properties of linear logic: $\otimes$ **L** permutes down (except if principle), and $\multimap$ **R** permutes down (except if principle).

If there is a proof of $\Theta(\langle A, M, B, S\rangle)$, then there is a cut-free proof, by the cut-elimination theorem 2. Given a cut-free proof, we first permute all applications of $\otimes$L and $\multimap$ **R** down as far as they will go. We are then left with some proof of the form:

$$\dfrac{\dfrac{\dfrac{\vdots}{(k\multimap c^{S1}),\cdots,(k\multimap c^{S3m}) \vdash (k^3\multimap c^B)^M}\otimes L \quad \vdots}{(k\multimap c^{S1})\otimes\cdots\otimes(k\multimap c^{S3M}) \vdash (k^3\multimap c^B)^M}\otimes L}{\vdash (k\multimap c^{S1})\otimes\cdots\otimes(k\multimap c^{S3m})\multimap(k^3\multimap c^B)^M}\multimap R$$

Informally, every $\otimes$ **R** proof step applied to a formula $(k^3\multimap c^B)^M$ corresponds to dividing a group of elements.

Formally, we now consider in detail the part of the above proof:

$$\dfrac{\vdots}{(k\multimap c^{S1}),\cdots,(k\multimap c^{S3M}) \vdash (k^3\multimap c^B)^M}$$

We now perform complete induction on M.

We show that if $M \neq 1$, the only possible next proof step is $\otimes$ **R**. If any other proof rule is applied to this sequent, one or both of the hypotheses will be unbalanced, and therefore unprovable. We show this by cases. The only formulas which may be principal in a rule other than $\otimes$ **R** are of the form $k\multimap c^{Si}$. If the $\multimap$L rule is applied, we have the following supposed proof for some Σ and Δ with the multiset union $\Sigma \bigcup \Delta \bigcup (k\multimap c^{Si})$ being equal to the left hand side of the conclusion:

$$\dfrac{\begin{array}{c}\vdots\\ \Sigma\vdash k\end{array} \qquad \begin{array}{c}\vdots\\ \Delta, c^{Si}\vdash (k^3\multimap c^B)^M\end{array}}{(k\multimap c^{S1}),\cdots,(k\multimap c^{S3M}) \vdash (k^3\multimap c^B)^M}\multimap L$$

However, given that Σ is made up of formulas from the conclusion, $\Sigma \vdash k$ cannot be balanced. The formula k has positive polarity, and the only formulas which contain k with negative polarity occur in $(k^3\multimap c^B)^M$. Thus if $M \neq 1$, the only possible next proof step is $\otimes$ **R**.

We may then focus on the case when $M = 1$. We claim that each remaining branch in such a proof corresponds exactly to one solution partition of the original partition problem. That is, we claim that when $M = 1$, we must be left with a sequent of the form:

$(k \multimap c^{Sx}), (k \multimap c^{Sy}), (k \multimap c^{Sz}) \vdash (k^3 \multimap c^B)$

Where $Sx + Sy + Sz = B$. That is, exactly three formulas of the form $k \multimap c^{Sx}$ in the antecedent, consequent $k^3 \multimap c^B$, and the sum of the weights is equal to B, as required by 3-Partition. To show this, note that we know the consequent is $k^3 \multimap c^B$. By counting k's, there must be exactly three formulas of the form $k \multimap c^{Sx}$. By another balancing argument on c's, the sum $Sx + Sy + Sz = B$, as required by 3-Partition. Thus we have shown that each branch of the proof where $M = 1$ corresponds exactly to one partition containing three elements whose sum of weights is B.

Thus by complete induction on M, one can show that from any proof of the given sequent, one can construct a solution to the 3-Partition problem.

Soundness If the given 3-Partition problem $\langle A, M, B, S\rangle$ is solvable, then we are able to find a proof of the MLL formula $[\langle A, M, B, S\rangle]$.

The proof is straightforward. For each group of three elements in the assumed solution to the 3-Partition problem, one forms the following subproof, assuming the elements of the group are numbered x, y, and z.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{c^{Sx}, c^{Sy}, c^{Sz} \vdash c^B} \quad \dfrac{}{k \vdash k}I}{(k \multimap c^{Sx}), c^{Sy}, c^{Sz}, k \vdash c^B}\multimap L \quad \dfrac{}{k \vdash k}I}{(k \multimap c^{Sx}), (k \multimap c^{Sy}), c^{Sz}, k, k \vdash c^B}\multimap L \quad \dfrac{}{k \vdash k}I}{(k \multimap c^{Sx}), (k \multimap c^{Sy}), (k \multimap c^{Sz}), k, k, k \vdash c^B}\multimap L}{(k \multimap c^{Sx}), (k \multimap c^{Sy}), (k \multimap c^{Sz}), k^2, k \vdash c^B}\otimes L}{(k \multimap c^{Sx}), (k \multimap c^{Sy}), (k \multimap c^{Sz}), k^3 \vdash c^B}\otimes L}{(k \multimap c^{Sx}), (k \multimap c^{Sy}), (k \multimap c^{Sz}) \vdash (k^3 \multimap c^B)}\multimap R
$$

The elided proof of $c^{Sx}, c^{Sy}, c^{Sz} \vdash c^B$ is guaranteed to exist by the conditions on the solution to 3-Partition, and consists entirely of $\otimes$ **R** and **I** rules.

Given the M proofs constructed as above from each of the M groups of elements, one combines them with $\otimes$ **R** into a proof of

$(k \multimap c^{S1}), \cdots, (k \multimap c^{S3M}) \vdash (k^3 \multimap c^B)^M$

The proof can then be completed with $3M - 1$ applications of $\otimes$ **L**, and ends with one application of $\multimap$ **R**.

5.2.3 Constant-Only Case

Some time ago, Girard developed a necessary condition for the provability of constant multiplicative linear expressions:

$$\begin{array}{rcl} M(1) & = & 1 \\ M(\perp) & = & 0 \\ M(A \mathbin{⅋} B) & = & M(A) + M(B) \\ M(A \otimes B) & = & M(A) + M(B) - 1 \end{array}$$

If a formula A is provable in multiplicative linear logic and contains no propositions, then $M(A) = 1$. In other words, the number of tensors is one less than the number of ones in any provable constant only MLL (COMLL) formula. Avron (and others) have studied generalizations of this "semantic" measure to include propositions (where a proposition p is given value 1, and $p^\perp$ is given value 0) yielding a necessary condition for MLL provability. One may go even further, achieving a necessary condition for MALL provability, using min for $\&$ and max for $\oplus$, and plus and minus infinity for the additive constants. For the latter case, the condition becomes if a formula A is provable in MALL, then $M(A) \geq 1$. Also, one may generalize these conditions somewhat, replacing all instances of 1 with any arbitrary constant c, and allowing propositions to have different (although fixed) values, where p has value v_p, and $p^\perp$ has value $c - v_p$ [15].

Although the above condition is necessary, there has been a question as to whether some form of simple "truth table" or numerical evaluation function like the above could yield a necessary and sufficient condition for provability of constant multiplicative (COMLL) expressions. Lincoln and Winkler showed that even this multiplicative constant evaluation or circuit evaluation problem is NP-complete.

The main idea stems from the observation that there are many infinite families of nonequivalent constant-only formulas. One such family is built from only $\otimes$ and $\perp$. For any natural numbers n, m, $\perp^n \vdash \perp^m$ if and only if $n = m$.

We will encode an NP-complete problem, 3-Partition, in MLL, and show that our encoding is sound and complete. The main idea is that the small-proof property of MLL allows us to encode "resource distribution" problems naturally. Since linear logic treats propositions as resources natively, it has been called "resource-consciousness" [26]. Note that since full linear logic is conservative over MLL, our encoding remains sound and complete even in larger fragments. This does not lead to new results, however, since the complexity of most larger linear logics have already been completely characterized [107].

Encoding Given an instance of 3-Partition (see 5.2.2) equipped with a set $A = \{a_1, \cdots, a_{3M}\}$, an integer B, and a unary function S, presented as a tuple $\langle A, M, B, S\rangle$, we define the encoding function θ as $\theta(\langle A, M, B, S\rangle) =$

$$[(\bot \multimap \bot^{S1}) \otimes \cdots \otimes (\bot \multimap \bot^{S3M})] \multimap (\bot^3 \multimap \bot^B)^M$$

Notation: $x^Y = \overbrace{x \otimes x \otimes \cdots \otimes x \otimes x}^{Y \text{ copies}}$ $\qquad x^{\langle Y\rangle} = \overbrace{x \parr x \parr \cdots \parr x \parr x}^{Y \text{ copies}}$

Using the contrapositive ($A \multimap B \quad \equiv \quad B^\bot \multimap A^\bot$), we can develop a "1 only" encoding:

$$[(1^{\langle S1\rangle} \multimap 1) \otimes (1^{\langle S2\rangle} \multimap 1) \otimes \cdots \otimes (1^{\langle S3M\rangle} \multimap 1)] \multimap (1^{\langle B\rangle} \multimap 1^{\langle 3\rangle})^M$$

Eliminating the linear implication in favor of $\parr$ these formulas become:

$$(1^{\langle S1\rangle} \otimes \bot) \parr (1^{\langle S2\rangle} \otimes \bot) \parr \cdots \parr (1^{\langle S3M\rangle} \otimes \bot) \parr (\bot^B \parr 1^{\langle 3\rangle})^M$$

We will use the last form of this formula, since it contains no implicit negations (linear implication). One may see this formula satisfies Girard's measure condition if there are $3 * M$ elements, and the sum of the sizes equals $M * B$, side conditions on the statement of 3-Partition (Garey+Johnson).

The claim is that these formulas are provable in the multiplicative fragment of linear logic if and only if the 3-Partition problem is solvable.

Soundness

LEMMA 24 (SOUNDNESS) If a 3-Partition problem $\langle A, M, B, S\rangle$ is solvable, then we are able to find a proof of the COMLL formula $\theta(\langle A, M, B, S\rangle)$.

Proof 25 The proof is straightforward. For each group of three elements in the assumed solution to the 3-Partition problem, one forms the following subproof, assuming the elements of the group are numbered x, y, and z.

$$
\begin{prooftree}
\AxiomC{$\vdots$}
\noLine
\UnaryInfC{$\vdash 1^{\langle Sx\rangle}, 1^{\langle Sy\rangle}, 1^{\langle Sz\rangle}, \bot^B$}
\AxiomC{}
\RightLabel{1}
\UnaryInfC{$\vdash 1$}
\RightLabel{$\bot$}
\UnaryInfC{$\vdash 1, \bot$}
\RightLabel{$\otimes$}
\BinaryInfC{$\vdash (1^{\langle Sx\rangle} \otimes \bot), 1^{\langle Sy\rangle}, 1^{\langle Sz\rangle}, 1, \bot^B$}
\AxiomC{}
\RightLabel{1}
\UnaryInfC{$\vdash 1$}
\RightLabel{$\bot$}
\UnaryInfC{$\vdash 1, \bot$}
\RightLabel{$\otimes$}
\BinaryInfC{$\vdash (1^{\langle Sx\rangle} \otimes \bot), (1^{\langle Sy\rangle} \otimes \bot), 1^{\langle Sz\rangle}, 1, 1, \bot^B$}
\AxiomC{}
\RightLabel{1}
\UnaryInfC{$\vdash 1$}
\RightLabel{$\bot$}
\UnaryInfC{$\vdash 1, \bot$}
\RightLabel{$\otimes$}
\BinaryInfC{$\vdash (1^{\langle Sx\rangle} \otimes \bot), (1^{\langle Sy\rangle} \otimes \bot), (1^{\langle Sz\rangle} \otimes \bot), 1, 1, 1, \bot^B$}
\RightLabel{$\parr$}
\UnaryInfC{$\vdash (1^{\langle Sx\rangle} \otimes \bot), (1^{\langle Sy\rangle} \otimes \bot), (1^{\langle Sz\rangle} \otimes \bot), 1^{\langle 2\rangle}, 1, \bot^B$}
\RightLabel{$\parr$}
\UnaryInfC{$\vdash (1^{\langle Sx\rangle} \otimes \bot), (1^{\langle Sy\rangle} \otimes \bot), (1^{\langle Sz\rangle} \otimes \bot), 1^{\langle 3\rangle}, \bot^B$}
\RightLabel{$\parr$}
\UnaryInfC{$\vdash (1^{\langle Sx\rangle} \otimes \bot), (1^{\langle Sy\rangle} \otimes \bot), (1^{\langle Sz\rangle} \otimes \bot), (1^{\langle 3\rangle} \parr \bot^B)$}
\end{prooftree}
$$

The elided proof of $\vdash 1^{\langle Sx\rangle}, 1^{\langle Sy\rangle}, 1^{\langle Sz\rangle}, \bot^B$ is guaranteed to exist by the conditions on the solution to 3-Partition. That is, since x, y, and z are from the same partition, the sum of Sx, Sy, and Sz must be equal to B.

Given the M proofs constructed as above from each of the M groups of elements, one combines them with $\otimes$ into a proof of

$$\vdash (1^{\langle S1\rangle} \otimes \bot), \cdots, (1^{\langle S3M\rangle} \otimes \bot), (1^3 \parr \bot^B)^M$$

The proof can then be completed with $3M$ applications of $\parr$.

Completeness

Lemma 25 (Completeness) For A, M, B, and S satisfying the constraints of 3-Partition, if there is a proof of the COMLL formula $\theta(\langle A, M, B, S\rangle)$, then the 3-Partition problem $\langle A, M, B, S\rangle$ is solvable.

Proof 26 To simplify this direction of the proof, we use the extra assumption that the "bin size" B is greater than 8. For a justification of this assumption, see Section 5.2.3. The following makes heavy use of Girard's condition on COMLL.

Assuming we have a proof of

$$\vdash (1^{\langle S1\rangle} \otimes \bot) \parr (1^{\langle S2\rangle} \otimes \bot) \parr \cdots \parr (1^{\langle S3M\rangle} \otimes \bot) \parr (\bot^B \parr 1^{\langle 3\rangle})^M$$

we show that the corresponding 3-Partition problem is solvable.

If there is a proof of this sequent, then there is a cut-free proof, by the cut-elimination theorem (Theorem 2). By repeated applications of Property 1, if there is a proof of this sequent, then there is a proof of $\vdash (1^{\langle S1\rangle} \otimes \bot), (1^{\langle S2\rangle} \otimes \bot), \cdots, (1^{\langle S3M\rangle} \otimes \bot), (\bot^B \parr 1^{\langle 3\rangle})^M$

We then perform complete induction on M.

If $M > 1$, the proof of this sequent must end in $\otimes$, since all formulas have main connective $\otimes$. We next show that the principal formula of that rule application must be $(\bot^B \parr 1^{\langle 3\rangle})^M$.

First, we note that each formula $(1^{\langle Sj\rangle} \otimes \bot)$ has measure $Sj - 1$. Since we are assuming $B > 8$, the initial conditions of the 3-Partition problem ensure that for all j, $Sj > 2$, and therefore $Sj - 1 > 1$. There is only one formula, $(\bot^B \parr 1^{\langle 3\rangle})^M$, with negative measure.

If we assume that one of the $(1^{\langle Si\rangle} \otimes \bot)$ formulas is principal in an application of $\otimes$, by Girard's condition, each hypothesis sequent must have measure one. In this case we have the following supposed proof for some Σ and Δ with the multiset union $\Sigma \bigcup \Delta \bigcup (1^{\langle Si\rangle} \otimes \bot)$ being equal to the conclusion:

$$\dfrac{\begin{array}{cc} \vdots & \vdots \\ \vdash \Sigma, \bot & \vdash \Delta, 1^{\langle Si \rangle} \end{array}}{\vdash (1^{\langle S1 \rangle} \otimes \bot), (1^{\langle S2 \rangle} \otimes \bot), \cdots, (1^{\langle S3M \rangle} \otimes \bot), (\bot^B \parr 1^{\langle 3 \rangle})^M} \otimes R$$

But $1^{\langle Si \rangle}$ which occurs in one hypothesis has measure > 2. Therefore, the formula with negative measure, $(\bot^B \parr 1^{\langle 3 \rangle})^M$, must occur in Δ. Now consider the other hypothesis, which must contain $\bot$, and other formulas Σ from the conclusion sequent. If any formulas of the form $(1^{\langle Sj \rangle} \otimes \bot)$ are included in Σ, the measure of that hypothesis is greater than 1. If no such formulas are included, then the sequent has measure 0. In either case, by Girard's condition that sequent is not provable. Thus the assumption that one of the $(1^{\langle Sj \rangle} \otimes \bot)$ formulas is principal must be in error, and $(\bot^B \parr 1^{\langle 3 \rangle})^M$ must be principal.

Thus if $M > 1$, the only possible next proof step is $\otimes$, with principal formula $(\bot^B \parr 1^{\langle 3 \rangle})^M$. We may then focus on the case when $M = 1$. We claim that each remaining branch in such a proof corresponds (more or less) to one solution partition of the original partition problem. That is, we claim that when $M = 1$, we must be left with a sequent of the form:

$$\vdash (1^{\langle S1 \rangle} \otimes \bot), (1^{\langle S2 \rangle} \otimes \bot), \cdots, (1^{\langle Sn \rangle} \otimes \bot), (\bot^B \parr 1^{\langle 3 \rangle})$$

There are exactly $n + B - 1$ occurrences of $\otimes$ in this sequent, and $\sum_{1 \le i \le n} Si + 3$ ones in this sequent. By Girard's condition, $(\sum_{1 \le i \le n} Si + 3) - (n + B - 1) = 1$, or equivalently $\sum_{1 \le i \le n} Si = n + B - 3$.

If $n = 0$, we have $0 = B - 3$, which is false by our assumption that $B > 8$. If $n = 1$, we have $S1 = B - 2$, but the sizes are bounded by $B/2$, and with the assumption that $B > 8$, there is a contradiction. Also, considering cases of $n > 4$, we have $\sum_{1 \le i \le n} Si = n + B - 3$, and the assumptions that $B > 8$ and $S_i > B/4$, thus we have $n(B/4) > n + B - 3$, which implies that $fracn - 3(n/4) - 1 > B$ and from this and $B > 8$, we have $n < 5$. This leaves the $n = 2$, $n = 3$, and $n = 4$ cases.

Thus we have a proof with M branches, each of which represents either two, three, or four elements.

In the case that $n = 3$, we have $\sum_{1 \le i \le 3} Si = B$, and thus this sequent identifies a partition of the original problem meeting the requirement that the sum equal B.

Note that since there are exactly $3M$ elements, and $\sum_{a \in A} Sa = mB$, there are exactly the same number of groups with four elements as there are groups with two elements. Also note that if $n = 2$, we have by the above constraint that $S1 + S2 = B - 1$, and if $n = 4$, then $S1 + S2 + S3 + S4 = B + 1$.

Further, we may analyze by cases to show that if there are any groups of four, then $B = 4C + 3$ for some integer C. If there are any groups of four, and $B = 4C$

for some C, then the smallest allowable element is $C + 1$, since the size of each element must strictly dominate $B/4$. However, taking four elements of size $C + 1$, the constraint $S1 + S2 + S3 + S4 = B + 1$ is violated. Similarly for $B = 4C + 1$ and $B = 4C + 2$. Thus if there is a group of four elements, then $B = 4C + 3$ for some C, and by simple algebra, the elements of any group of four elements all have size $C + 1$, and the elements of any group of two elements both have size $2C + 1$. Noting that there are exactly as many groups of two as groups of four, we may rearrange the elements into groups of three by taking two elements from the group of four and one element from the group of two. Both resulting groups of three have total size $4C + 3$, which happily is equal to B.

Thus, given any proof of $\theta(\langle A, M, B, S\rangle)$, we first see that one may identify M branches, each of which is of the form $\vdash (1^{\langle S1\rangle} \otimes \perp), (1^{\langle S2\rangle} \otimes \perp), \cdots, (1^{\langle Sn\rangle} \otimes \perp), (\perp^B \parr 1^{\langle 3\rangle})$. From these M branches, we may identify M partitions of three elements of the associate 3-Partition problem, some partitions directly from branches of the proof, and some partitions from a simple "reshuffling" of groups of four and two elements. In other words, from any proof of the given sequent, one may construct a solution to the 3-partition problem.

Additional Constraints On 3-Partition There are simplifying assumptions one can make about a 3-Partition problem which will alleviate the above minor difficulties mentioned above.

One may consider only those instances of 3-Partition where $B > 8$. One may show this by cases. If $B = 0$, or $B = 1$, $B = 2$, or $B = 4$ there are no possible problem instances satisfying $B/4 < s(a) < B/2$. For $B = 5$, all elements must be equal to 2, and thus there are no possible solutions. For $B = 8$, all elements must be equal to 3, and thus there are also no possible solutions in this case. For $B = 3$, all allowable problem instances have all elements equal to 1, and thus this case is solvable in polynomial (constant) time (report "YES"). For $B = 6$, similarly, all elements have size 2, and the answer is trivially "YES." For $B = 7$, all elements have size 2 or 3, and thus all partitions must be made up of two elements of size 2 and one element of size 3. The obvious counting algorithm thus solves this case in polynomial time. Thus for all cases where B is less than or equal to 8 the 3-Partition problem is solvable in polynomial time, and thus 3-Partition remains NP-complete with the further constraint that $B > 8$.

One may also consider the following looser specification of 3-Partition, which we will call 3-Partition'.

Instance:	Set A' of $3m$ elements, a bound $B' \in Z^+$, and a size $s(a') \in Z^+$ for each $a' \in A'$	
Question:	Can A' be partitioned into m disjoint sets $A'_1, A'_2, \cdots, A'_m$ such that, for $1 \leq i \leq m$, $\sum_{a' \in A'_i} s(a') = B'$ such that each set contains exactly 3 elements from A'?	3-Partition'

does not have a priori restrictions on the sizes of elements, but instead has an explicit specification that only partitions of three elements are allowed. One can immediately restrict $s(a')$ for all $a' \in A'$ to be $\leq B'$, for otherwise there is no solution, since all sizes are nonnegative.

From an instance of 3-Partition', one may generate an instance of 3-Partition by adding $B' + 1$ to the size of each element of A'. The instance of 3-Partition then is asked with $B = 4B' + 3$, and the size of each element satisfies the condition $B/4 \leq s(a) \leq B/2$, since $B = 4B' + 3$, $s(a') < B'$, and $s(a) = s(a') + B' + 1$. By adding more than $B' + 1$ to the size each element, one can create instances of 3-Partition where elements are as close to $B/3$ as desired. Thus one could avoid the complications involved in "reshuffling" the groups of four and two elements above by restricting the 3-Partition problem accordingly.

Another type of simplification can be achieved with other encodings of a 3-Partition problem. Consider the earlier encoding of 3-Partition in full multiplicative linear logic:

$$[(k \multimap c^{S(A_1)}) \otimes \cdots \otimes (k \multimap c^{S(A_{3M})}) \otimes (c^B \multimap j)^M] \multimap (k^3 \multimap j)^M$$

Constant-only encodings can be generated by replacing c by bottom, and k by $k^{\langle C \rangle}$ for some integer C. A value of C that is particularly interesting is $C = \sum_{a \in A} s(a)$. Although they are still polynomial, such encodings tend to be larger than the one advocated above, and result in somewhat less complicated proofs of soundness. The case of $C = 1$ is an incorrect encoding, and one may consider the "bottom only" encoding proved sound and complete above to be generated from the case $C = 0$.

5.2.4 comll is NP-Complete

From the preceding, we immediately achieve the main result of this chapter.

THEOREM 14 (COMLL NP-COMPLETE) The decision problem for constant-only multiplicative linear logic is np-complete.

Also, with an easy conservativity result, we find that this NP-Hardness proof suffices for multiplicative linear logic as well.

THEOREM 15 (CONSERVATIVITY) Multiplicative linear logic is conservative over constant-only multiplicative linear logic.

Proof 27 By induction on cut-free MLL proofs.

5.3 Larger Constant-Only Fragments

Kanovich has extended the results regarding constant-only multiplicative linear logic to other fragments. Using a unique encoding of propositions as constant-only expressions, he has obtained the surprising result that constant-only full linear logic is undecidable [82]. He also shows that constant-only multiplicative-additive linear logic is PSPACE-complete, just as its propositional variant is. The main idea stems from the observation above that there are many infinite families of nonequivalent constant-only formulas. Kanovich's results require more properties to hold of the family of formulas he uses, thus the $\perp^n$ family used above is not sufficient for his purposes. Large and complicated formulas are built up, and used to replace atomic propositions in propositional formulas. A general representation theorem is used to show that a propositional linear logic sequent is provable if and only if the sequent is provable when all propositions are replaced with formulas from this family. This result, along with the polynomial size of the encoding, immediately yields the undecidability of full constant-only linear logic and the PSPACE-completeness of constant-only MALL.

5.4 Summary of Chapter

In this chapter we have considered decision problems for several fragments of linear logic. Perhaps the highlight of this chapter is the NP-completeness of the constant-only fragment of multiplicative linear logic. This result follows Kanovich's results [85] on the NP-completeness of multiplicative linear logic, but sharpens the result to fragments without propositions. The hardness proof relies on subtle aspects of 3-Partition, an NP-complete problem. The other main result presented earlier in this chapter is that MALL is PSPACE complete.

6 Quantified Intuitionistic and Linear Logic

The quantifier rules of linear logic are the same as for most standard presentations of logic.

$$\exists\ \mathbf{R}\ \frac{\Gamma \vdash A\{t/x\}, \Sigma}{\Gamma \vdash \exists x.A, \Sigma} \qquad \frac{\Gamma \vdash A\{y/x\}, \Sigma}{\Gamma \vdash \forall x.A, \Sigma}\ \forall\ \mathbf{R}$$

$$\forall\ \mathbf{L}\ \frac{\Gamma, A\{t/x\} \vdash \Sigma}{\Gamma, \forall x.A \vdash \Sigma} \qquad \frac{\Gamma, A\{y/x\} \vdash \Sigma}{\Gamma, \exists x.A \vdash \Sigma}\ \exists\ \mathbf{L}$$

In the $\exists$ **R** and $\forall$ **L** rules, substituting an arbitrary term t for the free occurrences of x in A is denoted $A\{t/x\}$, where the bound variables in A are renamed to avoid any clashes. The $\forall$ **R** and $\exists$ **L** rules are only applicable if the variable y is not free in Γ, Σ, and any nonlogical theory axioms. As for the other connectives of linear logic, there are right and left versions of the $\exists$ and $\forall$ rules. One may take these rules to be first order or to be second order (propositional) or even higher order.

Some properties of propositional linear logic fail to extend to first order or higher order systems, but the most important properties of cut-elimination and consistency continue to hold [57].

6.1 First-Order Intuitionistic Logic

Since full first-order linear logic contains and is conservative over the propositional fragment, it is undecidable by virtue of Theorem 4. However, full first-order linear logic was known to be undecidable from the outset due to Girard's A^0 translation of first-order intuitionistic logic [57]. First-order intuitionistic logic (see Section 2.3) is translated as follows.

$$\begin{array}{rcll}
[A \Rightarrow B]^0 & \triangleq & ![A]^0 \multimap [B]^0 & \\
[A \wedge B]^0 & \triangleq & [A]^0 \,\&\, [B]^0 & \\
[A \vee B]^0 & \triangleq & ![A]^0 \oplus ![B]^0 & \\
[\exists x.A]^0 & \triangleq & \exists x.![A]^0 & \\
[\forall x.A]^0 & \triangleq & \forall x.![A]^0 & \\
[\neg A]^0 & \triangleq & (![A]^0) \multimap 0 & \\
[P]^0 & \triangleq & P & \text{Where } P \text{ is atomic}
\end{array}$$

Girard also presents an alternate translation, called A^*:

$$
\begin{aligned}
[A \Rightarrow B]^* &\triangleq\ !([A]^* \multimap [B]^*) \\
[A \wedge B]^* &\triangleq [A]^* \otimes [B]^* \\
[A \vee B]^* &\triangleq [A]^* \oplus [B]^* \\
[\exists x.A]^* &\triangleq \exists x.[A]^* \\
[\forall x.A]^* &\triangleq\ !\forall x.[A]^* \\
[\neg A]^* &\triangleq\ !([A]^* \multimap 0) \\
[P]^* &\triangleq\ !P \quad \text{Where } P \text{ is atomic}
\end{aligned}
$$

The A^0 translation is the most commonly cited, and formed a large part of the original impetus for Girard's development of linear logic.

THEOREM 16 (GIRARD) A sequent $\vdash \Gamma$ is provable in intuitionistic logic if and only if $\vdash [\Gamma]^0$ is provable in linear logic.

THEOREM 17 (GIRARD) A sequent $\vdash \Gamma$ is provable in intuitionistic logic if and only if $\vdash [\Gamma]^*$ is provable in linear logic.

The A^0 encoding exhibbits quite beautiful properties, and shows that intuitionistic implication is not necessarily an atomic primitive operation. Girard's embedding works at many levels of proof theory: formulas, proofs, and cut-elimination steps. Furthermore, this embedding extends naturally to second-order logic and can be restricted to propositional logic. In addition, using double negation one can embed classical logic in intuitionistic logic, and using the above translations then into linear logic. In any case, full first-order intuitionistic logic is undecidable, and thus so is full first-order linear logic.

6.2 Propostional Intuitionistic Logic

The complexity results of Chapter 5 raise the possibility of a modal-free translation for propositional LJ. Statman [138] has shown that propositional intuitionistic logic, as well as its purely implicational fragment, is PSPACE-complete. Hence a natural question arises whether (beyond an immediate Turing reduction) there exists a "logical" embedding of propositional intuitionistic logic into linear logic that does not rely on the modalities as the A^0 and A^* translations do.

There are two immediate obstructions to such an encoding. One cannot hope to have such an embedding which can extend to the predicate (first-order) case, since first-order multiplicative additive linear logic is decidable (see Section 6.3 immediately below), and first-order intuitionistic logic is undecidable (this is an immediate corollary of the negative interpretation of classical logic in intuitionistic logic and the undecidability of classical first-order logic, both of which may be found,*e.g.,* in [89]). Therefore it is impossible to obtain the desired embedding for first-order quantifiers. The second obstruction is the discrepancy in complexity on the level of cut-elimination (proof normalization). Already for the purely implicational fragment of propositional intuitionistic logic, cut-elimination is hyperexponential. In contrast, cut-elimination for MALL is known to be of much lower complexity, at most exponential. In fact, this is true not just in the propositional case, but also for first-order and for second-order MALL. The required bounds are given by the Small Normalization Theorem in [57]. Hence a translation that preserves normalization of proofs with cut would have to be hyperexponential. (However, it may be possible to use an optimized presentation of intuitionistic logic such as [76] to give a triple exponential translation that preserves normalization of optimized proofs with cut.)

These results do leave open the possibility of an efficient syntactic translation of propositional intuitionistic logic into MALL so that such a translation does preserve *cut-free* proofs of a certain optimized form. Such a translation was constructed in [104], and is an "asymmetrical interpretation," that is, positive occurrences of formulas are translated differently from negative occurrences [64]. It can therefore only be viewed as a translation on cut-free proofs, unlike Girard's symmetric translations A^0 and A^* of intuitionistic logic into linear logic.

$$\begin{array}{rcl}
[A \supset B]^+ & \triangleq & k \otimes ([A]^- \multimap (k \multimap [B]^+)) \\
{[p_i]^+} & \triangleq & k \otimes ((p_i \oplus b) \otimes \top) \\
\\
{[p_i]^-} & \triangleq & p_i \\
{[p_i \supset A]^-} & \triangleq & k \multimap (((k \multimap [p_i]^+) \multimap (k \otimes b)) \oplus (k \otimes [A]^-)) \\
{[(A \supset B) \supset C]^-} & \triangleq & k \multimap ((([(B \supset C)]^- \multimap (k \multimap [(A \supset B)]^+)) \multimap (k \otimes b)) \\
 & & \quad \oplus (k \otimes [C]^-))
\end{array}$$

This translation is then extended to sequents as follows.

$$\theta(\Gamma \vdash C) \triangleq [\Gamma]^-, k \vdash [C]^+$$

Here $[\Gamma]^-$ stands for the result of the application of $[\,]^-$ to each element of Γ. Note that the "key" k is present in the context of the encoding of a sequent. The notations $[\,]^+$ and $[\,]^-$ suggest the interpretation of positive and negative polarity of occurrences. The main idea underlying this translation is the use of "locks and keys" as in Section 5.1 to bind together the multiple linear connectives which translate intuitionistic implication. This translation uses an intermediate reformulation of propositional intuitionistic logic without contraction suggested by Vorobjev and Pliushkevichus [148, 128].

Unfortunately this linear translation is exponential in the depth of the input LJ formula. However, using depth reduction techniques which date back to [154] (see also [121, 104]), the depth can be reduced to a constant (three) and thus the translation becomes polynomial. In this case the translation provides an alternative demonstration of the PSPACE-completeness of MALL.

6.3 First-Order MALL

The above θ translation of propositional LJ doesn't extend to the first order-case, and Girard's A^0 and A^* translations freely use the modal !. Thus the question of decidability of first-order MALL is left open. A priori, there is no way to bound the size of instantiations of quantifiers. Techniques such as Skolemization[1] from classical logic do not directly apply. Consider the formula $(\exists x.q^{(}x) \parr p^{)} \parr ((\forall y.q(y)) \otimes p)$ This forula is not provable in first-order MALL, but its Skolemized form $p^{\parr} q^{(}v), q(c) \otimes p$ is provable. Thus Skolemization is unsound in first-order MALL. This difficulty can be traced to the lack of certain permutabilities in linear logic (see Section 2.9) and the lack of a prenex form for first-order fragments of linear logic.

However, dynamic forms of Skolemization can be applied to linear logic. The naive form of dynamic Skolemization has been studied by a number of authors. Lincoln and Shankar [108] and Cerrito [38] have developed such systems for linear logic, based in part on work in other nonclassical logics [136]. Lincoln and Shankar [108] present a series of proof search procedures for linear logic culminating in one where the permutability properties of linear logic are exploited to introduce an optimized form of dynamic Skolemization

6.3.1 Proof Search

The relevant ideas underlying first-order MALL proof search can be illustrated with examples. Consider the example of the linear sequent $\vdash (p^\perp \parr q^\perp) \parr (q \otimes p)$. This sequent has only one possible cut-free proof.

[1] The use of Herbrand functions [71] in proof search is colloquially called Skolemization.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash p, p}\,Id \qquad \dfrac{}{\vdash q, q}\,Id}{\vdash p, q, q \otimes p}\,\otimes}{\vdash p \parr q, q \otimes p}\,\parr}{\vdash (p \parr q) \parr (q \otimes p)}\,\parr$$

By the subformula property of cut-free proofs, the $\parr$ step must be the lowest one in the proof. For the next step of the search, we have a choice between the $\parr$ rule and the $\otimes$ rule. It is easy to see that an application of the $\otimes$ rule at this point causes the proof search to fail, whereas the search succeeds with the application of the $\parr$ rule below the $\otimes$ rule. This shows that the $\otimes$ step cannot always be permuted below the $\parr$ step.

Next, consider first-order variant of the above example mentioned above $\vdash (\exists x.q(x) \parr p), (\forall y.q(y)) \otimes p$. Considering the propositional structure of this formula we know that the $\parr$ step must occur below the $\otimes$ step. By the subformula property, the $\exists$ step must occur below the $\parr$ step and the $\forall$ step must occur above the $\otimes$ step. Putting this together in the only possible cut-free proof yields the following.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash p, p}\,Id \qquad \dfrac{\dfrac{}{\vdash q(c), q(c)}\,Id}{\vdash q(c), (\forall y.q(y))}\,\forall}{\vdash p, q(c), (\forall y.q(y)) \otimes p}\,\otimes}{\vdash p \parr q(c), (\forall y.q(y)) \otimes p}\,\parr}{\vdash (\exists x.p \parr q(x)), (\forall y.q(y)) \otimes p}\,\exists$$

This is not a correct proof because the aplication of $\forall$ violates the eigenvariable condition that c must not appear free in the conclusion. As noted above, the Skolemized form of the conclusion is $\vdash p \parr q(v), q(c) \otimes p$, which is easily proved in propositional linear logic when c instantiates v. With dynamic Skolemization where quantifiers are replaced by Herbrand variables or functions *during* the proof search, the proof search rightly fails, as shown below.

$$\dfrac{\dfrac{}{\vdash p{,}p}\,Id \qquad \dfrac{\dfrac{\text{unify fails}}{\vdash q^{(}v), q(h(v))}}{\vdash q^{(}v), (\forall y.q(y))}\,\forall}{\dfrac{\vdash p{,}q(v){,}(\forall y.q(y)) \otimes p}{\dfrac{\vdash p^{\parr} q^{(}v), (\forall y.q(y)) \otimes p}{\vdash (\exists x.p^{\parr} q^{(}x)), (\forall y.q(y)) \otimes p}\,\exists}\,\parr}\,\otimes$$

Note that the Herbrand variable v *dominates* the Herbrand function h since v is introduced below h in the search. The occurs check of unification fails, indicating a potential transgression of the eigenvariable condition. We can also successfully apply dynamic Skolemization to the positive example, $\vdash p^{\parr}(\exists x.q^{(}x)), (\forall y.q(y) \otimes p)$.

$$\dfrac{\dfrac{}{\vdash p{,}p}\,Id \qquad \dfrac{\dfrac{\{v \leftarrow c\}}{\vdash q^{(}v), q(c)}}{\vdash (\exists x.q^{(}x)), q(c)}\,\exists}{\dfrac{\vdash p{,}(\exists x.q^{(}x)), q(c) \otimes p}{\dfrac{\vdash p{,}(\exists x.q^{(}x)), (\forall y.q(y) \otimes p)}{\vdash p^{\parr}(\exists x.q^{(}x)), (\forall y.q(y) \otimes p)}\,\parr}\,\forall}\,\otimes$$

The formal proof system used to perform these proofs below is called LLV.

Using LLV, first-order MALL can be shown to be decidable in NEXPTIME. This follows from the fact that any cut-free LLV proof has a depth that is bounded by the number of connectives in the conclusion. This means that the number of axiom nodes in such a proof is bounded by an exponential in the size of the conclusion. Since LLV introduces a Herbrand term of the form $h(T)$ for each universal quantifier in the conclusion, where T is itself bounded by the number of existential quantifiers in the conclusion, each axiom sequent is at most quadratic in the size of conclusion. Since unification is linear [113], one can build a nondeterministic procedure to guess and check a proof in time that is exponential in the size of the conclusion.

THEOREM 18 First-order MALL is NEXPTIME-hard.

The multiplicative fragment of first-order linear logic can be shown to be decidable in NP by the same method. The number of axiom nodes in a LLV proof of a first-order MLL formula is linear in the size of the conclusion, and each axiom node in the search has size at most quadratic in the size of the conclusion. This results in a tight bound since the propositional fragment MLL is NP-complete (see Section 5.2, [85]).

$$\mathbf{I} \qquad \frac{}{{}^{mgu(A,B,U)}_{U} \vdash_T ?\Delta, A, B^{\perp}}$$

$$\otimes \qquad \frac{{}^{W}_{U} \vdash_T ?\Delta, \Sigma, A \qquad {}^{V}_{W} \vdash_T ?\Delta, B, \Gamma}{{}^{V}_{U} \vdash_T ?\Delta, \Sigma, (A \otimes B), \Gamma}$$

$$\parr \qquad \frac{{}^{V}_{U} \vdash_T \Sigma, A, B}{{}^{V}_{U} \vdash_T \Sigma, (A \parr B)}$$

$$\oplus \qquad \frac{{}^{V}_{U} \vdash_T \Sigma, A}{{}^{V}_{U} \vdash_T \Sigma, (A \oplus B)}$$

$$\oplus \qquad \frac{{}^{V}_{U} \vdash_T \Sigma, B}{{}^{V}_{U} \vdash_T \Sigma, (A \oplus B)}$$

$$\& \qquad \frac{{}^{W}_{U} \vdash_T \Sigma, A \qquad {}^{V}_{W} \vdash_T \Sigma, B}{{}^{V}_{U} \vdash_T \Sigma, (A \,\&\, B)}$$

$$?D \qquad \frac{{}^{V}_{U} \vdash_T \Sigma, ?A, A}{{}^{V}_{U} \vdash_T \Sigma, ?A}$$

$$!S \qquad \frac{{}^{V}_{U} \vdash_T ?\Sigma, A}{{}^{V}_{U} \vdash_T ?\Sigma, !A}$$

$$\perp \qquad \frac{{}^{V}_{U} \vdash_T \Sigma}{{}^{V}_{U} \vdash_T \Sigma, \perp}$$

$$1 \qquad \frac{}{{}^{U}_{U} \vdash_T 1}$$

$$\top \qquad \frac{}{{}^{U}_{U} \vdash_T \Sigma, \top}$$

$$\forall \qquad \frac{{}^{V}_{U} \vdash_T \Sigma, A\{h(T)/y\}}{{}^{V}_{U} \vdash_T \Sigma, \forall y.A}$$

$$\exists \qquad \frac{{}^{V}_{U} \vdash_{T \cup \{v\}} \Sigma, A\{v/y\}}{{}^{V}_{U} \vdash_T \Sigma, \exists y.A}$$

Figure 6.1
Sequent Rules for MALL1

THEOREM 19 First-order MLL is NP-complete.

6.3.2 Hardness

Since propositonal MALL is PSPACE-complete, and the above result shows only that MALL1 is in NEXPTIME, this leaves open the exact complexity of MALL1. Lincoln and Scedrov [103] developed a direct encoding of nondeterministic exponential time Turing machines by MALL1 formulas. This encoding is reminiscent of the standard proof of the PSPACE-hardness of quantified boolean formula validity [139, 74]. The encoding is also related to the logic programming simulation of Turing machines given in [137].

The main idea of the encoding given in [103] is the use of additive connectives to create an exponential number of leaves, and the use of first-order quantifiers and multiplicative connectives to encode the state of a Turing machine (or perhaps more precisely a two-stack machine). Thus in the full proof tree, one reads the complete Turing machine computation across the top of the proof, with all the action happening at the leaves. The machinery utilized throughout the remainder of the proof tree exists to create an exponential number of copies of the instruction set, and provide the glue to patch together the computation.

The soundness and completeness of the encoding yeilds the following lemma.

LEMMA 26 Provability in MALL1 is NEXPTIME-hard.

Together with the earlier results of proof search for this fragment, one achieves the following theorem.

THEOREM 20 First-order MALL is NEXPTIME-complete.

6.4 Second-Order Linear Logic

Second-order linear logic is formed from propositional linear logic by adding second-order propositional quantifiers and the the following proof rules.

$$\exists 2\ \mathbf{R}\quad \frac{\Gamma \vdash A\{t/x\}, \Sigma}{\Gamma \vdash \exists x.A, \Sigma} \qquad \frac{\Gamma \vdash A\{y/x\}, \Sigma}{\Gamma \vdash \forall x.A, \Sigma}\quad \forall 2\ \mathbf{R}$$

Restrictions similar to those for the first-order quantifiers apply to uses of the universal rule. Example formulas of second order linear logic include $\forall X.X$ which behaves very much like the additive constant 0, and $\exists Y.Y$ which is related to the additive constant $\top$. The first of these is not provable in linear logic, and the second is, by instantiation of $A \multimap A$ for Y.

In first-order MALL there is an immediate depth bound on cut-free proofs, although only with difficulty can one find a bound on the size of sequents. In second-order MALL there isn't any bound on the size of sequents, nor is there an effective bound on cut-free proof depth. In fact, Lincoln, Scedrov, and Shankar showed that IMALL2 and IMLL2 are undecidable [105].

The main idea of the proof of undecidability is the encoding of second-order intuitionistic logic (LJ2). Loeb and Gabbay independently showed that LJ2 is undecidable [111, 49]. By providing second-order formulas which mimic the structural rules of contraction and weakening, one can embed LJ2 into IMALL2 or IMLL2 directly.

One first defines the IMLL2 formulas C and W as follows.

$$C \triangleq \forall X.X \multimap (X \otimes X)$$

$$W \triangleq \forall X.X \multimap 1$$

The idea is that C represents contraction and W represents weakening. The encoding is then given by the following.

$$\begin{array}{rcl}
[True]^2 & \triangleq & 1 \\
[H]^2 & \triangleq & H \\
[A \Rightarrow B]^2 & \triangleq & [A]^2 \multimap [B]^2 \\
[A \wedge B]^2 & \triangleq & [A]^2 \otimes [B]^2 \\
[\forall X.A]^2 & \triangleq & \forall X.[A]^2 \\
[A_1, \ldots, A_n]^2 & \triangleq & [A_1]^2, \ldots, [A_n]^2 \\
[\Sigma \vdash A]^2 & \triangleq & C, C, C, W, [\Sigma]^2 \vdash [A]^2
\end{array}$$

The encoding $[\ ]^2$ exhibbits the following properties.

LEMMA 27 If $\Sigma \vdash A$ in LJ2, then $[\Sigma \vdash A]^2$ in IMLL2.

LEMMA 28 If $[\Sigma] \vdash [A]$ in IMALL2, then $\Sigma \vdash A$ in LJ2.

The IMLL2 formulas C and W are the image of provable LJ2 formulas $C' = \forall X.(X \Rightarrow X \wedge X)$ and $W' = \forall X.(X \Rightarrow True)$. That is, $[C'] = C$, $[W'] = W$, and both $\vdash C'$ and $\vdash W'$ are provable in LJ2. One then obtains the following result.

Lemma 29 If $C, C, C, W, [\Sigma] \vdash [A]$ in IMALL2, then $\Sigma \vdash A$ in LJ2.

Proof 28 By Lemma 28, given a proof of $C, C, C, W, [\Sigma] \vdash [A]$ in IMALL2, one can obtain a proof of $C', C', C', W', \Sigma \vdash A$ in LJ2. With cut and the proofs of $\vdash C'$ and $\vdash W'$, one can construct the LJ2 proof of $\Sigma \vdash A$ given below.

$$\dfrac{\begin{matrix}\vdots\\ \vdash W'\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\\ \vdash C'\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\\ \vdash C'\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\\ \vdash C'\end{matrix} \qquad \begin{matrix}\vdots\\ C', C', C', W', \Sigma \vdash A\end{matrix}}{C', C', W', \Sigma \vdash A}{\scriptstyle \mathrm{Cut}}}{C', W', \Sigma \vdash A}{\scriptstyle \mathrm{Cut}}}{W', \Sigma \vdash A}{\scriptstyle \mathrm{Cut}}}{\Sigma \vdash A}{\scriptstyle \mathrm{Cut}}$$

Thus these fragments are undecidable.

Theorem 21 IMLL2 and IMALL2 are undecidable.

This encoding of LJ2 is quite efficient at the levels of formula and proof, but is somewhat problematic at the level of cut normalization (lambda reduction). The standard sequent calculus cut-elimination procedure is extremely fast (polynomial) for IMLL and IMLL2 proofs. Cut normalization in propositional or second-order intuitionistic logic is equivalent to lambda reduction in the simply or polymorphically typed lambda calculus, which has much greater computational complexity. Thus there is no possibility for the standard cut-elimination procedures to correspond nicely in all cases. Using the translation from LJ2 to IMLL2 at the level of proofs, then using standard IMLL2 cut elimination [57], then translating back to LJ2 results in a nearly-normal LJ2 proof of the same sequent. The resulting proof has only three uses of cut: one on W' and two on C', and need make no use of contraction nor weakening structural rules of LJ2. This extremely effecient (polynomial time) process generates a proof that is not the same one that would be obtained by the standard LJ2 cut-elimination procedure.

6.4.1 Extensions to mall2 and mll2

Lafont showed that the encoding of contraction and weakening as second-order formulas carries over to the multiple-conclusion (classical) linear logic case MALL2, and Lafont and Scedrov refined that argument to apply to the smaller fragment MLL2 [96, 97]. Both of these arguments are based on the direct embedding of Minsky machines (see Chapter 4), where the repeated use of machine transitions is encoded using an idea very similar to the second-order formula C' above. Their semantic treatment of second-order quantifiers overcomes the general lack of proof-theoretic tools for higher-order logic. In retrospect, their semantic proof can then be applied even in the propositional case, as sketched in Section 3.4.

6.5 First- and Second-Order Linear Logic

Amoit obtained undecidability for the combination of first- and second-order MALL with certain function and predicate symbols. His proof uses a reduction from an undecidable matching problem, but relies on the presence of both first- and second-order quantified formulas.

6.6 Second-Order Circular Logic

Emms has developed a proof that the polymorphic lambek calculus is undecidable [47]. This result is based on the ideas of Section 6.4 and the idea that one can also encode the structural rule of exchange using second-order quantification, and thus one can obtain a direct embedding of LJ2 in a noncommutative intuitionistic logic. This technique can be extended to second-order multiplicative circular logic, and also demonstrates the undecidability of larger fragments containing the additive connectives. Kanovich has shown that in the same way that Lafont lifted the results of Section 6.4 from the intuitionistic setting to the classical setting in (commutative) linear logic, one can lift Emms result above about the noncommutative intuitionistic setting to the commutative intuitionistic setting [84].

6.7 Summary of Chapter

In this chapter several proof-theoretic relationships are drawn between linear and intuitionistic logic: propositional LJ is embedded into MALL, and LJ2 is embedded in IMALL2 and IMLL2. In addition, first-order MALL is shown to be EXPTIME- complete using a novel encoding of bounded Turing machine computation and dynamic Skolemization. Several other related results are referenced.

7 Natural Deduction

If age or weaknes doe prohibyte bloudletting, you must use boxing
PHILLIP BARROUGH

The Methode of Phisicke, 1583

Systems of natural deduction were independently invented by Jaskowski [77] and Gentzen [54]. Perhaps the best exposition of natural deduction is Prawitz [131]. Advocates of natural deduction claim that there is a similarity between natural deduction and intuitive human reasoning. Natural deduction systems tend to dispense with some of the beurocracy of sequent calculi having to do with permutabilities (see Section 2.9) or commutative reductions (see Section 2.6.1). Natural deduction systems also present the most direct connection between proof theory and functional computer programming languages through the Curry-Howard isomorphism.

Gentzen developed the natural deduction sytems for classical and intuitionistic logic presented below, along with his sequent-caluclus presentations LK (Figure 2.1) and LJ (Figure 2.2). However, it is not immediately apparent how best to extend these natural deduction systems to other nonstandard logics such as linear logic. Girard solved this problem for linear logic by providing proof nets, which have proven fruitful in the study of certain computational models.

7.1 Natural Deduction for Intuitionistic Logic: NJ

The standard natural deduction system for propositional intuitionistic logic follows Gentzen's original presentation [54]. In Section 7.5 we present Gentzen's sequent-style natural deduction, which was developed later [55].

In natural deduction a proof is a finite tree with the leaves labeled by formulas and internal nodes labeled by instances of natural deduction proof rules with their antecedent and consequent formulas. A leaf of the proof tree may be *active* or *discharged*. An active leaf is a hypothesis or assumption. A discharged leaf is a *closed* assumption. A completed proof of a formula A will appear as some tree with A at the root and all leafs discharged. The following is a very simple proof tree from active assumption A that reaches the conclusion A.

$$A$$

When an assumption A is closed, it is boxed and written as $[A]$. The rule for arrow introduction closes one assumption and introduces an arrow formula, as in the following completed proof of $A \Rightarrow A$.

$$\mathbf{I} \qquad A$$

$$\wedge\mathbf{I} \qquad \frac{\begin{matrix}\vdots & \vdots\\ A & B\end{matrix}}{A \wedge B} \qquad\qquad \frac{\begin{matrix}\vdots\\ A \wedge B\end{matrix}}{A} \qquad \frac{\begin{matrix}\vdots\\ A \wedge B\end{matrix}}{B} \qquad \wedge\mathbf{E}$$

$$\Rightarrow\mathbf{I} \qquad \frac{\begin{matrix}[A]\\ \vdots\\ B\end{matrix}}{A \Rightarrow B} \qquad\qquad \frac{\begin{matrix}\vdots & \vdots\\ A \Rightarrow B & A\end{matrix}}{B} \qquad \Rightarrow\mathbf{E}$$

$$\vee\mathbf{I} \qquad \frac{\begin{matrix}\vdots\\ A\end{matrix}}{A \vee B} \qquad \frac{\begin{matrix}\vdots\\ B\end{matrix}}{A \vee B} \qquad \frac{\begin{matrix} & [A] & [B]\\ \vdots & \vdots & \vdots\\ A \vee B & C & C\end{matrix}}{C} \qquad \vee\mathbf{E}$$

$$\bot\mathbf{I} \qquad \frac{\bot}{A}$$

Figure 7.1
Natural Deduction Rules for Intuitionistic Logic (NJ)

$$\frac{[A]}{A \Rightarrow A}$$

The rules of the NJ system for intuitionistic logic are given in Figure 7.1.

The rules ending in I are considered introduction rules and the rules ending in E are considered elimination rules for obvious reasons.

Note that any assumption may be considered to appear vacuously in any proof. For example, we can discharge the assumption B that doesn't happen to actually appear in the following proof.

$$\frac{A}{B \Rightarrow A}$$

Similarly, multiple occurrences of an assumption may be closed by a single application of an inference rule. For example, the assumption A is closed in both branches of the following proof by one application of $\Rightarrow$I.

$$\dfrac{\dfrac{[A] \quad [A]}{A \wedge A}}{A \Rightarrow (A \wedge A)}$$

We can extend NJ to a first- or higher-order system by adding the rules for quantifiers. The rules for the universal quantifier are as follows.

$$\forall\mathbf{I} \quad \frac{A}{\forall x.A} \qquad \frac{\forall x.A}{A\{t/x\}} \quad \forall\mathbf{E}$$

The natural deduction system NJ (Figure 7.1) proves exactly the same formulas as LJ (Figure 2.2).

THEOREM 22 There is a natural deduction proof of A from hypotheses Γ in NJ if and only if there is a sequent calculus proof of $\Gamma \vdash A$ in LJ.

This theorem extends to the first- and higher-order cases.

It should be noted that Jaskowski [77] independently developed a system of natural deduction with different syntax. Where Gentzen uses lines to denote inference steps and marks closed assumptions by boxing them, Jaskowski uses a single large box to denote both effects. This has the desirable property of connecting inferences with the assumptions that they close, a property shared by sequent-style natural deduction systems (see Section 7.5).

7.2 Natural Deduction for Classical Logic: NK

The natural deduction system NJ is presented for intuitionistic logic. One could use a double negation translation of classical logic into intuitionistic logic, but natural deduction for intuitionistic logic can be extended to classical logic with the addition of the single inference rule of contradiction. This rule suffices for propositional, first, and higher-order systems.

$$\dfrac{\begin{array}{c}[A \Rightarrow \bot] \\ \vdots \\ \bot\end{array}}{A} \quad \text{Contradiction}$$

This rule can be read as saying that if the negation of A leads to absurdity, then A can be considered proved. There are many equivalent presentations of classical logic, such as adding all axioms of the form $A \vee (A \Rightarrow \perp)$, which is the technique used by Gentzen. The system NJ with the additional rule of contradiction is called NK.

7.3 Natural Deduction for Modal Logic: S4

Natural deduction systems have also been defined for modal logics such as S4 . The logic S4 introduces a modal operator $\Box$ that can be read as logical necessity. Linear ! is closely related to $\Box$.

$$\Box\mathbf{I}\ \frac{A}{\Box A} \qquad\qquad \frac{\Box A}{A}\ \Box\mathbf{E}$$

The rule $\Box$I comes with an additional side condition that all active assumptions are modal. Thus $\Box$I is said to be an improper inference rule.

The system S5 is defined with exactly the same rules of inference, but where the side condition on $\Box$I is weakend to allow the negations of modal formulas.

7.4 Normal Forms for Natural Deduction

Natural deduction proofs admit a particularly simple normal form. Consider the following proof.

$$\dfrac{\dfrac{\begin{matrix}\vdots \\ A\end{matrix} \quad \begin{matrix}\vdots \\ B\end{matrix}}{A \wedge B}}{A}$$

The formula $A \wedge B$ is said to be a *maximum* formula since it is larger (contains more symbols) than the formulas immediately above it and below it in the natural deduction proof. This proof can be simplified by removing the $\wedge$I and $\wedge$E inferences.

$$\begin{matrix}\vdots \\ A\end{matrix}$$

Gentzen showed that removing introduction-elimination pairs in propositional natural deduction proofs is a process that always terminates and in the case of intuitionistic logic leads to a unique normal form of a proof.

THEOREM 23 If there is a natural deduction proof of A from hypotheses Γ, then there is a normal proof of A from hypotheses Γ.

Normal proofs have the property that there exists an occurrence of a formula A in the proof such that all inferences above it are eliminations, and all inferences below it are introductions. This formula A is a minimum formula: it is the smallest formula in the thread from hypotheses to conclusion. Gentzen extended this result to the first-order case, and to propositional and predicate classical logic. In all these cases the normal form theorem can be immediately used to demonstrate the consistency of the logic in question.

THEOREM 24 The classical and intuitionistic natural deduction systems are consistent: $\perp$ is not provable in either.

In the case of second-order LJ, Gentzen left the normal form theorem for Loeb and Girard to demonstrate. The normal form theorem for LJ2 implies the consistency of number theory. This follows from an encoding of Peano arithmetic in LJ2 [131].

An immediate corollary of the normal form theorem is the subformula principle.

THEOREM 25 Every formula occurrence in an intuitionistic natural deduction proof of A from assumptions Γ is a subformula of A or of Γ.

This theorem can be extended to classical logic (with the additional rule of contradiction) where the property is weakened to one where subformulas may additionally arise from the hypothesis of contradiction or $\perp$.

7.5 Sequent-Style Natural Deduction

Natural deduction systems, especially those for modal logics, have an unfortunate lack of locality. That is, the assumptions closed by an inference rule can be distant from the application of that inference rule. This can complicate the use these systems. Thus Gentzen defined a sequent-style of natural deduction proof that records the current set of assumptions along with the conclusion at every inference step. This has the small disadvantage that some formulas are written down repeatedly, brought from one inference step to the next.

Sequent-style natural deduction can be derived by considering a sequent as a summary of an entire natural deduction proof. Thus a derivable sequent $\Gamma \vdash A$ is analogous to a natural deduction proof of A from the active assumptions in Γ. There are some choices to make in presentation, but to facilitate comparison to a related system for linear logic, a presentation based on multisets is used here.

$$\textbf{I} \qquad \frac{}{A \vdash A}$$

$$\textbf{Subst} \qquad \frac{\Sigma \vdash A \quad A, \Gamma \vdash B}{\Sigma, \Gamma \vdash B}$$

$$\textbf{Exch. Left} \qquad \frac{\Sigma, A, B, \Delta \vdash A}{\Sigma, B, A, \Delta \vdash A}$$

$$\textbf{A} \qquad \frac{\Delta \vdash (A \Rightarrow B) \qquad \Sigma \vdash A}{\Sigma, \Delta \vdash B}$$

$$\Rightarrow\textbf{R} \qquad \frac{\Sigma, A \vdash B}{\Sigma \vdash (A \Rightarrow B)}$$

$$\wedge\textbf{L} \qquad \frac{\Delta \vdash (A \wedge B) \quad \Sigma, A, B \vdash C}{\Sigma, \Delta \vdash C}$$

$$\wedge\textbf{R} \qquad \frac{\Sigma \vdash A \qquad \Gamma \vdash B}{\Sigma, \Gamma \vdash (A \wedge B)}$$

$$\textbf{WL} \qquad \frac{\Sigma \vdash B}{\Sigma, A \vdash B}$$

$$\textbf{CL} \qquad \frac{\Sigma, A, A \vdash B}{\Sigma, A \vdash B}$$

$$1\textbf{L} \qquad \frac{\Delta \vdash 1 \qquad \Sigma \vdash A}{\Sigma, \Delta \vdash A}$$

$$1\textbf{R} \qquad \frac{}{\vdash 1}$$

Figure 7.2
Sequent-Style Natural Deduction for LJ

Extensions of this system to the predicate, higher-order, and/or classical case can easily be defined. A normalization procedure analogous to that for natural deduction systems

7.6 Term Assignment – The Lambda Calculus

The Curry-Howard correspondence between natural deduction proofs and typed λ-terms holds also at the level of proof reduction and β-reduction. The correspondence can be made clear by investigating the type system for the simply-typed lambda calculus.

A type system can be created by decorating a sequent-style natural deduction system. Taking the system from Figure 7.2, one can add term annotations directly to produce the type system in Figure 7.3. Such a sequent is composed of a type context, a $\vdash$, a term, and a type. A type context is a multiset of term and type pairs. The informal meaning of a sequent is that if one assumes variables have the types given in the type context, then the term has the given type. The usual rules for lambda calculus include only I,A,Exchange,$\Rightarrow$R,WL, and CL.

7.7 Natural Deduction for Linear Logic: NL

The original presentation of linear logic by Girard [57] included a new kind of natural deduction system for linear logic called proof nets. Proof nets have similarities to Andrew's Matings [11, 10] , Bawden's connection graphs [25] , and other approaches to mechanical theorem proving for other logics [29]. Proof nets differ from standard natural deduction systems like NJ in that there are nonlocal correctness criteria, that the deduction in the end forms a graph rather than a tree, and that proof nets are able to elegantly handle the symmetric multiple-conclusion case. However, other modal logics such as S4 have natural deduction systems with nonlocal correctness criteria, different formulations of standard natural deduction can take the shape of a graph, and there exist more standard presentations of natural deduction for multiple-conclusion calculi [145]. Nonetheless, proof nets have proven fruitful in the study of linear proof theory and in the study of concurrent execution [65, 3].

A *proof structure* is a tree much like a natural deduction tree with additional links marking all the axioms. All internal nodes of the tree must be labeled by instances of proof net deduction rules with their antecedent and consequent formulas. All leaves (propositional atoms) must be linked via an axiom link to exactly one other leaf. Some, but not all proof structures are proof nets. Conditions must be enforced

I	$$\frac{}{x : A \vdash x : A}$$
Subst	$$\frac{\Sigma \vdash t : A \quad x : A, \Gamma \vdash u : B}{\Sigma, \Gamma \vdash u[t/x] : B}$$
Exch. Left	$$\frac{\Sigma, x : A, y : B, \Delta \vdash u : A}{\Sigma, y : B, x : A, \Delta \vdash u : A}$$
A	$$\frac{\Delta \vdash u : (A \Rightarrow B) \qquad \Sigma \vdash t : A}{\Sigma, \Delta \vdash (u\, t) : B}$$
$\Rightarrow$**R**	$$\frac{\Sigma, x : A \vdash t : B}{\Sigma \vdash \lambda x.t : (A \Rightarrow B)}$$
$\wedge$**L**	$$\frac{\Delta \vdash u : (A \wedge B) \quad \Sigma, x : A, y : B \vdash t : C}{\Sigma, \Delta \vdash \ \mathtt{let}\ (x \times y)\ \mathtt{be}\ u\ \mathtt{in}\ t : C}$$
$\wedge$**R**	$$\frac{\Sigma \vdash u : A \qquad \Gamma \vdash t : B}{\Sigma, \Gamma \vdash (u \times t) : (A \wedge B)}$$
WL	$$\frac{\Sigma \vdash t : B}{\Sigma, x : A \vdash t : B}$$
CL	$$\frac{\Sigma, x : A, x : A \vdash t : B}{\Sigma, x : A \vdash t : B}$$
1**L**	$$\frac{\Delta \vdash u : 1 \qquad \Sigma \vdash t : A}{\Sigma, \Delta \vdash \ \mathtt{let}\ 1\ \mathtt{be}\ u\ \mathtt{in}\ t : A}$$
1**R**	$$\frac{}{\vdash 1 : 1}$$

Figure 7.3
Simply-Typed Lambda Calculus

to weed out those proof structures that correspond to provable sequents from those that do not.

Using translations of LJ into LL such as A^0, A^*, or θ (see Chapter 6), one can use proof nets as a new kind of natural deduction system for LJ. Further, using a double negation translation of LK into LJ, one can use proof nets to reason about classical logic.

7.7.1 Proof Nets For MLL

For MLL there are only two kinds of inference rules.

$$\frac{A \quad B}{A \parr B} \qquad \frac{A \quad B}{A \otimes B}$$

Axiom links mark together exactly one positive propositional atom (p) and one negative propositional atom ($p^{\perp}$), and as mentioned above all leaves of the proof structure must be so marked.

A *proof net* is proof structure that meets certain additional conditions. The original conditions defined by Girard, the trip conditions, were somewhat inefficient, but conditions defined by Danos and Regnier [44] provide the same (optimal) efficiency as sequent-calculus proof search.

A proof structure can be mapped to a set of graphs where the nodes of the graph are all formula occurrences appearing in the proof structure. The conclusion of every $\otimes$ inference is linked to each of its immediate hypotheses, the conclusion of every $\parr$ inference is linked to one or the other, but not both, of its hypotheses. If there are n $\parr$ inferences in a proof structure there are 2^n different graphs that correspond to that structure. The Danos-Regnier condition on proof structures is that every possible such graph is a connected tree. Danos and Regnier show that any proof structure meeting this condition is a proof net, and thus can be sequentialized [57] into a sequent calculus proof [44]. The Danos-Regnier condition can be checked in polynomial time, and for any MLL formula there is only one possible tree of inferences. Thus the choice of how propositional atoms are linked provides the most significant challenge for proof search using proof nets in MLL.

7.7.2 Proof Nets For Larger Fragments

Proof nets for multiplicative linear logic (MLL) behave particularly well, not requiring boxes. However, Girard's original construction of proof nets for full propositional linear logic [57] requires boxes for additive and exponential connectives. Boxes embody improper inferences in a natural deduction system: their correctness depends on a nonlocal criteria (that no axiom links cross into or out of a box).

Several variants of proof nets have been explored that get around the requirement for boxes for the additives [26, 27, 62]. In related work Ketonen and Bellin [29] have developed a proof-net like system for direct logic [88, 87].

7.7.3 Sequentialization of Proof Nets

Proof nets, like other natural deduction systems provide an alternative to the sequent calculus. One can translate any sequent proof into a proof net and any proof net into a sequent proof. If one translates some sequent proof into a proof net and then translates that proof net back into the sequent calculus, one arrives at a sequent proof that differs only by permutable inferences from the original proof. This shows that for some purposes, permutable inferences of the sequent calculus are irrelevant beurocracy. However, the permutable inferences of linear sequent calculus do not form an impediment to any complexity or theoretical concerns of proof theory.

Proof nets have been studied from the categorical perspective, where they highlight connections to C^*-algebras, *-autonomous categories [24] and other well-studied structures [135, 134]. Blute studies proof nets and coherence theorems [36].

7.7.4 Other Natural Deduction Systems

Several variants of proof nets have been proposed. Roorda [132] studies proof nets for noncommutative linear logic (the Lambek calculus) and quantum nets. Lafont studies a computationally interesting variant called interaction nets [95].

Proof nets are not the only structure that provides a correct criteria for provability yet dispenses with the somewhat beurocratic concerns over permutable inferences as in the sequent calculus. There have been several proposals of sequent-style natural deduction systems for linear logic [102, 31, 143, 123]. These systems all come equiped with a term assignment that provides a direct connection functional computation, discussed at length in the next chapter.

The LM system displayed in Figure 7.4 and Figure 7.5 was proposed with term annotations in [102]. An alternative system was proposed in the same document that was motivated by Abramsky's term annotations of sequent proofs [2]. The alternative system is the same as LM except one rule: the !SR rule is replaced by the following rule straight out of the sequent calculus.

$$\frac{!\Sigma \vdash A}{!\Sigma \vdash !A}\ !\mathrm{R2}$$

The resulting system is not closed under substitution: that is, one cannot eliminate all uses of Subst from proofs using only !R. Benton et al propose a natural

I	$\dfrac{}{A \vdash A}$
Subst	$\dfrac{\Sigma \vdash A \quad A, \Gamma \vdash B}{\Sigma, \Gamma \vdash B}$
Exch. Left	$\dfrac{\Sigma, A, B, \Delta \vdash A}{\Sigma, B, A, \Delta \vdash A}$
A	$\dfrac{\Delta \vdash (A \multimap B) \qquad \Sigma \vdash A}{\Sigma, \Delta \vdash B}$
$\multimap$**R**	$\dfrac{\Sigma, A \vdash B}{\Sigma \vdash (A \multimap B)}$
$\otimes$**L**	$\dfrac{\Delta \vdash (A \otimes B) \quad \Sigma, A, B \vdash C}{\Sigma, \Delta \vdash C}$
$\otimes$**R**	$\dfrac{\Sigma \vdash A \qquad \Gamma \vdash B}{\Sigma, \Gamma \vdash (A \otimes B)}$
$\oplus$**L**	$\dfrac{\Delta \vdash (A \oplus B) \quad \Sigma, A \vdash C \quad \Sigma, B \vdash C}{\Sigma, \Delta \vdash C}$
$\oplus$**R1**	$\dfrac{\Sigma \vdash A}{\Sigma \vdash (A \oplus B)}$
$\oplus$**R2**	$\dfrac{\Sigma \vdash B}{\Sigma \vdash (A \oplus B)}$
& L1	$\dfrac{\Delta \vdash (A \,\&\, B) \qquad \Sigma, A \vdash C}{\Sigma, \Delta \vdash C}$
& L2	$\dfrac{\Delta \vdash (A \,\&\, B) \qquad \Sigma, B \vdash C}{\Sigma, \Delta \vdash C}$
& R	$\dfrac{\Sigma \vdash A \qquad \Sigma \vdash B}{\Sigma \vdash (A \,\&\, B)}$

Figure 7.4
LM: Sequent-Style Natural Deduction for Linear Logic

$$\textbf{!WL}\quad \frac{\Delta \vdash !A \qquad \Sigma \vdash B}{\Sigma, \Delta \vdash B}$$

$$\textbf{!DL}\quad \frac{\Delta \vdash !A \qquad \Sigma, A \vdash B}{\Sigma, \Delta \vdash B}$$

$$\textbf{!CL}\quad \frac{\Delta \vdash !A \qquad \Sigma, !A, !A \vdash B}{\Sigma, \Delta \vdash B}$$

$$\textbf{!SR}\quad \frac{\Delta_i \vdash !B_i \qquad !B_i, !\Sigma \vdash A}{\Delta_1 \cdots \Delta_n, !\Sigma \vdash !A}$$

$$1\textbf{L}\quad \frac{\Delta \vdash 1 \qquad \Sigma \vdash A}{\Sigma, \Delta \vdash A}$$

$$1\textbf{R}\quad \frac{}{\vdash 1}$$

Figure 7.5
LM: Continued

deduction system equivalent to LM with the !DL rule replaced by the following.

$$\frac{\Sigma \vdash !A}{\Sigma \vdash A}\ !\text{D2}$$

This rule is simpler and yields the same nice proof-theoretic properties as LM. The term annotations in the systems, discussed in the next chapter, differ as well. Mackie and Wadler also use the more complicated !DL form [112, 151], although there appears to be no advantage to that form.

Noting that the rules of !R and !D for linear ! are the same as the rules for S4, one might ask if there is a more straightforward application of Prawitz' older techniques to capture linear logic. Two answers have arisen. Troelstra [143] presents an alternate rule for !SR that instead of subproofs of $\Delta_i \vdash !B_i$, requires that a basis or slice through the proof can be made finding all conclusion sequents of the form $!B_i$. Thus the promotion rule is unary, as opposed to the N-ary !SR formulation. Troelstra has shown that this system is equivalent to the alternatives presented above.

A second alternative has been presented by Mints [123]. In this version a new grouping operator $\langle\,\rangle$ on multisets is developed. A *!-formula* is a formula beginning with !, a 1, or of the form $\langle$ *list of formulas* $\rangle$. All the modal rules of LM are replaced by the following.

$$\langle\,\rangle I \qquad \frac{\Gamma \vdash !A}{\langle\Gamma\rangle \vdash !A}$$

$$!I \qquad \frac{\Theta \vdash A}{\Theta \vdash !A} \qquad \text{Where } \Theta \text{ consists of !-formulas}$$

$$\langle\,\rangle E \qquad \frac{\langle\Gamma\rangle \vdash A}{\Gamma \vdash A}$$

$$Weak \qquad \frac{\Gamma \vdash D \qquad \Sigma \vdash B}{\Gamma, \Sigma \vdash B} \qquad \text{Where } D \text{ is a 1 or begins with !}$$

Mints demonstrates that in the resulting system every derivable sequent has a normal derivation using the standard techniques of Prawitz [131].

7.8 Summary of Chapter

In this chapter natural deduction for intuitionistic logic and classical logic are presented. Standard extensions of these systems to quantified and modal variants are sketched. Proof nets for MLL are introduced, along with brief descriptions of the development of more standard natural deduction systems for linear logic.

8 Linear Lambda Calculus

> When someone says "I want a programming language in which I need only say what I wish done," give him a lollipop. ALAN PERLIS
>
> *Epigrams on Programming* 1982

In this chapter a declarative language is presented which is based on linear logic through a Curry-Howard style correspondence. Very similar languages have been studied in the past, but in this chapter we present a type-soundness (subject-reduction) theorem, and most general type theorem.

8.1 Background

Historically, intuitionistic logic has been the basis for type systems, via the Curry-Howard isomorphism, or "formulas-as-types" principle [75]. Through this isomorphism, intuitionistic proofs of propositions may be viewed as functional programs, and logical propositions may viewed as types. A similar use of linear logic has been initiated by Girard and Lafont, and Abramsky [63, 2]. In [63], a linear calculus was developed which effectively determines reduction order, while explicitly marking the points where contraction and weakening are used. Abramsky further defined a type inference system, here called SEQ, which is discussed in Section 8.3.3. Abramsky went on to generalize this system into one incorporating quantifiers and full linear logic, a move which enabled him to interpret linear types in terms of concurrent computations. Recently Chirimar, Gunter, and Riecke [41] have implemented a version of the linear calculus. In this chapter we restrict our attention to intuitionistic linear logic.

One important property of type systems is *subject reduction,* which states that if a term t has type A, then any term produced by any number of reduction (evaluation) steps still has type A. This is crucial if we wish to use types to statically determine execution properties of terms. While it may be possible to prove subject reduction for a type system based on sequent calculus rules, there is a significant technical obstacle. If we wish to reason about the effect of reduction, we need to understand the connection between the syntactic form of a term and the set of possible types. However, with sequent calculus rules such as SEQ, a single term may have typing proofs of many different forms. (This is because uses of **Cut**, which are essential for typing terms not in normal form, are not reflected in the syntax of terms.) To avoid this problem, we formulate an equivalent set of natural deduction style typing rules, called NAT. This system has the property that for each form of linear term, there is exactly one typing rule that may be used to give a type. Using the natural deduction typing rules, subject reduction may be proved by traditional means. In addition, with syntax-directed typing rules, it is possible to formulate a unification-based algorithm that determines the most general types of any linear lambda term.

An interesting property of NAT is that one essential rule, **!SR,** based on the modal operator ! of linear logic, involves substitution into terms. Since a term may be written as the result of substitution in many different ways, this rule gives us a system in which a term may have several different principal linear types. Of course, since NAT is equivalent to SEQ, this is not an idiosyncrasy of our presentation, but a property shared by Abramsky's system SEQ that seems inherent to linear logic. If we simplify the **!SR** typing rule of NAT, we obtain an *inequivalent* system, which we call NAT2. If we restrict reduction to closed terms, then subject reduction holds for this system. However, the provable typing judgments are not closed under substitution. Essentially this system has been studied by [112], who proves the existence of unique most-general types. Benton, Bierman, de Paiva, and Hyland have studied a system very close to NAT2 with additional syntax embedded in the **!SR** rule. Their system exhibits a unique most general type property, and may have additional desirable properties [31]. This system has been extended to one that syntactically separates linear from nonlinear portions as in Girards LU [33]. Wadler also discusses alternative rules for **!SR** [149, 150, 151] Since these systems are not equivalent to SEQ and NAT, it seems an important research problem to evaluate the trade-offs between the systems.

In the following section, we describe the problems that arise in using linear logic formulas as types for pure lambda terms. This motivates the use of linear lambda calculus with explicit copy and discard primitives. In Section 8.3, we present the syntax of linear lambda calculus and the typing rules. SEQ and NAT are proved equivalent in Section 8.3, where we also prove complementary term subformula and type subformula properties for the two proof systems. A type inference algorithm and proof of most general typing are given in Section 8.4, with the subject reduction property proved in Section 8.5. The remaining sections of this chapter discuss our execution model and TIM-based implementation.

8.2 Why explicit storage operations?

In the pure lambda calculus (typed or untyped), there are no explicit `store`, `read`, `copy`, or `discard` primitives. The usual implementations of languages based on lambda calculus perform these operations as needed, according to one of several possible strategies. In other words, these operations are implicit in the language but explicit in the implementation. Since `store`, `read`, `copy`, and `discard` are explicit in the proof system of linear logic, we might attempt to insert these operations into lambda terms as part of inferring linear logic types.

In the remainder of this motivational section, two particular problems are sketched: the lack of a natural form of principal type and the failure of subject reduction theorem. The first, along with the undecidability results of [107], suggests that the process of inferring types will be algorithmically tractable only if additional operations or typing constraints are added to lambda calculus. The failure of subject reduction reinforces this conclusion by showing that additional operations are needed in the language to determine the order of function application and discarding of data.

In this section, we consider a type system derived from the NAT rules, in Appendix A.5, by modifying each rule whose name begins with ! so that the term in the consequent is the same as the term in the antecedent. This has the effect of assigning $\multimap, !$ types to pure lambda terms in a way that allows ! operations to be done implicitly at any point in the evaluation of terms. The main properties of this system are that a function of type $A \multimap B$ must use its argument of type A "exactly once" in producing a result of type B. However, if A is of the form $!C$, then the copy and discard rules associated with ! types allow us to define functions that use their argument zero or more times.

All of the intuitive points we will consider may be illustrated using the the λ term

$\lambda\, q.\, \lambda\, r.\, \lambda\, s.\, (\lambda x.q)(r\ s)$

The subterm $(\lambda x.q)$ must have a linear type of the form

$(\lambda\, x.q) : (!B \multimap A)$,

since only arguments of ! type need not appear in the body of a function. Consequently, the application $(r\,s)$ must have type $!B$. There are two possible types of r and s. One is that r is a non-discardable function that produces discardable output. That is, $r : (C \multimap !B)$, $s : C$. The other possibility is more subtle and requires more detailed understanding of the type system. If r and s are both discardable resources, such as $r : !(!C \multimap B)$ and $s : !C$, then by the usual application rule we have $rs : B$. However, whenever we have an expression of type B such that all variables appearing in the term have a type beginning with !, the **!SR** rule allows use to conclude that the term has type $!B$. Using our concepts of linear and nonlinear memory, the **!SR** rule may be explained by saying that if we define a value of type B by referring only to values in non-linear memory, the value we define may reside in non-linear memory, and have type $!B$.

From the discussion above, we can see that there are two types for the example λ-term:

$$A \multimap !(!C \multimap B) \multimap !C \multimap A$$

$$A \multimap (C \multimap !B) \multimap C \multimap A$$

Associated with these types are two different orders of evaluation. The first is the type of the function that reduces the outermost application first; the application $(r\ s)$ is thrown away before it is ever evaluated. The second type is the type of the function that first reduces the inner term $(r\ s)$, to obtain a discardable value, and then reduces the outer term. In the first case neither r nor s are used at all. In the second case they are both used, but the result is not.

The first general conclusion that follows from this example is that not all terms have a most general type with respect to substitution. This is evident since we have two types for the example term such that neither is a substitution instance of the other, and it can be checked that no shorter type is derivable for this term. Moreover, the two types are disjoint in this system: we can find terms of each type that do not have the other type.

A second conclusion follows from comparing the informal operational readings of each typing with the reduction rules of lambda calculus. In particular, consider the type $A \multimap (C \multimap !B) \multimap C \multimap A$. We may understand the correctness of this type by saying that we apply the second argument to the third and then discard the resulting discardable value. However, this informal reading assumes a particular reduction order. By the usual reduction rules of lambda calculus, we may obtain a term

$$\lambda q.\, \lambda r.\, \lambda s.\, q$$

in which the second and third arguments do not occur. Since this term does not have the linear type
$A \multimap (C \multimap !B) \multimap C \multimap A$, the subject reduction property fails for this simplified system. This is a serious problem, since we always expect types to be preserved by reduction. If types are not preserved by reduction, then reduction of well-typed terms may lead to terms that are not well-typed. Essentially, this means that static typing does not prevent run-time type errors.

Intuitively, the failure of subject reduction seems to result from the contrast between a careful accounting of resources in linear logic and the inherent ambiguity in reduction order in the λ calculus. This implies that reduction order must be restricted in some way. The most natural approach seems to be to introduce additional constants that indicate where the operations associated with ! types are performed. This restricts the set of types in a way that makes type inference possible and also provides a convenient framework for restricting evaluation order. In particular, using explicit discard, we may say explicitly, inside a term, whether a discard happens before or after a function application.

8.3 The Linear Calculus

We describe the terms of linear lambda calculus in Section 8.3.1 and give three sets of typing rules. The first, SEQ, given in Appendix A.4, is the standard set of rules given by applying the Curry-Howard isomorphism to Girard's sequent calculus proof system, restricted to intuitionistic linear logic. The second system, NAT, given in Appendix A.5, is based on a Gentzen-style sequent calculus presentation of natural deduction for intuitionistic linear logic. The third system, NAT2, is closely related to NAT, differing only in the **!SR** and **Subst** rules. The difference between NAT and NAT2 lies in the point of view taken on whether the **!SR** rule is a "left" rule or "right" rule of the sequent calculus: it introduces a type constructor on the right, so it appears to be a "right" rule, while it depends on the form of the context, or left side of a sequent, so it may also be a "left" rule. Traditionally, right rules of sequent calculus and introduction rules of natural deduction systems are analogous, while left rules of sequent calculus correspond to a combination of elimination rules and substitution. Consequently, the translation of left rules into natural deduction should be closed under substitution while the translation of right rules should be direct.

Several researchers have independently formulated similar typing rules, although none incorporate a rule of the form of the **!SR** rule of NAT. Lafont, Girard, Abramsky, and others have studied systems very similar to SEQ [63, 2]. In recent notes [1, 151] and an MS thesis [112], systems close to NAT2 have been studied. Wadler also discusses alternative rules for **!SR** and the implications of syntaxless **Subst** rule in the context of a NAT-like system. We take this parallel development of ideas as evidence that these are natural formulations of type systems based on linear logic. We show that NAT and SEQ give the same set of types to each linear term in Section 8.3.5, while NAT2 provides equivalence only up to a point. Technical theorems showing the "term-driven" nature of NAT and NAT2 and the "type-driven" nature of SEQ are proved in Section 8.3.5.

8.3.1 Linear terms and reduction

The linear calculus may be considered a functional programming language with fine-grained control over the use of data objects. To a first approximation, no function may refer to an argument twice without explicitly copying, nor ignore an argument without explicitly discarding it. The reason we say, "to a first approximation" is that the notion of referring to an argument twice is somewhat subtle, especially in the presence of additive type connectives. For example, a variable should appear "once" in both branches of a case statement, which is an additive operator. A more precise understanding of the restrictions on use and discard may be gained from reading the typing rules. In our presentation of the linear calculus, we have not restricted reduction order completely (in contrast to [2], for example.) However, as pointed out in Section 8.2, the order of certain reductions must be determinate.

Using x, y, z for term variables, and t, u, v for terms, the syntax of linear lambda terms are summarized by the grammar given in Figure 8.1.

All the `let A be B in C` constructs bind variables in A by pattern-matching A against the result of evaluating B, and then evaluate C. For example, consider the term `let` $x \times y$ `be` $(1 \times ((\lambda x.x)1))$ `in` $(x \times y)$. First the subject term is evaluated (to 1×1), then x and y are bound (both to 1), and finally $(x \times y)$ is evaluated (in the extended context) producing a final result of (1×1).

The term `store` u is a reusable, or delayed version of u. The `copy` operation inserts multiple copies of a term `store` u, while `discard` completely eliminates a `store` u term. These `copy` and `discard` operations may be implemented by pointer manipulations (implementing sharing) or by explicit copying. The `read` construct forces evaluation of a `store` d term. The interaction between `read` and `store` is the critical point where the linear calculus determines reduction order. In other terminology, `store` is a wrapper or box which is only opened when the term must be `read`.

The reduction rules for linear lambda calculus are given in Appendix A.3. A linear term t *reduces* to a linear term s if $t \twoheadrightarrow s$ can be inferred from the linear calculus evaluation rules. Abramsky has demonstrated determinacy for a more restricted form of reduction relation in [2]. The reduction relation given in Appendix A.3 only allows "nonlinear" reductions (involving the **!WL**, **!DL**, and **!CL** reduction rules) to apply at the "top level" of a term, in the empty context. However, the remaining "linear" reduction steps may be applied anywhere in a term. Thus reduction is neither a congruence with respect to all term formation rules, nor is

term	::=	x	variable
	|	let_{cut} x `be` t `in` u	bind x to result of t in u
	|	`let` $x \times y$ `be` t `in` u	bind x to car, y to cdr of t in u
	|	$(t \times u)$	eager pair (like cons in ML)
	|	$(t\, u)$	application
	|	$(\lambda x.t)$	abstraction
	|	$inr(t)$	determines right branch of case
	|	$inl(t)$	determines left branch of case
	|	`case` t `of` $inl(x) \Rightarrow u, inr(y) \Rightarrow v$	evaluate t then branch
	|	$\langle t, u\rangle$	lazy pair
	|	`let` $\langle \_, x\rangle$ `be` t `in` u	bind x to cdr of lazy pair
	|	`let` $\langle x, \_\rangle$ `be` t `in` u	bind x to car of lazy pair
	|	`let` 1 `be` t `in` u	evaluate t to 1, then become u
	|	`store` u	store or delay u
	|	`discard` t `in` u	throw away t
	|	`read store` x `as` t `in` u	evaluate t, bind x
	|	`copy` $x@y$ `as` t `in` u	binds x and y to t

Figure 8.1
Grammar of the Linear Lambda Calculus

it deterministic. With Mitschke's δ-reduction theorem [23] this reduction system can be shown to be confluent on untyped terms, even though not all untyped terms have a normal form. For typed terms, the usual cut-elimination procedure provides a proof of weak normalization for this reduction system.

8.3.2 Typing preliminaries

We review some standard definitions. A *type multiset* or *type environment* is a multiset of pairs $x_i : A_i$ of variables x_i and linear logic formulas A_i. A *typing judgment* is a type multiset Γ, a single linear term t, and a single linear logic formula A, separated by a $\vdash$, constructed as follows: $\Gamma \vdash t : A$. A *typing deduction* is a tree, presented with the root at the bottom, and the leaves at the top. Each branch of a deduction is a sequence of applications of the proof rules, some of which, such as $\otimes R$ in SEQ, represent branching points in the deduction tree, some, such as $\multimap R$, which extend the length of a branch, and some, such as identity, which terminate a branch. Any branch not terminated by identity or 1 **R** is called an assumption. The leaves therefore embody the type assumptions and the root the conclusion. Such a structure is said to be a deduction of the conclusion from the assumptions. A *proof* is a typing deduction with no assumptions. That is, all the branches terminate with an application or identity or 1 **R**. A closed linear term t is said to be *linearly typeable* if there exists some proof with conclusion $\vdash t : A$ for some linear formula A.

8.3.3 The typing rules

The first system we will study is called SEQ, the rules for which are given in Appendix A.4. This formulation is due mainly to Abramsky [2]. We have modified the syntax used in the original presentation slightly, but the idea is the same: take the rules for (intuitionistic) propositional linear logic and decorate them with linear terms. In the system SEQ, cut-free derivations produce linear terms in normal form. Some derivations with cut correspond to linear terms in non-normal form, and cut-elimination steps transform the term, essentially performing beta-reduction and other linear reduction steps. Performing cut-elimination on an SEQ proof is analogous to reduction in the linear calculus, although the exact correspondence is somewhat complicated.

The typing rules for the second system, called NAT, are given in Appendix A.5. The main difference between the two systems is that NAT is more "term-driven," while SEQ is more "type-driven." In Section 8.3.5, we show that NAT is equivalent to SEQ for proving typing judgments. This is not surprising since NAT is based on a

Gentzen-style sequent calculus presentation of natural deduction for intuitionistic linear logic, while SEQ is based on a sequent calculus presentation of the same logic. The term "decorations" have been chosen in NAT so that the provable sequents in these systems are the same.

In devising the NAT rules from SEQ, we were guided by the correspondence between intuitionistic sequent calculus and natural deduction [131, Appendix A]. The main idea is to interpret sequent proof rules as instructions for constructing natural deduction proofs. The sequent rules acting on the left determine constructions on the top (hypotheses) of a natural deduction proof, while sequent rules acting on the right extend the natural deduction proof from the bottom (conclusion). The **Cut** rule is interpreted by substituting a proof for a hypothesis. In order for this to work, the natural deduction proof system must have the substitution property formalized by the derived **Subst** rule in Appendix A.5.

A simple example that illustrates the general pattern is the tensor rule acting on the left, $\otimes$**L**. In the conclusion of the sequent rule, there is a new term variable $z :(A \otimes B)$. However, we want natural deduction proofs to be closed under the operation of substituting terms for variables (hypotheses). If we use **Cut** in a sequent proof to replace $z :(A \otimes B)$, we end up with a sequence of proof steps whose hypotheses and conclusion are identical to the antecedent and consequent of the natural deduction $\otimes$**L** rule in Appendix A.5.

The most unusual rule of the NAT system is the **!SR** rule. This may be understood by considering the **!SR** rule of SEQ and remembering that when we substitute proofs for hypotheses in NAT, we must still have a well-formed natural deduction proof. If we follow SEQ **!SR** with **Cut**, we may use $\Delta \vdash t :!B$ and $x :!B, !\Sigma \vdash u :A$ to prove $\Delta, !\Sigma \vdash \texttt{store}\ u[t/x] :!A$. Generalizing to any number of **Cut**'s, so that natural deduction proofs will be closed under substitution, we obtain the **!SR** rule in Appendix A.5. This rule may without loss of generality be restricted to the case where all the Δ_i contain some non-! type.

The third and final system we consider is called NAT2. The NAT2 rules are generated from the NAT rules by removing the rules of **Subst** and **!SR** from NAT and replacing them with:

$$\frac{\Sigma \vdash t :A \quad x :A, \Gamma \vdash u :B}{\Sigma, \Gamma \vdash\ let_{cut}\, x\ \texttt{be}\ t\ \texttt{in}\ u :B}\ let_{cut}$$

$$\frac{!\Sigma \vdash t :A}{!\Sigma \vdash\ \texttt{store}\ t :!A}\ \textbf{!SR}$$

The reason for the explicit syntax of let_{cut} is that **Subst** is not a derived rule of NAT2, and for Theorem 28 we need some cut-like rule. These differences lead to subtlely different properties of NAT and NAT2 systems. For example, since **Subst** is syntaxless in NAT, one can show that that one need not consider **Subst** in searching for type derivations. On the other hand, NAT2 is entirely driven by term syntax, leading to a unique principle type theorem.

8.3.4 nat Subst Elimination

The subst elimination theorem states that whatever can be proven in NAT can also be proven without the use of the **Subst** rule.

The following detailed demonstration of the subst elimination theorem consists of a proof normalization procedure which slowly eliminates subst from any NAT proof. The procedure may increase the size of the proof, although of course it will still be a proof of the same sequent.

We will call a formula which appears in a hypothesis of an application of **Subst** but which does not occur in the conclusion a *subst formula.* In the list of NAT rules in appendix A.5 the subst formula in the **Subst** rule is the formula named A.

We also define the *degree* of a **Subst** to be the number of symbols in its subst formulas. For concreteness, we define here what is meant by number of symbols. We will consider each propositional symbol p_i to be a single symbol. We also consider the negation of each propositional symbol ${p_i}^\perp$ to be a single symbol. Finally, we count each connective and constant, $\otimes, \multimap, \oplus, \&, ?, !, 1, \perp, 0, \top$, as a single symbol, but do not count parentheses. It is important to note that negation is defined, and therefore is not a connective. We also define the *degree* of a proof to be the maximum degree of any subst in the proof, or zero if there are no substs.

Operationally, the subst elimination procedure defined below first finds one of the "highest" substs of maximal degree in the proof. That is, an application of **Subst** for which all applications of **Subst** in the derivation of both hypotheses are of smaller degree. Then a reduction is applied to that occurrence of **Subst**, which simplifies or eliminates it, although it may replicate some other portions of the original proof. We iterate this procedure to remove all substs of some degree, and then iterate the entire procedure to eliminate all substs. In this way, any NAT proof may be normalized into one without any uses of the **Subst** rule, at the possible expense of an exponential blowup in the size of the resulting proof tree.

Technically, we begin with a lemma which constitutes the heart of the proof of subst-elimination. Although the proof of this lemma is rather lengthy, the reasoning is straightforward, and the remainder of the proof of subst-elimination is quite simple.

LEMMA 30 (REDUCE-ONE-SUBST) Given a proof of the sequent $\Gamma \vdash A$ in NAT which ends in an application of **Subst** of degree d, and where the degree of the proofs of both hypothesis is less than d, we may construct a proof of $\Gamma \vdash A$ in NAT of degree less than d.

Proof 29 By induction on the size of the proof of $\Gamma \vdash A$.

Given a derivation which ends in a **Subst**, we perform case analysis on the rule which is applied immediately above the **Subst** on the right hand branch.

In each case we will provide a reduction, which may eliminate the subst entirely, or replace it with one or two smaller substs. Since this is a proof by induction on the size of a derivation, one may view this proof as a procedure which pushes **Subst**s of large degree up a derivation. Informally, this procedure pushes **Subst**s up through a derivation until the critical point is reached where the right branch of the proof above the subst is simply identity or some constant rule.

We now give the reductions for every proof rule on the right branch.

I

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : A\end{array} \qquad \dfrac{}{x : A \vdash x : A}\,{\scriptstyle I}}{\Gamma \vdash x[t/x] : A}\,{\scriptstyle Subst}$$

$$\Downarrow$$

$$\begin{array}{c}\vdots\\ \Gamma \vdash t : A\end{array}$$

The formula t is identical to the formula $x[t/x]$ by the definition of substitution.

Subst, formula descends from right

$$\dfrac{\begin{array}{c}\vdots\\ \Delta \vdash t : C\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ x : C, \Sigma \vdash u : A\end{array} \quad \begin{array}{c}\vdots\\ y : A, \Gamma \vdash v : D\end{array}}{x : C, \Sigma, \Gamma \vdash v[u/y] : D}\,{\scriptstyle Subst}}{\Delta, \Sigma, \Gamma \vdash (v[u/y])[t/x] : D}\,{\scriptstyle Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Delta \vdash t : C\end{array} \quad \begin{array}{c}\vdots\\ x : C, \Sigma \vdash u : A\end{array}}{\Delta, \Sigma \vdash u[t/x] : A}\,{\scriptstyle Subst} \qquad \begin{array}{c}\vdots\\ y : A, \Gamma \vdash v : D\end{array}}{\Delta, \Sigma, \Gamma \vdash v[(u[t/x])/y] : D}\,{\scriptstyle Subst}$$

The terms $(v[u/y])[t/x]$: and $v[(u[t/x])/y]$: are the same since x doesn't occur in v.

Subst, formula descends from left

$$\dfrac{\begin{array}{c}\vdots\\ \Delta \vdash t :C\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ \Sigma \vdash u :A\end{array} \quad \begin{array}{c}\vdots\\ x :A, y :C, \Gamma \vdash v :D\end{array}}{\Sigma, y :C, \Gamma \vdash v[u/x] :D}{\scriptstyle Subst}}{\Sigma, \Delta, \Gamma \vdash (v[u/x])[t/y] :D}{\scriptstyle Subst}$$

$$\Downarrow$$

$$\dfrac{\begin{array}{c}\vdots\\ \Sigma \vdash u :A\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ \Delta \vdash t :C\end{array} \quad \begin{array}{c}\vdots\\ x :A, y :C, \Gamma \vdash v :D\end{array}}{x :A, \Delta, \Gamma \vdash v[t/y] :D}{\scriptstyle Subst}}{\Sigma, \Delta, \Gamma \vdash (v[t/y])[u/x] :D}{\scriptstyle Subst}$$

The terms $(v[u/x])[t/y]$: and $(v[t/y])[u/x]$: are the same since x isn't in t, and y isn't in u.

$\otimes$ R, formula descends from right

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t :C\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ x :C, \Sigma \vdash u :A\end{array} \qquad \begin{array}{c}\vdots\\ \Delta \vdash v :B\end{array}}{x :C, \Sigma, \Delta \vdash (u \times v) :(A \otimes B)}{\scriptstyle \otimes R}}{\Gamma, \Sigma, \Delta \vdash (u \times v)[t/x] :(A \otimes B)}{\scriptstyle Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t :C\end{array} \qquad \begin{array}{c}\vdots\\ x :C, \Sigma \vdash u :A\end{array}}{\Gamma, \Sigma \vdash u[t/x] :A}{\scriptstyle Subst} \qquad \begin{array}{c}\vdots\\ \Delta \vdash v :B\end{array}}{\Gamma, \Sigma, \Delta \vdash (u[t/x] \times v) :(A \otimes B)}{\scriptstyle \otimes R}$$

The formula $(u[t/x] \times v)$ is identical to the formula $(u \times v)[t/x]$ since x may not occur in v.

$\otimes$ R, formula descends from left

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : C\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ \Sigma \vdash u : A\end{array} \qquad \begin{array}{c}\vdots\\ x : C, \Delta \vdash v : B\end{array}}{\Sigma, x : C, \Delta \vdash (u \times v) : (A \otimes B)}{\scriptstyle \otimes R}}{\Sigma, \Gamma, \Delta \vdash (u \times v)[t/x] : (A \otimes B)}{\scriptstyle Subst}$$

$$\Downarrow$$

$$\dfrac{\begin{array}{c}\vdots\\ \Sigma \vdash u : A\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : C\end{array} \qquad \begin{array}{c}\vdots\\ x : C, \Delta \vdash v : B\end{array}}{\Gamma, \Delta \vdash v[t/x] : A}{\scriptstyle Subst}}{\Sigma, \Gamma, \Delta \vdash (u \times v[t/x]) : (A \otimes B)}{\scriptstyle \otimes R}$$

The formula $(u \times v[t/x])$ is identical to the formula $(u \times v)[t/x]$ since x may not occur in u.

$\otimes$ L, formula descends from the right

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : C\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ \Delta \vdash r : (A \otimes B)\end{array} \qquad \begin{array}{c}\vdots\\ \Sigma, x : A, y : B, s : C \vdash u : D\end{array}}{\Sigma, \Delta, s : C \vdash \text{ let } (x \times y) \text{ be } r \text{ in } u : D}{\scriptstyle \otimes L}}{\Sigma, \Delta, \Gamma \vdash (\text{ let } (x \times y) \text{ be } r \text{ in } u)[t/s] : D}{\scriptstyle Subst}$$

$$\Downarrow$$

$$\dfrac{\begin{array}{c}\vdots\\ \Delta \vdash r : (A \otimes B)\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : C\end{array} \qquad \begin{array}{c}\vdots\\ \Sigma, x : A, y : B, s : C \vdash u : D\end{array}}{\Sigma, x : A, y : B, \Gamma \vdash u[t/s] :}{\scriptstyle Subst}}{\Sigma, \Delta, \Gamma \vdash \text{ let } (x \times y) \text{ be } r \text{ in } (u[t/s]) : D}{\scriptstyle \otimes L}$$

$\otimes$ L, formula descends from the left

$$\dfrac{\begin{matrix}\vdots\\ \Gamma\vdash t:C\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\\ \Delta, s:C \vdash r:(A\otimes B)\end{matrix} \qquad \begin{matrix}\vdots\\ \Sigma, x:A, y:B\vdash u:D\end{matrix}}{\Sigma,\Delta,s:C\vdash \text{ let } (x\times y) \text{ be } r \text{ in } u :D}{\scriptstyle\otimes L}}{\Sigma,\Delta,\Gamma\vdash (\text{ let }(x\times y)\text{ be } r \text{ in } u)[t/s]:D}{\scriptstyle Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \Gamma\vdash t:C\end{matrix} \qquad \begin{matrix}\vdots\\ \Delta, s:C \vdash r:(A\otimes B)\end{matrix}}{\Delta,\Gamma\vdash r[t/s]:(A\otimes B)}{\scriptstyle Subst} \qquad \begin{matrix}\vdots\\ \Sigma, x:A, y:B\vdash u:D\end{matrix}}{\Sigma,\Delta,\Gamma\vdash \text{ let } (x\times y) \text{ be } (r[t/s]) \text{ in } u :D}{\scriptstyle\otimes L}$$

$\multimap$ R

$$\dfrac{\begin{matrix}\vdots\\ \Gamma\vdash t:C\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\\ \Sigma, s:C, x:A \vdash u:B\end{matrix}}{\Sigma, s:C\vdash \lambda x.u:(A\multimap B)}{\scriptstyle\multimap R}}{\Sigma,\Gamma\vdash (\lambda x.u)[t/s]:(A\multimap B)}{\scriptstyle Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \Gamma\vdash t:C\end{matrix} \qquad \begin{matrix}\vdots\\ \Sigma, s:C, x:A \vdash u:B\end{matrix}}{\Sigma,\Gamma, x:A\vdash u[t/s]:B}{\scriptstyle Subst}}{\Sigma,\Gamma\vdash \lambda x.(u[t/s]):(A\multimap B)}{\scriptstyle\multimap R}$$

The formula $\lambda x.(u[t/s])$ is identical to the formula $(\lambda x.u)[t/s]$ since x may not occur in t.

A, formula descends from right

$$\dfrac{\begin{matrix}\vdots\\ \Gamma\vdash v:C\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\\ \Delta\vdash t:(A\multimap B)\end{matrix} \qquad \begin{matrix}\vdots\\ x:C,\Sigma\vdash u:A\end{matrix}}{x:C,\Sigma,\Delta\vdash (tu):B}{\scriptstyle A}}{\Sigma,\Gamma,\Delta\vdash (tu)[v/x]:B}{\scriptstyle Subst}$$

$$\Downarrow$$

$$\dfrac{\begin{matrix}\vdots\\ \Delta\vdash t:(A\multimap B)\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\\ \Gamma\vdash v:C\end{matrix} \qquad \begin{matrix}\vdots\\ x:C,\Sigma\vdash u:A\end{matrix}}{\Gamma,\Sigma\vdash u[v/x]:A}{\scriptstyle Subst}}{\Sigma,\Gamma,\Delta\vdash (t(u[v/x])):B}{\scriptstyle A}$$

A, formula descends from left

$$
\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash v : C\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ x : C, \Delta \vdash t : (A \multimap B)\end{array} \qquad \begin{array}{c}\vdots\\ \Sigma \vdash u : A\end{array}}{x : C, \Sigma, \Delta \vdash (tu) : B}{\scriptstyle A}}{\Gamma, \Sigma, \Delta \vdash (tu)[v/x] : B}{\scriptstyle Subst}
$$

$$\Downarrow$$

$$
\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash v : C\end{array} \qquad \begin{array}{c}\vdots\\ x : C, \Delta \vdash t : (A \multimap B)\end{array}}{\Gamma, \Sigma \vdash t[v/x] : (A \multimap B)}{\scriptstyle Subst} \qquad \begin{array}{c}\vdots\\ \Sigma \vdash u : A\end{array}}{\Gamma, \Sigma, \Delta \vdash ((t[v/x])u) : B}{\scriptstyle A}
$$

$\oplus$ R1

$$
\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : C\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ x : C, \Sigma \vdash u : A\end{array}}{x : C, \Sigma \vdash inl(u) : (A \oplus B)}{\scriptstyle \oplus R}}{\Gamma, \Sigma \vdash inl(u)[t/x] : (A \oplus B)}{\scriptstyle Subst}
$$

$$\Downarrow$$

$$
\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : C\end{array} \qquad \begin{array}{c}\vdots\\ x : C, \Sigma \vdash u : A\end{array}}{\Gamma, \Sigma \vdash u[t/x] : A}{\scriptstyle Subst}}{\Gamma, \Sigma \vdash inl(u[t/x]) : (A \oplus B)}{\scriptstyle \oplus R}
$$

$\oplus$ R2

$$
\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : C\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ x : C, \Sigma \vdash u : B\end{array}}{x : C, \Sigma \vdash inr(u) : (A \oplus B)}{\scriptstyle \oplus R}}{\Gamma, \Sigma \vdash inr(u)[t/x] : (A \oplus B)}{\scriptstyle Subst}
$$

$$\Downarrow$$

$$
\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : C\end{array} \qquad \begin{array}{c}\vdots\\ x : C, \Sigma \vdash u : B\end{array}}{\Gamma, \Sigma \vdash u[t/x] : B}{\scriptstyle Subst}}{\Gamma, \Sigma \vdash inr(u[t/x]) : (A \oplus B)}{\scriptstyle \oplus R}
$$

$\oplus$ L, formula descends from the right

$$
\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : C\end{array} \quad \dfrac{\begin{array}{c}\vdots\\ \Delta \vdash w : (A \oplus B)\end{array} \quad \begin{array}{c}\vdots\\ x : C, \Sigma, y : A \vdash u : D\end{array} \quad \begin{array}{c}\vdots\\ x : C, \Sigma, j : B \vdash v : D\end{array}}{x : C, \Sigma, \Delta \vdash \text{ case } w \text{ of } inl(y) \Rightarrow u, inr(j) \Rightarrow v : D}\oplus L}{\Gamma, \Sigma, \Delta \vdash (\text{ case } w \text{ of } inl(y) \Rightarrow u, inr(j) \Rightarrow v)[t/x] : D}Subst
$$

$$\Downarrow$$

$$
\dfrac{\begin{array}{c}\vdots\\ \Delta \vdash w : (A \oplus B)\end{array} \quad \dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : C\end{array} \quad \begin{array}{c}\vdots\\ x : C, \Sigma, y : A \vdash u : D\end{array}}{\Gamma, \Sigma, y : A \vdash u[t/x] : D}Subst \quad \mathcal{A}}{\Gamma, \Sigma, \Delta \vdash \text{ case } w \text{ of } inl(y) \Rightarrow u[t/x], inr(j) \Rightarrow v[t/x] : D}\oplus L
$$

Where for space reasons $\mathcal{A}$ stands for the proof:

$$
\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : C\end{array} \quad \begin{array}{c}\vdots\\ x : C, \Sigma, j : B \vdash v : D\end{array}}{\Gamma, \Sigma, j : B \vdash v[t/x] : D}Subst
$$

$\oplus$ L,formula descends from the left

$$
\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : C\end{array} \quad \dfrac{\begin{array}{c}\vdots\\ x : C, \Delta \vdash w : (A \oplus B)\end{array} \quad \begin{array}{c}\vdots\\ \Sigma, y : A \vdash u : D\end{array} \quad \begin{array}{c}\vdots\\ \Sigma, j : B \vdash v : D\end{array}}{x : C, \Sigma, \Delta \vdash \text{ case } w \text{ of } inl(y) \Rightarrow u, inr(j) \Rightarrow v : D}\oplus L}{\Gamma, \Sigma, \Delta \vdash (\text{ case } w \text{ of } inl(y) \Rightarrow u, inr(j) \Rightarrow v)[t/x] : D}Subst
$$

$$\Downarrow$$

$$
\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : C\end{array} \quad \begin{array}{c}\vdots\\ x : C, \Delta \vdash w : (A \oplus B)\end{array}}{\Gamma, \Delta \vdash w[t/x] : (A \oplus B)}Subst \quad \begin{array}{c}\vdots\\ \Sigma, y : A \vdash u : D\end{array} \quad \mathcal{A}}{\Sigma, \Gamma, \Delta \vdash \text{ case } w[t/x] \text{ of } inl(y) \Rightarrow u, inr(j) \Rightarrow v : D}\oplus L
$$

Where for space reasons, the $\mathcal{A}$ stands for the proof:

$$
\begin{array}{c}\vdots\\ \Sigma, j : B \vdash v : D\end{array}
$$

& R

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma\vdash t:C\end{array}\qquad \dfrac{\begin{array}{c}\vdots\\ x:C,\Sigma\vdash u:A\end{array}\quad \begin{array}{c}\vdots\\ x:C,\Sigma\vdash v:B\end{array}}{x:C,\Sigma\vdash \langle u,v\rangle:(A\&B)}{\scriptstyle \&R}}{\Gamma,\Sigma\vdash\langle u,v\rangle[t/x]:(A\oplus B)}{\scriptstyle Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Gamma\vdash t:C\end{array}\quad \begin{array}{c}\vdots\\ x:C,\Sigma\vdash u:A\end{array}}{\Gamma,\Sigma\vdash u[t/x]:A}{\scriptstyle Subst}\quad \dfrac{\begin{array}{c}\vdots\\ \Gamma\vdash t:C\end{array}\quad \begin{array}{c}\vdots\\ x:C,\Sigma\vdash v:B\end{array}}{\Gamma,\Sigma\vdash v[t/x]:B}{\scriptstyle Subst}}{\Gamma,\Sigma\vdash\langle u[t/x],v[t/x]\rangle:(A\&B)}{\scriptstyle \&R}$$

& L1, formula descends from the right

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma\vdash t:C\end{array}\qquad \dfrac{\begin{array}{c}\vdots\\ \Delta\vdash w:(A\oplus B)\end{array}\quad \begin{array}{c}\vdots\\ x:C,\Sigma,y:A\vdash u:D\end{array}}{x:C,\Sigma,\Delta\vdash \text{ let }\langle y,\_\rangle\text{ be }w\text{ in }u:D}{\scriptstyle \&L1}}{\Gamma,\Sigma,\Delta\vdash(\text{ let }\langle y,\_\rangle\text{ be }w\text{ in }u)[t/x]:D}{\scriptstyle Subst}$$

$$\Downarrow$$

$$\dfrac{\begin{array}{c}\vdots\\ \Delta\vdash w:(A\oplus B)\end{array}\qquad \dfrac{\begin{array}{c}\vdots\\ \Gamma\vdash t:C\end{array}\quad \begin{array}{c}\vdots\\ x:C,\Sigma,y:A\vdash u:D\end{array}}{\Gamma,\Sigma,y:A\vdash u[t/x]:D}{\scriptstyle Subst}}{\Gamma,\Sigma,\Delta\vdash\text{ let }\langle y,\_\rangle\text{ be }w\text{ in }(u[t/x]):D}{\scriptstyle \&L1}$$

& L1, formula descends from the left

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma\vdash t:C\end{array}\qquad \dfrac{\begin{array}{c}\vdots\\ x:C,\Delta\vdash w:(A\oplus B)\end{array}\quad \begin{array}{c}\vdots\\ \Sigma,y:A\vdash u:D\end{array}}{x:C,\Sigma,\Delta\vdash \text{ let }\langle y,\_\rangle\text{ be }w\text{ in }u:D}{\scriptstyle \&L1}}{\Gamma,\Sigma,\Delta\vdash(\text{ let }\langle y,\_\rangle\text{ be }w\text{ in }u)[t/x]:D}{\scriptstyle Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Gamma\vdash t:C\end{array}\quad \begin{array}{c}\vdots\\ x:C,\Delta\vdash w:(A\oplus B)\end{array}}{\Gamma,\Delta\vdash w[t/x]:(A\oplus B)}{\scriptstyle Subst}\qquad \begin{array}{c}\vdots\\ \Sigma,y:A\vdash u:D\end{array}}{\Gamma,\Sigma,\Delta\vdash\text{ let }\langle y,\_\rangle\text{ be }w[t/x]\text{ in }u:D}{\scriptstyle \&L1}$$

& L2, formula descends from the right

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : C\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ \Delta \vdash w : (A \oplus B)\end{array} \qquad \begin{array}{c}\vdots\\ x : C, \Sigma, y : B \vdash u : D\end{array}}{x : C, \Sigma, \Delta \vdash \text{ let } \langle \_, y \rangle \text{ be } w \text{ in } u : D}{\scriptstyle \& L2}}{\Gamma, \Sigma, \Delta \vdash (\text{ let } \langle \_, y \rangle \text{ be } w \text{ in } u)[t/x] : D}{\scriptstyle Subst}$$

$$\Downarrow$$

$$\dfrac{\begin{array}{c}\vdots\\ \Delta \vdash w : (A \oplus B)\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : C\end{array} \qquad \begin{array}{c}\vdots\\ x : C, \Sigma, y : B \vdash u : D\end{array}}{\Gamma, \Sigma, y : B \vdash u[t/x] : D}{\scriptstyle Subst}}{\Gamma, \Sigma, \Delta \vdash \text{ let } \langle \_, y \rangle \text{ be } w \text{ in } (u[t/x]) : D}{\scriptstyle \& L2}$$

& L2, formula descends from the left

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : C\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ x : C, \Delta \vdash w : (A \oplus B)\end{array} \qquad \begin{array}{c}\vdots\\ \Sigma, y : B \vdash u : D\end{array}}{x : C, \Sigma, \Delta \vdash \text{ let } \langle \_, y \rangle \text{ be } w \text{ in } u : D}{\scriptstyle \& L2}}{\Gamma, \Sigma, \Delta \vdash (\text{ let } \langle \_, y \rangle \text{ be } w \text{ in } u)[t/x] : D}{\scriptstyle Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : C\end{array} \qquad \begin{array}{c}\vdots\\ x : C, \Delta \vdash w : (A \oplus B)\end{array}}{\Gamma, \Delta \vdash w[t/x] : (A \oplus B)}{\scriptstyle Subst} \qquad \begin{array}{c}\vdots\\ \Sigma, y : B \vdash u : D\end{array}}{\Gamma, \Sigma, \Delta \vdash \text{ let } \langle \_, y \rangle \text{ be } w[t/x] \text{ in } u : D}{\scriptstyle \& L2}$$

!WL, formula descends from the right

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : C\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ \Delta \vdash w : !B\end{array} \qquad \begin{array}{c}\vdots\\ x : C, \Sigma \vdash u : D\end{array}}{x : C, \Sigma, \Delta \vdash \text{ discard } w \text{ in } u : D}{\scriptstyle !WL}}{\Gamma, \Sigma, \Delta \vdash (\text{ discard } w \text{ in } u)[t/x] : D}{\scriptstyle Subst}$$

$$\Downarrow$$

$$\dfrac{\begin{array}{c}\vdots\\ \Delta \vdash w : !B\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash t : C\end{array} \qquad \begin{array}{c}\vdots\\ x : C, \Sigma \vdash u : D\end{array}}{\Gamma, \Sigma \vdash u[t/x] : D}{\scriptstyle Subst}}{\Gamma, \Sigma, \Delta \vdash \text{ discard } w \text{ in } (u[t/x]) : D}{\scriptstyle !WL}$$

!WL, formula descends from the left

$$\dfrac{\begin{matrix}\vdots\\ \Gamma \vdash t : C\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\\ x : C, \Delta \vdash w : !B\end{matrix} \qquad \begin{matrix}\vdots\\ \Sigma \vdash u : D\end{matrix}}{x : C, \Sigma, \Delta \vdash \text{ discard } w \text{ in } u : D}{\scriptstyle !WL}}{\Gamma, \Sigma, \Delta \vdash (\text{ discard } w \text{ in } u)[t/x] : D}{\scriptstyle Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \Gamma \vdash t : C\end{matrix} \qquad \begin{matrix}\vdots\\ x : C, \Delta \vdash w : !B\end{matrix}}{\Gamma, \Delta \vdash w[t/x] : !B}{\scriptstyle Subst} \qquad \begin{matrix}\vdots\\ \Sigma \vdash u : D\end{matrix}}{\Gamma, \Sigma, \Delta \vdash \text{ discard } w \text{ in } u : D}{\scriptstyle !WL}$$

!CL, formula descends from the right

$$\dfrac{\begin{matrix}\vdots\\ \Gamma \vdash t : C\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\\ \Delta \vdash w : !B\end{matrix} \qquad \begin{matrix}\vdots\\ s : C, \Sigma, x : !B, y : !B \vdash u : D\end{matrix}}{s : C, \Sigma, \Delta \vdash \text{ copy } x@y \text{ as } w \text{ in } u : D}{\scriptstyle !CL}}{\Gamma, \Sigma, \Delta \vdash (\text{ copy } x@y \text{ as } w \text{ in } u)[t/s] : D}{\scriptstyle Subst}$$

$$\Downarrow$$

$$\dfrac{\begin{matrix}\vdots\\ \Delta \vdash w : !B\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\\ \Gamma \vdash t : C\end{matrix} \qquad \begin{matrix}\vdots\\ s : C, \Sigma, x : !B, y : !B \vdash u : D\end{matrix}}{\Gamma, \Sigma, x : !B, y : !B \vdash u[t/s] : D}{\scriptstyle Subst}}{\Gamma, \Sigma, \Delta \vdash \text{ copy } x@y \text{ as } w \text{ in } (u[t/s] : D}{\scriptstyle !CL}$$

!CL, formula descends from the left

$$\dfrac{\begin{matrix}\vdots\\ \Gamma \vdash t : C\end{matrix} \qquad \dfrac{\begin{matrix}\vdots\\ s : C, \Delta \vdash w : !B\end{matrix} \qquad \begin{matrix}\vdots\\ \Sigma, x : !B, y : !B \vdash u : D\end{matrix}}{s : C, \Sigma, \Delta \vdash \text{ copy } x@y \text{ as } w \text{ in } u : D}{\scriptstyle !CL}}{\Gamma, \Sigma, \Delta \vdash (\text{ let } x@y \text{ as } w \text{ in } u)[t/s] : D}{\scriptstyle Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ \Gamma \vdash t : C\end{matrix} \qquad \begin{matrix}\vdots\\ s : C, \Delta \vdash w : !B\end{matrix}}{\Gamma, \Delta \vdash w[t/s] : !B}{\scriptstyle Subst} \qquad \begin{matrix}\vdots\\ \Sigma, x : !B, y : !B \vdash u : D\end{matrix}}{\Gamma, \Sigma, \Delta \vdash \text{ copy } x@y \text{ as } (w[t/s]) \text{ in } u : D}{\scriptstyle !CL}$$

!DL, formula descends from the right

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash t : C} \qquad \dfrac{\dfrac{\vdots}{\Delta \vdash w : !B} \qquad \dfrac{\vdots}{x : C, \Sigma, y : B \vdash u : D}}{x : C, \Sigma, \Delta \vdash \text{ read store}y \text{ as } w \text{ in } u : D}\,{!DL}}{\Gamma, \Sigma, \Delta \vdash (\text{ read store}y \text{ as } w \text{ in } u)[t/x] : D}\,\mathit{Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\vdots}{\Delta \vdash w : !B} \qquad \dfrac{\dfrac{\vdots}{\Gamma \vdash t : C} \qquad \dfrac{\vdots}{x : C, \Sigma, y : B \vdash u : D}}{\Gamma, \Sigma, y : B \vdash u[t/x] : D}\,\mathit{Subst}}{\Gamma, \Sigma, \Delta \vdash \text{ read store } y \text{ as } w \text{ in } (u[t/x]) : D}\,{!DL}$$

!DL, formula descends from the left

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash t : C} \qquad \dfrac{\dfrac{\vdots}{x : C, \Delta \vdash w : !B} \qquad \dfrac{\vdots}{\Sigma, y : B \vdash u : D}}{x : C, \Sigma, \Delta \vdash \text{ read store } y \text{ as } w \text{ in } u : D}\,{!DL}}{\Gamma, \Sigma, \Delta \vdash (\text{ read store}y \text{ as } w \text{ in } u)[t/x] : D}\,\mathit{Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash t : C} \qquad \dfrac{\vdots}{x : C, \Delta \vdash w : !B}}{\Gamma, \Delta \vdash w[t/x] : !B}\,\mathit{Subst} \qquad \dfrac{\vdots}{\Sigma, y : B \vdash u : D}}{\Gamma, \Sigma, \Delta \vdash \text{ read store } y \text{ as } w[t/x] \text{ in } u : D}\,{!DL}$$

1L, formula descends from the right

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash t : C} \qquad \dfrac{\dfrac{\vdots}{\Delta \vdash w : 1} \qquad \dfrac{\vdots}{x : C, \Sigma \vdash u : D}}{x : C, \Sigma, \Delta \vdash \text{ let } w \text{ be } 1 \text{ in } u : D}\,1L}{\Gamma, \Sigma, \Delta \vdash (\text{let } w \text{ be } 1 \text{ in } u)[t/x] : D}\,\mathit{Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\vdots}{\Delta \vdash w : 1} \qquad \dfrac{\dfrac{\vdots}{\Gamma \vdash t : C} \qquad \dfrac{\vdots}{x : C, \Sigma \vdash u : D}}{\Gamma, \Sigma \vdash u[t/x] : D}\,\mathit{Subst}}{\Gamma, \Sigma, \Delta \vdash \text{ let } w \text{ be } 1 \text{ in } (u[t/x]) : D}\,1L$$

1L, formula descends from the left

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash t : C} \qquad \dfrac{\dfrac{\vdots}{x : C, \Delta \vdash w : 1} \qquad \dfrac{\vdots}{\Sigma \vdash u : D}}{x : C, \Sigma, \Delta \vdash \text{ let } w \text{ be } 1 \text{ in } u : D}{\scriptstyle 1L}}{\Gamma, \Sigma, \Delta \vdash (\text{let } w \text{ be } 1 \text{ in } u)[t/x] : D}{\scriptstyle Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash t : C} \qquad \dfrac{\vdots}{x : C, \Delta \vdash w : 1}}{\Gamma, \Sigma \vdash w[t/x] : 1}{\scriptstyle Subst} \qquad \dfrac{\vdots}{\Sigma \vdash u : D}}{\Gamma, \Sigma, \Delta \vdash \text{ let } w[t/x] \text{ be } 1 \text{ in } u : D}{\scriptstyle 1L}$$

$\top$ **R**

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash t : C} \qquad \dfrac{}{x : C, \Sigma \vdash \top : \top}{\scriptstyle \top R}}{\Gamma, \Sigma \vdash \top[t/x] : \top}{\scriptstyle Subst}$$

$$\Downarrow$$

$$\dfrac{}{\Gamma, \Sigma \vdash \top : \top}{\scriptstyle \top R}$$

0 L subst formula on the left

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash t : C} \qquad \dfrac{\dfrac{\vdots}{\Delta \vdash v : 0}}{x : C, \Sigma, \Delta \vdash \text{ let } v \text{ be } 0 \text{ in } u : A}{\scriptstyle 0L}}{\Gamma, \Sigma, \Delta \vdash (\text{ let } v \text{ be } 0 \text{ in } u)[t/x] : A}{\scriptstyle Subst}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\vdots}{\Delta \vdash v : 0}}{\Gamma, \Sigma, \Delta \vdash (\text{ let } v \text{ be } 0 \text{ in } u)[t/x] : A}{\scriptstyle 0L}$$

This exhausts all the cases.

Thus, we have a procedure which given a proof which ends in **Subst** of degree d, and which has no applications of **Subst** in the proof of either hypothesis of degree greater than or equal to d, produces a proof of degree less than d.

LEMMA 31 (LOWER-DEGREE-SUBSTS) If a sequent is provable in NAT with a proof of degree $d > 0$, then it is provable in NAT with a proof of degree less than d.

Proof 30 By induction on the height of the derivation tree of the conclusion. We show that given any proof of degree d of $\Gamma \vdash t:A$ in NAT, we may find a (possibly much larger) proof of $\Gamma \vdash t:A$ in NAT of degree less than d.

We examine the proof of $\Gamma \vdash t:A$. Since the degree of this proof is greater than zero, there must be some **Subst** in the proof. If the last rule is not **Subst**, then by induction we may form proofs of its hypotheses of degree less than d. Applying the same rule to the resulting reduced degree hypotheses produces the desired proof of degree less than d.

In the case that the last rule is **Subst**, we have the following situation for some Σ and Δ which together (in multiset union) make up Γ:

$$\frac{\begin{array}{c}\vdots\\ \Sigma \vdash u:B\end{array} \qquad \begin{array}{c}\vdots\\ \Delta, x:B \vdash v:A\end{array}}{\Gamma \vdash v[u/x]:A}\,{\scriptstyle Subst} \qquad \text{where } \Sigma \cup \Delta = \Gamma \ \text{ and } v[u/x] = t$$

By induction, we can produce proofs of $\Sigma \vdash u:B$ and $\Delta, x:B \vdash v:A$ of degree less than d. By a single application of Lemma 30 to the resulting proof constructed from the modified hypotheses, we obtain a proof of $\Gamma \vdash t:A$ of degree less than d.

THEOREM 26 (NAT SUBST-ELIMINATION) If a sequent is provable in NAT, then it is provable in NAT without using the **Subst** rule.

Proof 31 By induction on the degree of the assumed proof. We may apply Lemma 31 at each inductive step, and at the base case the degree of the proof is zero, so therefore by definition of proof degree there are no substs, and we have our desired subst-free proof.

8.3.5 Equivalence of seq and nat

In the last two subsections, we presented two systems intended for the automatic inference of linear type information for linear terms. In the following Theorem 27, we prove that the two systems are equivalent, that is, that any linear type judgement achievable in one system is also achievable in the other, up to reductions of let_{cut}, which marks occurrences of the unrestricted cut rule in NAT.

We should emphasize that although NAT and SEQ are equivalent (up to let_{cut} reduction) in the sense that any typing judgement provable in SEQ is provable in NAT and vice-versa, they are not equivalent with respect to operations on proofs. In particular, cut-elimination is of hyperexponential complexity in SEQ, and subst-elimination is of polynomial complexity in NAT. This is explained by the fact that all the left rules of NAT have an "implicit cut" incorporated into them. As a result, cut-elimination in SEQ does not correspond to subst-elimination in NAT, nor is there a direct connection between subst-elimination in NAT and reduction in the linear calculus. Instead, in NAT one focuses on the elimination of introduction/elimination pairs as the model of computation.

THEOREM 27 (SEQ EQUIV NAT) A type sequent $\Gamma \vdash t : A$ is provable in SEQ if and only $\Gamma \vdash t : A$ is provable in NAT.

Proof 32 One can show that each rule in SEQ is derivable in NAT, and vice versa, using local transformations. All the right rules, identity, are the same in both systems, and **Cut** and **Subst** take the same form. For most left rules, one may simulate the NAT version of the rule in SEQ with one application of the rule of similar name and one application of **Cut**. One may simulate the SEQ version of most left rules in NAT by using the rule of the same name and identity. The multi-hypothesis **!SR** rule of NAT is derivable in SEQ with the use of **!SR** and multiple instances of **Cut**.

One may transform instances of the **Cut** rule in SEQ as applications of **Subst**, which is a derivable rule in NAT. Upon removal of **Subst**, one may see that **Cut** in SEQ corresponds to introduction-elimination pairs of rules in NAT.

There is a somewhat looser correspondence between NAT2 and SEQ, than that just claimed between NAT and SEQ.

A term t' is *related* to a term t if t can be obtained from t' by replacing occurrences $let_{cut} x$ `be` t `in` v with $v[t/x]$.

THEOREM 28 (SEQ EQUIV NAT2) A type sequent $\Gamma \vdash t : A$ is provable in SEQ if and only $\Gamma \vdash t' : A$ is provable in NAT2 for some t' related to t.

This theorem may be proven in the same manner as the above, although in some cases the extra syntax of let_{cut} is used in the NAT2 term t'. The reason for let_{cut} is that **Subst** is not a derived rule of NAT2.

We now turn our attention to the main technical differences between the three sets of typing rules. If we imagine searching for a cut-free proof of a typing derivation, beginning with a prospective conclusion and progressing toward appropriate instances of the axioms, our search will be driven by the form of the term in NAT2 and the form of the type in SEQ. To state this precisely, we begin by reviewing the routine definitions of type subformula and term subformula.

A type A is a *type subformula* of a type B if A syntacticly occurs in B. Similarly, a term t is a *term subformula* of a term s if t syntacticly occurs in s.

LEMMA 32 (NAT2 TERM SUBFORMULA PROPERTY) In any proof of $\vdash u : B$ in NAT2, every term t that appears anywhere in the proof is a subformula of u.

Note that the system NAT fails to have this property because of the form of the **!SR** rule.

LEMMA 33 (SEQ TYPE SUBFORMULA PROPERTY) For any cut-free proof of $\vdash t : B$ in SEQ, any type A that appears anywhere in the proof is a subformula of B.

Note that the cut rule violates the subformula property, and so Lemma 33 does not hold of SEQ proofs with cut.

8.4 Most General Linear Type

8.4.1 Most General Types in nat and seq

In this section, we show that every linearly typable term has a finite set of most general types. This set may be used to decide the set of types of the term. More specifically, a linear formula, A, is *more general* than another, B, if there exists a substitution σ mapping linear propositions to linear formulas such that $(A)\sigma = B$. A set, S, of formulas is more general than another, T, if every element of T is a substitution instance of some element of S. Given a term t, the typing algorithm either returns a finite set of formulas more general than all types of t, or terminates with *failure* if t has no linear type. The number of formulas in the set of most general types is bounded by an exponential function of the number of uses of `store` in the term. Without `store`, every typable term has a single most general type.

Up to **!SR**, the rules of NAT that are used in a typing derivation, and the order of application, are totally determined by the syntactic structure of the linear term. For example, if the term is a variable then the only possible proof in NAT is one use of identity. If the term is $\lambda x.t$, then the only possible rule is $\multimap$ **R**. The only freedom, except for **!SR**, is in the choice of linear types for variables and the division of a type multiset among hypotheses (in the rules with multiple hypotheses). However, type judgements contain exactly the set of free variables of a term in the type context. This property determines the division of a multiset among hypotheses of the rule. Thus, for NAT without **!SR**, we may compute the most general typing by a simple Prolog program, obtained by translating the typing rules into Horn clauses in a straightforward manner.

An example that shows the complications associated with terms of the form `store` t is

$\lambda a.\lambda b.$ `store` $(($ `read` `store` c `as` a `in` $c)\, b)$

which has the two incomparable NAT and SEQ types

$!(!B \multimap C) \multimap !B \multimap !C$
$!(B \multimap !C) \multimap B \multimap !!C$

The basic idea is very similar to the example in Section 8.2 that involves implicit `store`. If a :$!A$, then the expression (`read` `store` c `as` a `in` c) has type A. This gives us the two typing judgments

$a :!(!B \multimap C),\, b :!B \vdash$ (`read` `store` c `as` a `in` $c)\, b : C$
$a :!(B \multimap !C),\, b : B \vdash$ (`read` `store` c `as` a `in` $c)\, b : !C$

The common feature of these judgments is that they both allow us to apply ! to the expression, in the first case because all of the types of free variables begin with ! and in the second case because the type of the expression begins with !. In NAT, the two typings of the expression with `store`are derived using two different substitutions in the **!SR**, rule. In SEQ, the two typings are derived using **Cut** to substitute into a `store` expression in two different ways.

Accounting for the possibility of several different substitutions in the **!SR** rules (some of which are provably unnecessary), all of the NAT rules are straightforward syntax-directed rules that may be translated into Prolog Horn clauses without complication. This gives us an algorithm that finds a finite set of most general types for each linearly typeable term or (since the search is bounded) terminates with failure on untypeable terms.

THEOREM 29 (MGT) Every NAT typeable term has a finite set of most general types. There is a unification-based algorithm that, given any term, either computes a set of most general types or halts with *failure* if the term is not typeable.

8.4.2 Most General Types in nat2

In the simplified NAT2 system, **!SR** is replaced by a simple syntax-directed rule with no possibility of substitution. Consequently, the most general type of any typeable linear term may be computed by a unification-based algorithm or simple Prolog program. The following most general typing theorem is due independently to Mackie [112].

THEOREM 30 (MGT) There is a unification-based algorithm that computes a most general linear type for any NAT2 typeable term t and terminates in *failure* on any untypeable term.

Proof Sketch. If t is NAT2 typeable, then it has a type judgment in NAT2 ending in a sequent $\vdash t : A$ for some type A. This type judgment is unique up to the instantiation of types at the axioms. The most general type will be found by unification, where fresh type variables are initially used at each application of identity. ∎

It is well known that one may view a (traditional, well-typed) functional program as a proof notation [64] for intuitionistic logic. In this light, the type of a program is the conclusion of the proof which it represents. In our present context, we view a linear program as proof notation for intuitionistic linear logic. Thus one could view this procedure as an intuitionistic linear logic proof checker, which also computes the most general conclusion which may be drawn from the given proof.

Noting that the type rules for NAT2 may be written as horn clauses, we have implemented NAT2 in Prolog. The side conditions on the rules may easily be encoded in Prolog. For example the **!SR** rule requires a procedure which ensures that element of the type context has ! type (`bangify`). The type system implemented in this manner has the wonderful property that given a linear term, no search is required to discover it's proof, if one exists, and the entire type checking may be performed in low order polynomial time.

In the Prolog implementation of NAT2 most of which is shown in Figures 8.2 and 8.3, there is exactly one clause for each inference rule of NAT2. There are also predicates defining `append`, `remove`, `notmember`, and `bangify`, although only `bangify` is given in the figure. We give the code implementing NAT2 for the $\otimes, \multimap, !$ fragment of the logic. For this implementation to be sound, the Prolog implementation would have to provide a sound unification procedure (with occurs-check).

```
nat([ type(var(X), A) ], var(X), A).
nat(Context, apply(U, T), B) :-
        nat(Delta, U, la(A, B)),
        nat(Sigma, T, A),
        append(Delta, Sigma, Context).

nat(Sigma, lambda(var(X), T), la(A, B)) :-
        nat(Sigma1, T, B),
        remove(type(var(X), A), Sigma1, Sigma),
        notmember(type(var(X),Any), Sigma).

nat(Context, let(U, times(var(X), var(Y)), T), C) :-
        nat(Delta, U, tensor(A, B)),
        nat(Sigma1, T, C),
        remove(type(var(X), A), Sigma1, Sigma2),
        remove(type(var(Y), B), Sigma2, Sigma3),
        append(Delta, Sigma3, Context).

nat(Context, times(U, T), tensor(A, B)) :-
        nat(Sigma, U, A),
        nat(Gamma, T, B),
        append(Sigma, Gamma, Context).

nat(Context, let(U, var(X), T), B) :-
        nat(Delta, U, A),
        nat(Sigma1, T, B),
        remove(type(var(X), A), Sigma1, Sigma2),
        append(Delta, Sigma2, Context).
```

Figure 8.2
Prolog Implementation of the $\otimes, \multimap$ fragment of NAT2

```
nat(Context, discard(U, T), B) :-
        nat(Delta, U, ofcourse(A)),
        nat(Sigma, T, B),
        append(Delta, Sigma, Context).

nat(Context, copy(var(X), var(Y), U, T), B) :-
        nat(Delta, U, ofcourse(A)),
        nat(S1, T, B),
        remove(type(var(X), ofcourse(A)), S1, S2),
        remove(type(var(Y), ofcourse(A)), S2, S3),
        append(Delta, S3, Context).

nat(Context, read(store(var(X)), U, T), B) :-
        nat(Delta, U, ofcourse(A)),
        nat(Sigma1, T, B),
        remove(type(var(X), A), Sigma1, Sigma2),
        append(Delta, Sigma2, Context).

nat(Sigma, store(T), ofcourse(A)) :-
        nat(Sigma, T, A),
        bangify(Sigma).

nat(Context, let(U, 1, T), A) :-
        nat(Delta, U, 1),
        nat(Sigma, T, A),
        append(Delta, Sigma, Context).

bangify([]).
bangify([ type(var(X), ofcourse(A)) | Rest]) :-
        bangify(Rest).
```

Figure 8.3
Prolog Implementation of the ! fragment of NAT2

8.5 Type Soundness

In this section we prove a technical property commonly called "subject reduction" for both NAT and SEQ. This property is that if linear term t has type A, and t reduces to t', then t' also has type A. The term t' may also have other types; rewriting a term may allow us to deduce more typing properties. However, if the typing rules allows us to derive some property of a term, this property remains as we reduce, or evaluate, the term. Without the subject reduction property, it might be possible for a typed term to become untypeable during execution. In this case, we would consider the type system "unsound" as a method for determining the absence of type errors.

We prove subject reduction by considering NAT first and then deriving the result for SEQ as a corollary. The main reason that the proof is simpler for NAT can be seen by comparing the $\multimap$**L** rule of SEQ with the **A** (application) rule of NAT. When an application $(\lambda x.t)u$ is β-reduced, the structure of the NAT rules guarantee that this was typed by the **A** rule and t was typed subject to some hypothesis about the type of variable x. The same type may be given to $t[u/x]$ using a substitution instance of this proof. In SEQ, we would need a series of detailed lemmas giving us some information about the possible structure of the typing proof for $(\lambda x.t)u$. In particular, since the sequent rule $\multimap$**L** allows arbitrary substitution, the structure of the typing proof is not determined by the form of an application.

The following lemma is the key step in the inductive proof of Theorem 31. It covers the case of the very general **!SR** rule, essentially stating that for one-step linear reduction, terms of ! type do not interact with any other terms. Note that the reduction relation $\rightarrow_0^{\mathcal{C}}$ does not include any of the ! reductions.

LEMMA 34 If $t = r[s_1/x_1, \cdots, s_n/x_n]$, and $t \rightarrow_0^{\mathcal{C}} u$ and for $1 \leq i \leq n : \Delta_i \vdash s_i :!B_i$ then $(u = r'[s_1/x_1, \cdots, s_n/x_n]$ and $r \rightarrow_0^{\mathcal{C}} r')$ or
$(\exists j : u = r[s_1/x_1, \cdots, s'_j/x_j, \cdots, s_n/x_n]$ and $s_j \rightarrow_0^{\mathcal{C}} s'_j)$.

THEOREM 31 (NAT SUBJECT REDUCTION) If there is a proof of $\vdash t :A$ in NAT, and $t \twoheadrightarrow s$, then there is a proof of $\vdash s :A$ in NAT.

Proof 33 Induction on the derivation $t \twoheadrightarrow s$.

COROLLARY 8.5.1 If there is a proof of $\vdash t :A$ in SEQ, and $t \twoheadrightarrow s$, then there is a proof of $\vdash s :A$ in SEQ.

The stronger property that if there is any typing proof of $\Sigma \vdash t : A$ for term t with free variables, and $t \twoheadrightarrow s$, then there is a typing proof of $\Sigma \vdash s : A$ does not hold. As we have seen, control over evaluation order is of critical importance in maintaining linear type soundness, as the following example demonstrates.

$x : !A \vdash$ `discard` x `in` 1

The above judgment is provable in SEQ, NAT, and NAT2, but after one step of reduction, the judgment becomes $x : !A \vdash 1$ which is not provable in any type system discussed in this chapter. Chirimar, Gunter, and Riecke have also noticed this failure of subject reduction for open terms [41].

On the other hand, we do have this more general form of subject reduction of the reflexive transitive closure of $\rightarrow_0$. That is, we may reduce using $\rightarrow_0$ anywhere in a term and still preserve the types.

With a slight modification of the systems we are working with, an intermediate form of these subject reduction theorems is possible. If Γ and Σ are multisets, then we write $\Sigma \subseteq \Gamma$ to mean that Σ may by obtained from Γ by removing elements or adding duplicates. The following theorem holds in a version of these type systems where the restriction that every variable occur exactly once in binding and once in use is relaxed to the restriction that every variable occur as many times in binding as it occurs in use, and all occurrences of a variable have the same type.

THEOREM 32 (GENERALIZED SUBJECT REDUCTION) If there is a proof of $!\Gamma \vdash t : A$ in NAT or SEQ, and $t \twoheadrightarrow s$, then there is a proof of $!\Sigma \vdash s : A$ in NAT and SEQ, where $!\Sigma \subseteq !\Gamma$.

Also, a weaker form of subject reduction theorem holds for NAT2. The restrictions on reduction order are sufficient to guarantee that whenever $(\lambda x.t)u$ is reduced, under the conditions given in the theorem below, $t[u/x]$ has the same type.

THEOREM 33 If there is a proof of $\vdash t : A$ in NAT2, and $t \twoheadrightarrow s$, then there is a proof of $\vdash s' : A$ in NAT2, for some term s' related to s.

8.6 Alternate Term Syntax

Several term assignments other than those described above due to Abramsky [2] and Lincoln and Mitchell [102] have been proposed for linear logic. Wadler has proposed several possibilities along with categorical and practical motivation for his choices [152, 153, 151, 150, 149]. Wakeling and Runciman [155] and Baker [16] propose specific term calculi. Benton et al study a system very close to NAT2 with additional syntax embedded in the **!SR** rule. Their system exhibits a unique most general type property, and has additional desirable properties [31]. However, this system requires additional "commuting conversions" to identify terms that differ only in the arbitrary order of hypotheses in promotion. Troelstra [143] presents a variant that overcomes the need for commuting conversions but where terms do not uniquely determine deductions. System NAT2 above also has this property. However, Troelstra's system appears not to admit a nice normal form of deductions. Mints [123] presents a term assignment with similar properties.

8.7 Implementation

There are several alternative approaches to the implementation of linear lambda calculus.

One problem with Lafont's method of eliminating garbage collection is that it requires a tremendous amount of duplication. Essentially, in comparison with a standard reference counting scheme, garbage collection is eliminated by making every datum have reference count one. This is achieved by copying the datum whenever we would otherwise increment the reference count. A consequence is that there is a significant increase in the amount of storage space required. We believe that in practice, it is useful to consider the trade-offs between copying and garbage collection. In particular, if a datum is large, then copying it even a small number of times may be prohibitive, and may outweigh the benefit of suspending garbage collection.

Several authors have considered the tradeoffs in this area. Wakeling and Runciman [155] study linear modifications to the G-Machine [78], and suggest studying the spineless G-Machine. Lincoln and Mitchell [102] consider an implementation based on the TIM, which is closely related to the spineless G-Machine. Wadler [152, 153, 151, 150] gives a motivation for variants of linear languages based monads and gives concrete suggestions for alternative syntax for **!SR** [149, 150, 151]. Benton et al study a system very close to NAT2 with additional syntax embedded in the **!SR** rule [31]. Their system exhibits a unique most general type property, and may have additional desirable properties [32]. In the most well engineered and heavily used such system, Baker [16, 18, 20, 21, 19, 22] proposes a similar memory model and develops useful syntactic shorthand and implementation techniques.

The key point of departure of most implementations of the linear lambda calculus from standard implementations of functional languages is the memory model. The linear version often incorporates two spaces, one linear, and one nonlinear. The idea is that objects in the linear space are purely linear, and thus have a reference count of exactly one at all times. Objects in the nonlinear space represent "stored" or reusable entities. Little or no static information is available about reference counts of objects in this space. The execution model ensures that a ! or `store` instruction (corresponding to the !SR rule of linear logic) allocates objects in nonlinear space. Once an object is stored, it may be discarded or copied, a `discard` operation removes a pointer to a stored object, and a `copy` operation simply copies a pointer to an object in nonlinear space, thus implementing sharing, or call-by-need. However, in this nonlinear space, objects can be referenced any number of times (including 0), requiring some form of dynamic garbage collection. In the linear space, objects are never shared; there is always exactly one reference to all objects. Thus garbage collection is not needed, since objects in that space become garbage the first time they are used, and linear objects may always be updated in place. In other words, dynamic garbage collection is never applied to objects in linear space, but may occasionally be applied to objects in nonlinear space. Update in place is always applicable to linear objects, but is never applied to nonlinear objects.

Wadler [149] and Lincoln and Mitchell [102] explore the possibility of static detection of opportunities for update-in-place of arrays and other mutable data structures. Most such structures contain elements such as built-in datatypes integer and string which are nonlinear and thus can be freely shared. Such data structures are not truly mutable, since they are only allowed to be altered when there are no other pointers to the structure. Thus no "side effects" occur, even when some data structures are mutated. In this approach code is "used-up" by execution, and thus recursive programs must be nonlinear.

Some implementations of the linear calculus have effectively assumed a single memory space. A potential disadvantage of the single memory space is that it obfuscates the distinction between shared and unshared objects. Lafont built an implementation of the linear calculus with the fantastic property that dynamic garbage collection is never used: all terms effectively have exactly one reference to them, and thus become garbage the first (only) time they are referenced. However, Lafont avoids garbage collection by copying, a rather costly implementation technique [68]. Chirimar, Gunter, and Riecke have described an implementation which also focuses on the issue of garbage collection [41]. In their implementation, objects may be shared, so dynamic garbage collection is potentially required on all objects. However, the linear types of terms may be used to identify potential times at which objects may become garbage. Their implementation does not include the additives, but is extended with a recursion operator and polymorphism.

Abramsky has described the implementation of a linear SECD machine further studied by Mackie [2, 112], and went on to generalize the linear calculus to one based on classical linear logic and described an implementation based on the chemical abstract machine [34]. Wadler [151] has also described several implementation issues regarding the linear calculus. He points out the importance of `!(!A)` being isomorphic to `!A` (which is true in the preferred operational model), and suggests several extensions, including, for example, arrays, let! with read-only access, the removal of syntax for weakening and contraction, etc.

Baker [16, 18, 20, 21, 19, 22] presents a series of example programs in linear lisp, a variant of linear lambda calculus. These programs admirably demonstrate the flexibility and potential performance of a statically linearly typed programming language.

8.8 Summary of Chapter

We have presented a linear calculus and three type inference systems: SEQ, NAT, and NAT2. We have shown that SEQ and NAT equivalent, and that NAT2 is closely related. We have demonstrated the existence of most general types and the subject reduction theorem. The linear calculus and very closely related type systems have appeared elsewhere, perhaps most well known in [94, 2].

9 Conclusion

> The outcome of any serious research can only be to make two questions grow where only one grew before THORSTEN VEBLEN 1857-1929
>
> *The place of science in modern civilizations and other essays*, 1919

This book investigates computational aspects of linear logic. The main results of this work support the proposition that "Linear logic is a *computational* logic behind logics." This book augments the proof theoretic framework of linear logic by providing theorems such as permutability, impermutability, and cut-standardization with non-logical theories. On this expanded proof theoretic base, many complexity results are proved using the Girard correspondence between proofs and computations. Among these results are the undecidability of propositional linear logic, the PSPACE-completeness of MALL, and the NP-completeness of the constant-only multiplicative fragment of linear logic. Another application of proof theory to computation is explored for a functional language LM and its (compiled) implementation. The proposed linear type system for LM yields compile-time type information about resource manipulation, which can be used to control aspects of program execution such as storage allocation, garbage collection, and array update in place. Most general type and subject reduction theorems are proved, and relations between this and related work is briefly drawn.

In more detail, we demonstrate the power of the Girard correspondence between linear logic proofs and computation, establishing the complexity of the decision problem in several fragments of linear logic. There had been little previous work on decision problems in linear logic, and full propositional linear logic had been suspected to be decidable. Using the Girard correspondence, this book connects full propositional linear logic and counter machines in much the same way that a fragment of linear logic had been connected to the Petri net reachability problem. These connections between linear logic and machines shows how to read proofs as descriptions of (successful) computations. This approach yields a proof that propositional linear logic is undecidable. Other computationally interesting classes are also naturally represented in fragments of linear logic: semi-Thue systems may be embedded in noncommutative linear logic, classical quantified boolean formulas may be encoded in MALL, showing that MALL is PSPACE-complete, and 3-Partition can be encoded in constant-only multiplicative linear logic, showing that constant-only MLL is NP-complete.

Finally, exploring an additional computational interpretation of linear logic based on the Curry-Howard isomorphism, this book reviews a line of research begun early in the history of linear logic. A proof of the subject-reduction (type-soundness) and the most general type theorems are presented for LM, a language related to those studied by others [94, 2, 150, 151, 41, 112, 31]. This book pushes proof theory a little further further into compilation, providing a sound theoretical basis for some compiler optimizations which are currently performed in an ad hoc manner.

For future directions, the complexity of a few interesting decision problems for linear logic is still unknown. Without additives, multiplicative exponential linear logic (MELL) can encode Petri-net reachability problems, which are known to be EXPSPACE-hard, but decidable [110, 118, 117, 116, 93]. In general the decidability of the decision problem for MELL is open, although some small advances have been made in this direction. There are many remaining open problems in related logics. The exact complexity of noncommutative MLL remains a tricky issue (is this fragment in NP?), as do some problems regarding affine logic (linear logic with weakening) and direct logic (linear logic with contraction).

Establishing close connections between intuitive computational models and logical systems is a promising approach for the study of proof theory and for the study of more mainstream computer science. Indeed, linear logic has been introduced only recently and the body of work connecting theoretical computer science and linear logic is already quite large. This book contributes to this exciting application of logic in computer science, showing that linear logic is not about "Truth"; it is about computation.

begin at the beginning...and go on til you come to the end: then stop LEWIS CARROL (CHARLES LUTWIDGE DODSON) 1832-1898
Alices Adventures in Wonderland, 1865

A Formal Systems

Here a review of some of the formal systems considered in this book. These are provided here to facilitate future reference and the comparison of differing systems.

A.1 Linear Logic Sequent Calculus Rules

I	$\dfrac{}{A \vdash A}$	$\dfrac{\Gamma_1 \vdash A, \Sigma_1 \quad \Gamma_2, A \vdash \Sigma_2}{\Gamma_1, \Gamma_2 \vdash \Sigma_1, \Sigma_2}$	**Cut**
E L	$\dfrac{\Gamma_1, A, B, \Gamma_2 \vdash \Sigma}{\Gamma_1, B, A, \Gamma_2 \vdash \Sigma}$	$\dfrac{\Gamma \vdash \Sigma_1, A, B, \Sigma_2}{\Gamma \vdash \Sigma_1, B, A, \Sigma_2}$	**E R**
$\otimes$ **L**	$\dfrac{\Gamma, A, B \vdash \Sigma}{\Gamma, (A \otimes B) \vdash \Sigma}$	$\dfrac{\Gamma_1 \vdash A, \Sigma_1 \quad \Gamma_2 \vdash B, \Sigma_2}{\Gamma_1, \Gamma_2 \vdash (A \otimes B), \Sigma_1, \Sigma_2}$	$\otimes$ **R**
$\parr$ **L**	$\dfrac{\Gamma_1, A \vdash \Sigma_1 \quad \Gamma_2, B \vdash \Sigma_2}{\Gamma_1, \Gamma_2, (A \parr B) \vdash \Sigma_1, \Sigma_2}$	$\dfrac{\Gamma \vdash A, B, \Sigma}{\Gamma \vdash (A \parr B), \Sigma}$	$\parr$ **R**
$\oplus$ **L**	$\dfrac{\Gamma, A \vdash \Sigma \quad \Gamma, B \vdash \Sigma}{\Gamma, (A \oplus B) \vdash \Sigma}$	$\dfrac{\Gamma \vdash A, \Sigma \quad \Gamma \vdash B, \Sigma}{\Gamma \vdash (A \& B), \Sigma}$	& **R**
& **L1**	$\dfrac{\Gamma, A \vdash \Sigma}{\Gamma, (A \& B) \vdash \Sigma}$	$\dfrac{\Gamma \vdash A, \Sigma}{\Gamma \vdash (A \oplus B), \Sigma}$	$\oplus$ **R1**
& **L2**	$\dfrac{\Gamma, B \vdash \Sigma}{\Gamma, (A \& B) \vdash \Sigma}$	$\dfrac{\Gamma \vdash B, \Sigma}{\Gamma \vdash (A \oplus B), \Sigma}$	$\oplus$ **R2**
! W	$\dfrac{\Gamma \vdash \Sigma}{\Gamma, !A \vdash \Sigma}$	$\dfrac{\Gamma, !A, !A \vdash \Sigma}{\Gamma, !A \vdash \Sigma}$	**! C**
! D	$\dfrac{\Gamma, A \vdash \Sigma}{\Gamma, !A \vdash \Sigma}$	$\dfrac{!\Gamma \vdash A, ?\Sigma}{!\Gamma \vdash !A, ?\Sigma}$	**! S**
? W	$\dfrac{\Gamma \vdash \Sigma}{\Gamma \vdash ?A, \Sigma}$	$\dfrac{\Gamma \vdash ?A, ?A, \Sigma}{\Gamma \vdash ?A, \Sigma}$	**? C**
? D	$\dfrac{\Gamma \vdash A, \Sigma}{\Gamma \vdash ?A, \Sigma}$	$\dfrac{!\Gamma, A \vdash ?\Sigma}{!\Gamma, ?A \vdash ?\Sigma}$	**? S**
0 **L**	$\Gamma, 0 \vdash \Sigma$	$\Gamma \vdash \top, \Sigma$	$\top$ **R**
1 **L**	$\dfrac{\Gamma \vdash \Sigma}{\Gamma, 1 \vdash \Sigma}$	$\dfrac{\Gamma \vdash \Sigma}{\Gamma \vdash \bot, \Sigma}$	$\bot$ **R**
$\bot$ **L**	$\bot \vdash$	$\vdash 1$	1 **R**

A linear logic sequent is composed of two sequences of linear logic formulas separated by a $\vdash$. In most proofs, sequents are assumed to be constructed from *multisets* of linear logic formulas, effectively ignoring applications of the exchange rule.

The following notational conventions are used.

p_i	Positive propositional literal
$p_i^{\perp}$	Negative propositional literal
A, B, C	Arbitrary formulas
$\Sigma, \Gamma, \Delta, \Theta$	Arbitrary sequences of formulas
$\otimes$	Tensor, the multiplicative conjunction
1	One, the unit of tensor
$⅋$	Par, the multiplicative disjunction
$\perp$	Bottom, the unit of par
$\&$	With, the additive conjunction
$\top$	Top, the unit of with
$\oplus$	Plus, the additive disjunction
0	Zero, the unit of plus

A.2 Prop. Intuitionistic Linear Logic

I	$\dfrac{}{A \vdash A}$	$\dfrac{\Sigma \vdash A \quad A, \Gamma \vdash B}{\Sigma, \Gamma \vdash B}$	**Cut**
E L	$\dfrac{\Gamma_1, A, B, \Gamma_2 \vdash C}{\Gamma_1, B, A, \Gamma_2 \vdash C}$		
$\otimes$**L**	$\dfrac{\Sigma, A, B \vdash C}{\Sigma, (A \otimes B) \vdash C}$	$\dfrac{\Sigma \vdash A \quad \Gamma \vdash B}{\Sigma, \Gamma \vdash (A \otimes B)}$	$\otimes$**R**
$\multimap$**L**	$\dfrac{\Sigma \vdash A \quad \Gamma, B \vdash C}{\Sigma, \Gamma, (A \multimap B) \vdash C}$	$\dfrac{\Sigma, A \vdash B}{\Sigma \vdash (A \multimap B)}$	$\multimap$**R**
$\oplus$**L**	$\dfrac{\Sigma, A \vdash C \quad \Sigma, B \vdash C}{\Sigma, (A \oplus B) \vdash C}$	$\dfrac{\Sigma \vdash A \quad \Sigma \vdash B}{\Sigma \vdash (A \,\&\, B)}$	**& R**
& L1	$\dfrac{\Sigma, A \vdash C}{\Sigma, (A \,\&\, B) \vdash C}$	$\dfrac{\Sigma \vdash A}{\Sigma \vdash (A \oplus B)}$	$\oplus$**R1**
& L2	$\dfrac{\Sigma, B \vdash C}{\Sigma, (A \,\&\, B) \vdash C}$	$\dfrac{\Sigma \vdash B}{\Sigma \vdash (A \oplus B)}$	$\oplus$**R2**
!WL	$\dfrac{\Sigma \vdash A}{\Sigma, !B \vdash A}$		
!DL	$\dfrac{\Sigma, A \vdash B}{\Sigma, !A \vdash B}$	$\dfrac{\Sigma \vdash A}{\Sigma \vdash ?A}$	**?DR**
!CL	$\dfrac{\Sigma, !A, !A \vdash B}{\Sigma, !A \vdash B}$		
?SL	$\dfrac{!\Sigma, A \vdash ?B}{!\Sigma, ?A \vdash ?B}$	$\dfrac{!\Sigma \vdash A}{!\Sigma \vdash !A}$	**!SR**
1**L**	$\dfrac{\Sigma \vdash A}{\Sigma, 1 \vdash A}$	$\dfrac{}{\vdash 1}$	1**R**
0**L**	$\dfrac{}{\Sigma, 0 \vdash A}$	$\dfrac{}{\Sigma \vdash \top}$	$\top$**R**

An intuitionistic linear logic sequent is composed of a multiset of linear logic formulas, and a single formula separated by a $\vdash$. The intuitionistic proof system is restricted so that there is no way to derive a sequent where the multiset on the right of the $\vdash$ contains more than one element.

A.3 Linear Calculus

The notation $t \twoheadrightarrow u$ is meant to be read "t evaluates in any number of steps to u." As usual, the following rules are universally quantified over terms t, u, v, and variables x, y, z.

$$\frac{t \twoheadrightarrow \lambda x.v \qquad u \twoheadrightarrow c \qquad v[c/x] \twoheadrightarrow d}{(tu) \twoheadrightarrow d}$$

$$\frac{t \twoheadrightarrow (c \times d) \qquad u[c/x, d/y] \twoheadrightarrow e}{\text{let } x \times y \text{ be } t \text{ in } u \twoheadrightarrow e} \qquad \frac{t \twoheadrightarrow c \qquad u \twoheadrightarrow d}{(t \times u) \twoheadrightarrow (c \times d)}$$

$$\frac{t \twoheadrightarrow inl(c) \qquad u[c/x] \twoheadrightarrow d}{\text{case } t \text{ of } inl(x) \Rightarrow u, inr(y) \Rightarrow v \twoheadrightarrow d}$$

$$\frac{t \twoheadrightarrow inr(c) \qquad v[c/y] \twoheadrightarrow d}{\text{case } t \text{ of } inl(x) \Rightarrow u, inr(y) \Rightarrow v \twoheadrightarrow d}$$

$$\frac{t \twoheadrightarrow \langle v, w\rangle \qquad v \twoheadrightarrow c \qquad u[c/x] \twoheadrightarrow d}{\text{let } \langle x, \_\rangle \text{ be } t \text{ in } u \twoheadrightarrow d} \qquad \frac{t \twoheadrightarrow c}{inl(t) \twoheadrightarrow inl(c)}$$

$$\frac{t \twoheadrightarrow \langle v, w\rangle \qquad w \twoheadrightarrow c \qquad u[c/y] \twoheadrightarrow d}{\text{let } \langle \_, y\rangle \text{ be } t \text{ in } u \twoheadrightarrow d} \qquad \frac{t \twoheadrightarrow c}{inr(t) \twoheadrightarrow inr(c)}$$

$$\frac{}{t \twoheadrightarrow t} \qquad \frac{t \twoheadrightarrow u \qquad u \twoheadrightarrow v}{t \twoheadrightarrow v}$$

$$\frac{u \twoheadrightarrow c}{\text{discard } t \text{ in } u \twoheadrightarrow c} \qquad \frac{t \twoheadrightarrow c \quad u[c/x] \twoheadrightarrow d}{let_{cut}\, x \text{ be } t \text{ in } u \twoheadrightarrow d}\ let_{cut}$$

$$\frac{t \twoheadrightarrow !v \qquad v \twoheadrightarrow c \qquad u[c/x] \twoheadrightarrow d}{\text{read } !x \text{ as } t \text{ in } u \twoheadrightarrow d}$$

$$\frac{u[t/x, t/y] \twoheadrightarrow c}{\text{copy } x@y \text{ as } t \text{ in } u \twoheadrightarrow c}$$

$$\frac{t \twoheadrightarrow 1 \qquad u \twoheadrightarrow c}{\text{let } 1 \text{ be } t \text{ in } u \twoheadrightarrow c} \qquad \frac{}{1 \twoheadrightarrow 1}$$

A.4 SEQ Proof Rules

An SEQ sequent is composed of a type context, a $\vdash$, a linear term, and a linear type. The informal meaning of a sequent is that if one assumes the types of variables given by the type context, then the linear term has the given linear type.

I $$\frac{}{x:A \vdash x:A}$$

Cut $$\frac{\Sigma \vdash t:A \quad x:A, \Gamma \vdash u:B}{\Sigma, \Gamma \vdash u[t/x]:B}$$

E Left $$\frac{\Sigma, x:A, y:B, \Delta \vdash u:A}{\Sigma, y:B, x:A, \Delta \vdash u:A}$$

$\otimes$L $$\frac{\Sigma, x:A, y:B \vdash t:C}{\Sigma, z:(A \otimes B) \vdash \text{ let } z \text{ be } (x \times y) \text{ in } t:C}$$

$\otimes$R $$\frac{\Sigma \vdash t:A \qquad \Gamma \vdash u:B}{\Sigma, \Gamma \vdash (t \times u):(A \otimes B)}$$

$\multimap$L $$\frac{\Sigma \vdash t:A \qquad \Gamma, x:B \vdash u:C}{\Sigma, \Gamma, f:(A \multimap B) \vdash u[(ft)/x]:C}$$

$\multimap$R $$\frac{\Sigma, x:A \vdash t:B}{\Sigma \vdash \lambda x.t:(A \multimap B)}$$

$\oplus$L $$\frac{\Sigma, x:A \vdash u:C \qquad \Sigma, y:B \vdash v:C}{\Sigma, z:(A \oplus B) \vdash \text{ case } z \text{ of } inl(x) \Rightarrow u, inr(y) \Rightarrow v:C}$$

& R $$\frac{\Sigma \vdash t:A \qquad \Sigma \vdash u:B}{\Sigma \vdash \langle t, u \rangle:(A \,\&\, B)}$$

& L1 $$\frac{\Sigma, x:A \vdash t:C}{\Sigma, z:(A \,\&\, B) \vdash \text{ let } z \text{ be } \langle x, \_ \rangle \text{ in } t:C}$$

$\oplus$R1 $$\frac{\Sigma \vdash t:A}{\Sigma \vdash inl(t):(A \oplus B)}$$

& L2 $$\frac{\Sigma, y:B \vdash t:C}{\Sigma, z:(A \,\&\, B) \vdash \text{ let } z \text{ be } \langle \_, x \rangle \text{ in } t:C}$$

$\oplus$R2 $$\frac{\Sigma \vdash u:B}{\Sigma \vdash inr(u):(A \oplus B)}$$

$$\textbf{!WL} \quad \frac{\Sigma \vdash t : A}{\Sigma, z : !B \vdash \text{ let } z \text{ be } \_ \text{ in } t : A}$$

$$\textbf{!DL} \quad \frac{\Sigma, x : A \vdash t : B}{\Sigma, z : !A \vdash \text{ let } z \text{ be } !x \text{ in } t : B}$$

$$\textbf{?DR} \quad \frac{\Sigma \vdash t : A}{\Sigma \vdash ?t : ?A}$$

$$\textbf{!CL} \quad \frac{\Sigma, x : !A, y : !A \vdash t : B}{\Sigma, z : !A \vdash \text{ let } z \text{ be } x@y \text{ in } t : B}$$

$$\textbf{?SL} \quad \frac{!\Sigma, x : A \vdash t : ?B}{!\Sigma, z : ?A \vdash \text{ let } z \text{ be } ?x \text{ in } t : ?B}$$

$$\textbf{!SR} \quad \frac{!\Sigma \vdash t : A}{!\Sigma \vdash !t : !A}$$

$$1\textbf{L} \quad \frac{\Sigma \vdash t : A}{\Sigma, z : 1 \vdash \text{ let } z \text{ be } 1 \text{ in } t : A}$$

$$1\textbf{R} \quad \frac{}{\vdash 1 : 1}$$

$$0\textbf{L} \quad \frac{}{\Sigma, z : 0 \vdash \text{ let } z \text{ be } 0 \text{ in } t : A}$$

$$\top\textbf{R} \quad \frac{}{\Sigma \vdash \top : \top}$$

A.5 NAT Proof Rules

I $$\frac{}{x:A \vdash x:A}$$

Subst $$\frac{\Sigma \vdash t:A \quad x:A, \Gamma \vdash u:B}{\Sigma, \Gamma \vdash u[t/x]:B}$$

Exch. Left $$\frac{\Sigma, x:A, y:B, \Delta \vdash u:A}{\Sigma, y:B, x:A, \Delta \vdash u:A}$$

A $$\frac{\Delta \vdash u:(A \multimap B) \qquad \Sigma \vdash t:A}{\Sigma, \Delta \vdash (u\,t):B}$$

$\multimap$R $$\frac{\Sigma, x:A \vdash t:B}{\Sigma \vdash \lambda x.t:(A \multimap B)}$$

$\otimes$L $$\frac{\Delta \vdash u:(A \otimes B) \quad \Sigma, x:A, y:B \vdash t:C}{\Sigma, \Delta \vdash \ \mathtt{let}\ (x \times y)\ \mathtt{be}\ u\ \mathtt{in}\ t:C}$$

$\otimes$R $$\frac{\Sigma \vdash u:A \qquad \Gamma \vdash t:B}{\Sigma, \Gamma \vdash (u \times t):(A \otimes B)}$$

$\oplus$L $$\frac{\Delta \vdash t:(A \oplus B) \quad \Sigma, x:A \vdash u:C \quad \Sigma, y:B \vdash v:C}{\Sigma, \Delta \vdash \ \mathtt{case}\ t\ \mathtt{of}\ inl(x) \Rightarrow u, inr(y) \Rightarrow v:C}$$

$\oplus$R1 $$\frac{\Sigma \vdash t:A}{\Sigma \vdash inl(t):(A \oplus B)}$$

$\oplus$R2 $$\frac{\Sigma \vdash t:B}{\Sigma \vdash inr(t):(A \oplus B)}$$

& L1 $$\frac{\Delta \vdash u:(A \,\&\, B) \qquad \Sigma, x:A \vdash t:C}{\Sigma, \Delta \vdash \ \mathtt{let}\ \langle x, \_\rangle\ \mathtt{be}\ u\ \mathtt{in}\ t:C}$$

& L2 $$\frac{\Delta \vdash u:(A \,\&\, B) \qquad \Sigma, y:B \vdash t:C}{\Sigma, \Delta \vdash \ \mathtt{let}\ \langle \_, y\rangle\ \mathtt{be}\ u\ \mathtt{in}\ t:C}$$

& R $$\frac{\Sigma \vdash t:A \qquad \Sigma \vdash u:B}{\Sigma \vdash \langle t, u\rangle:(A \,\&\, B)}$$

$$\textbf{!WL} \quad \frac{\Delta \vdash u :!A \qquad \Sigma \vdash t :B}{\Sigma, \Delta \vdash \ \texttt{discard}\ u\ \texttt{in}\ t :B}$$

$$\textbf{!DL} \quad \frac{\Delta \vdash u :!A \qquad \Sigma, x :A \vdash t :B}{\Sigma, \Delta \vdash \ \texttt{read}\ \ \texttt{store}\ x\ \texttt{as}\ u\ \texttt{in}\ t :B}$$

$$\textbf{!CL} \quad \frac{\Delta \vdash u :!A \qquad \Sigma, x :!A, y :!A \vdash t :B}{\Sigma, \Delta \vdash \ \texttt{copy}\ x@y\ \texttt{as}\ u\ \texttt{in}\ t :B}$$

$$\textbf{!SR} \quad \frac{\Delta_i \vdash t_i :!B_i \qquad x_i :!B_i, !\Sigma \vdash u :A}{\Delta_1 \cdots \Delta_n, !\Sigma \vdash \ \texttt{store}\ u[t_1/x_1 \cdots t_n/x_n] :!A}$$

$$1\textbf{L} \quad \frac{\Delta \vdash u :1 \qquad \Sigma \vdash t :A}{\Sigma, \Delta \vdash \ \texttt{let}\ 1\ \texttt{be}\ u\ \texttt{in}\ t :A}$$

$$1\textbf{R} \quad \frac{}{\vdash 1 :1}$$

NAT sequent is composed of a type context, a $\vdash$, a linear term, and a linear type. The informal reading of NAT sequents is the same as the reading of SEQ sequents. If one assumes the types of variables given by the type context, then the linear term to the right of the $\vdash$ has the given linear type.

B Bibliography

Bibliography

[1] S. Abramsky. Tutorial on linear logic. Lecture Notes from Tutorial at ILPS, 1991.

[2] S. Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111:3–57, 1993.

[3] S. Abramsky. Interaction categories and typed concurrent programming. In *Proceedings of the International Summer School of Marktoberdorf*, NATO Advanced Science Institutes, series F, 1994.

[4] S. Abramsky and R. Jagadeesan. New foundations for the geometry of interaction. In *Proc. 7-th Annual IEEE Symposium on Logic in Computer Science, Santa Cruz, California*, pages 211–222. IEEE Computer Society Press, Los Alamitos, California, June 1992.

[5] S. Abramsky and S. Vickers. Quantales, observational logic, and process semantics. Preprint, January 1990.

[6] V. Abrusci. Sequent calculus for intuitionistic linear propositional logic. In: Mathematical Logic, ed by P. Petkov.

[7] J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3), 1992.

[8] J.-M. Andreoli and R. Pareschi. Logic programming with sequent systems: a linear logic approach. In *Proc. Workshop on Extensions of Logic Programming, Tuebingen.* Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin, 1990.

[9] J.-M. Andreoli and R. Pareschi. Linear objects: Logical processes with built-in inheritance. *New Generation Computing*, 9, 1991.

[10] P.B. Andrews. On simplifying the matrix of a WFF. *J. Symbolic Logic*, 33(2):180–192, 1968.

[11] P.B. Andrews. Resolution with merging. *J. ACM*, 17(3):367–381, 1968.

[12] A. Asperti. A logic for concurrency. Technical report, Dipartimento di Informatica, Universitá di Pisa, 1987.

[13] A. Asperti, G.-L. Ferrari, and R. Gorrieri. Implicative formulae in the 'proofs as computations' analogy. In *Proc. 17-th ACM Symp. on Principles of Programming Languages, San Francisco*, pages 59–71, January 1990.

[14] A. Avron. The semantics and proof theory of linear logic. *Theoretical Computer Science*, 57:161–184, 1988.

[15] A. Avron. Some properties of linear logic proved by semantic methods. Manuscript, December 1991.

[16] H. G. Baker. Lively linear lisp — 'look ma, no garbage!'. *ACM Sigplan Notices*, 27(8):89–98, Aug. 1992.

[17] H.G. Baker. Unify and conquer (garbage, updating, aliasing, ...) in functional languages. In *Proc. 1990 ACM Conf. on Lisp and Functinoal Programming*, pages 218–226. Association for Computing Machinery, June 1990.

[18] H.G. Baker. The Boyer benchmark meets linear logic. Draft, 1993. Available from the author.

[19] H.G. Baker. Linear logic and permutation stacks — the forth shall be first. Draft, 1993. Available from the author.

[20] H.G. Baker. A linear logic quicksort. Draft, 1993. Available from the author.

[21] H.G. Baker. The linear logic variable — it's not just for theoreticians anymore. Draft, 1993. Available from the author.

[22] H.G. Baker. Sparse polynomials and linear logic. Draft, 1993. Available from the author.

[23] H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* North Holland, 1984.

[24] M. Barr. **-Autonomous Categories*, volume 752 of *Lecture Notes in Mathematics.* Springer-Verlag, Berlin, 1979.

[25] A. Bawden. Connection graphs. In *Proc. ACM Symp. on Lisp and Functional Programming*, pages 258–265, 1986.

[26] G. Bellin. *Mechanizing Proof Theory: Resource-Aware Logics and Proof-Transformations to Extract Implicit Information.* PhD thesis, Stanford University, 1990.

[27] G. Bellin. Proof Nets for Multiplicative-Additive Linear Logic. Draft, 1991.

[28] G. Bellin. Proof-nets for multiplicative and additive linear logic. Technical Report LFCS-91-161, Deparment of Computer Science, University of Edinburgh, May 1991.

[29] G. Bellin and J. Ketonen. A decision procedure revisited: Notes on direct logic, linear logic, and its implementation. *Theoretical Computer Science*, 95:115–142, 1992.

[30] J. Van Bentham. *Language in Action.* North-Holland, 1991.

[31] N. Benton, G. Bierman, J. M. E. Hyland, and V. de Paiva. Term assignment for intuitionistic linear logic. Technical Report 262, Computer Laboratory, University of Cambridge, August 1992.

[32] N. Benton, G. Bierman, J. M. E. Hyland, and V. de Paiva. A term calculus for intuitionistic linear logic. In M. Bezem and J. F. Groote, editors, *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, pages 75–90. Springer-Verlag LNCS 664, 1993.

[33] P. N. Benton. A mixed linear and non-linear logic: Proofs, terms, and models (extended abstract). In L Pacholski and J. Tiuryn, editors, *Proceedings of the 8th Workshop on Computer Science Logic*, number 933 in Lecture Notes in Computer Science, pages 121–135. Springer-Verlag, 1994.

[34] G. Berry and G. Boudol. The chemical abstract machine. In *Proc. 17-th ACM Symp. on Principles of Programming Languages, San Francisco*, pages 81–94, January 1990.

[35] A. Blass. A game semantics for linear logic. *Annals Pure Appl. Logic*, 56:183–220, 1992. Special Volume dedicated to the memory of John Myhill.

[36] R. Blute. Proof nets and coherence theorems. In *Category Theory and Computer Science, Proceedings 1991.* Springer LNCS 530, 1991.

[37] S. Cerrito. A linear semantics for allowed logic programs. In *Proc. 5-th IEEE Symp. on Logic in Computer Science, Philadelphia*, June 1990.

[38] S. Cerrito. Herbrand methods in linear logic. Unpublished Draft, 1992.

[39] A.K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

[40] A.K. Chandra and L.J. Stockmeyer. Alternation. In *Proc. 17th Ann. IEEE Symp. on Foundations of Computer Science*, pages 98–108, 1976.

[41] J. Chirimar, C. Gunter, and J. Riecke. Linear ML. In *Proc. ACM Symposium on Lisp and Functional Programming, San Francisco.* ACM Press, June 1992.

[42] S. A. Cook. The complexity of theorem-proving procedures. In *Proc. 3-rd Annual ACM Symposium on Theory of Computing*, pages 151–158. Association for Computing Machinery, 1971.

[43] P-L. Curien. Games and sequentiality. Linear Logic Mailing List, 1992. linear@cs.stanford.edu.

[44] V. Danos and L. Regnier. The structure of multiplicatives. *Archive for Mathematical Logic*, 28:181–203, 1989.

[45] V.C.V. de Paiva. A dialectica-like model of linear logic. In *Category Theory and Computer Science*, pages 341–356. Springer LNCS 389, September 1989.

[46] J. Dunn. Relevence logic and entailment. In: Handbook of Philosophical Logic III, ed by D. Gabbay and F. Gunther, 1986.

[47] M. Emms. Decision problems for non-commutative linear logic. Draft, 1995.

[48] J. Fairbairn and S. Wray. Tim: A simple, lazy abstract machine to execute supercombinators. In *3rd Conf. on Functional Programming and Computer Architecture, Lecture Notes in Computer Science 274*, New York, 1985. Springer-Verlag.

[49] D.M. Gabbay. *Semantical investigations in Heyting's intuitionistic logic.* Reidel, Dordrecht, 1981.

[50] D. Galmiche and G. Perrier. Automated deduction in additive and multiplicative linear logic. In *Logic at Tver '92, Logical Foundations of Computer Science Symposium*, number 620 in Lecture Notes in Computer Science, pages 151–162. Springer-Verlag, July 1992.

[51] D. Galmiche and G. Perrier. On proof normalization in linear logic. *Theoretical Computer Science*, 1995. Accepted for publication.

[52] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co., 1979.

[53] V. Gehlot and C.A. Gunter. Normal process representatives. In *Proc. 5-th IEEE Symp. on Logic in Computer Science, Philadelphia*, June 1990.

[54] G. Gentzen. Untersuchungen uber das logische Schliessen. *Mathematische Annalen*, 39:176–211, 1934.

[55] G. Gentzen. Die widerspruchsfreiheit der reinen Zahlentheorie. *Mathematische Annalen*, 112:493–565, 1936.

[56] G. Gentzen. *Collected Works. Edited by M.E. Szabo.* North-Holland, Amsterdam, 1969.

[57] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[58] J.-Y. Girard. Geometry of interaction I: Interpretation of system F. In *Logic Colloquium '88*, Amsterdam, 1989. North-Holland.

[59] J.-Y. Girard. Towards a geometry of interaction. In: Contemporary Math. 92, Amer. Math. Soc., 1989. 69-108.

[60] J.-Y. Girard. Geometry of interaction II: Deadlock-free algorithms. In: Springer LNCS 417, 1990.

[61] J.-Y. Girard. La logique linéaire. *Pour La Science, Édition Francaise de Scientific American*, 150:74–85, April 1990.

[62] J-Y. Girard. Proof-nets: the parallel syntax for proof-theory. Draft, 1994.

[63] J.-Y. Girard and Y. Lafont. Linear logic and lazy computation. In *TAPSOFT '87, Volume 2*, pages 52–66. Springer LNCS 250, 1987.

[64] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types.* Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1989.

[65] J.-Y. Girard, A. Scedrov, and P.J. Scott. Bounded linear logic: A modular approach to polynomial time computability. *Theoretical Computer Science*, 97:1–66, 1992.

[66] G. Gonthier, M. Abadi, and J-J. Levy. Linear logic without boxes. In *Proc. 7th IEEE Symp. on Logic in Computer Science*, 1992.

[67] C.A. Gunter and V. Gehlot. Nets as tensor theories. In G. De Michelis, editor, *Proc. 10-th International Conference on Application and Theory of Petri Nets, Bonn*, pages 174–191, 1989.

[68] J.C. Guzman and P. Hudak. Single-threaded polymorphic lambda calculus. In *Proc. 5-th IEEE Symp. on Logic in Computer Science, Philadelphia*, June 1990.

[69] J. Harland and D. Pym. The uniform proof-theoretic foundation of linear logic programming. In *Symposium on Logic Programming*, pages 304–318. MIT Press, October 1991.

[70] J. Harland and D. Pym. On resolution in fragments of classical linear logic. In *LPAR'92, Conference on Logic Programming and Automated Reasoning*, number 624 in Lecture Notes in Artificial Intelligence, pages 30–41. Springer-Verlag, July 1992.

[71] J. Herbrand. Investigations in proof theory. In J. van Heijenoort, editor, *From Frege to Gödel: A Sourcebook of Mathematical Logic, 1879–1931*, pages 525–581. Harvard University Press, Cambridge, MA, 1967. First published 1930.

[72] D. Hilbert and W. Ackermann. *Grundzuge der theoretischen Logic.* Berlin, 1938. Translation available as *Principles of Mathematical Logic*, New York, 1950.

[73] J.S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110:327–365, 1994. Extended abstract appeared in Proc. 6-th Annual IEEE Symposium on Logic in Computer Science, Amsterdam, July, 1991, pages 32-42. Draft available using anonymous ftp from host ftp.cis.upenn.edu and the file pub/papers/miller/ic92.dvi.Z.

[74] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley Publishing Company, 1979.

[75] W. Howard. The formulas-as-types notion of construction. In *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism*, pages 479–490. Academic Press, 1980.

[76] J. Hudelmaier. *Bounds for Cut Elimination in Intuitionistic Propositional Logic.* PhD thesis, Universität Tübingen, 1989.

[77] S. Jaskowski. On the rules of suppositions in formal logic. *Studia Logica*, 1, 1934.

[78] S.L.P. Jones. *The Implementation of Functional Programming Languages.* Prentice Hall, 1987.

[79] M. Kanovich. The multiplicative fragment of linear logic is NP-complete. Technical Report X-91-13, Institute for Language, Logic, and Information, June 1991.

[80] M. Kanovich. The multiplicative fragment of linear logic is NP-complete. Email Message, 1991.

[81] M. Kanovich. Mail to linear logic mailing list. Email Message, 1992.

[82] M. Kanovich. The complexity of neutrals in linear logic. In *Proc. 10-th Annual IEEE Symposium on Logic in Computer Science, San Diego, California*, page To Appear. IEEE Computer Society Press, Los Alamitos, California, June 1995.

[83] M. Kanovich. The direct simulation of minsky machines in linear logic. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in linear logic.* London Mathematical Society Lecture Note Series, Cambridge University Press, 1995. Available by anonymous ftp from lmd.univ-mrs.fr as /pub/girard/Synsem.ps.Z.

[84] M. Kanovich. Undecidability of non-commutative second order multiplicative linear logic. Email Message to Linear Mailing List, May 1995.

[85] M.I. Kanovich. The complexity of Horn fragments of linear logic. *Annals of Pure and Applied Logic*, 69:195–241, 1994. Extended abstract in: 7-th Annual IEEE Symposium on Logic in Computer Science, Santa Cruz, California, June 1992, pages 200-210.

[86] R.M. Keller. Vector replacement systems: A formalism for modelling asynchronous systems. Technical Report CSL TR 117, Princeton University, 1972.

[87] J. Ketonen and R. Weyhrauch. A decidable fragment of predicate calculus. *Theoretical Computer Science*, 32:297–307, 1984.

[88] J. Ketonen and R. Weyhrauch. A decidable fragment of predicate calculus. *Theoretical Computer Science*, 32, 1984.

[89] S.C. Kleene. *Introduction to Metamathematics*. North-Holland, 1952.

[90] S.C. Kleene. Permutability of inferences in Gentzen's calculi LK and LJ. *Memoirs of the AMS*, 1952.

[91] N. Kobayashi and A. Yonezawa. Acl - a concurrent linear logic programming paradigm. In *Symposium on Logic Programming*, pages 279–294, October 1993.

[92] A. P. Kopylov. Second order linear affine logic is undecidable. In *Proc. 10-th Annual IEEE Symposium on Logic in Computer Science, San Diego, California*. IEEE Computer Society Press, Los Alamitos, California, June 1995.

[93] S.R. Kosaraju. Decidability of reachability in vector addition systems. In *Proc. 14-th ACM Symp. on Theory of Computing*, pages 267–281, 1982.

[94] Y. Lafont. The linear abstract machine. *Theoretical Computer Science*, 59:157–180, 1988.

[95] Y. Lafont. Interaction nets. In *Proc. 17-th ACM Symp. on Principles of Programming Languages, San Francisco*, pages 95–108, January 1990.

[96] Y. Lafont. The undecidability of second order linear logic without exponentials. Journal of Symbolic Logic, to appear, 1995. Available by anonymous ftp from lmd.univ-mrs.fr as /pub/lafont/undecid.ps.Z.

[97] Y. Lafont and A. Scedrov. The undecidability of second order multiplicative linear logic. Manuscript, 1995. Available by anonymous ftp from ftp.cis.upenn.edu as pub/papers/scedrov/mrs.[dvi,ps].Z.

[98] Y. Lafont and T. Streicher. Game semantics for linear logic. In *Proc. 6-th Annual IEEE Symposium on Logic in Computer Science, Amsterdam*, pages 43–50. IEEE Computer Society Press, Los Alamitos, California, July 1991.

[99] J. Lambek. The mathematics of sentence structure. *Amer. Math. Monthly*, 65:154–169, 1958.

[100] J. Lambek. How to program an infinite abacus. *Canadian Math. Bulletin*, 4:295–302, 1961.

[101] P. Lincoln. Linear logic. *ACM SIGACT Notices*, 23(2):29–37, Spring 1992.

[102] P. Lincoln and J. Mitchell. Operational aspects of linear lambda calculus. In *Proc. 7th IEEE Symp. on Logic in Computer Science*. IEEE Computer Society Press, Los Alamitos, California, 1992.

[103] P. Lincoln and A. Scedrov. First order linear logic without modalities is NEXPTIME-hard. *Theoretical Computer Science*, 135:139–154, 1994. Available using anonymous ftp from host ftp.cis.upenn.edu and the file pub/papers/scedrov/mall1.dvi.Z.

[104] P. Lincoln, A. Scedrov, and N. Shankar. Linearizing intuitionistic implication. *Annals of Pure and Applied Logic*, 60:151–177, 1993. Extended abstract appeared in Proc. 6-th Annual IEEE Symposium on Logic in Computer Science, Amsterdam, July, 1991, pages 51-62. Draft available using anonymous ftp from host ftp.cis.upenn.edu and the file pub/papers/scedrov/lss91.[dvi,ps].Z.

[105] P. Lincoln, A. Scedrov, and N. Shankar. Decision problems for second order linear logic. In *Proc. 10-th Annual IEEE Symposium on Logic in Computer Science, San Diego, California*. IEEE Computer Society Press, Los Alamitos, California, June 1995.

[106] P. Lincoln and T. Winkler. Constant-Only Multiplicative Linear Logic is NP-Complete. *Theoretical Computer Science*, 135:155–169, 1994. Available using anonymous ftp from host ftp.csl.sri.com and the file pub/lincoln/comult-npc.dvi.

[107] P. D. Lincoln, J. C. Mitchell, A. Scedrov, and N. Shankar. Decision problems for propositional linear logic. *Annals of Pure and Applied Logic*, 56:239–311, 1992. Extended abstract in: 31st Annual IEEE Symposium on Foundations of Computer Science, Volume 2, St. Louis, Missouri, October, 1990, pages 662-671.

[108] P. D. Lincoln and N. Shankar. Proof search in first-order linear logic and other cut-free sequent calculi. In *Proc. 9-th Annual IEEE Symposium on Logic in Computer Science, Paris*, pages 282–291, July 1994.

[109] P.D. Lincoln. *Computational Aspects of Linear Logic.* PhD thesis, Stanford University, 1992.

[110] R. Lipton. The reachability problem is exponential-space hard. Technical Report 62, Department of Computer Science, Yale University, January 1976.

[111] M.H. Loeb. Embedding first-order predicate logic in fragments of intuitionistic logic. *Journal of Symbolic Logic*, 41:705–718, 1976.

[112] I. Mackie. Lilac - a functional programming language based on linear logic. Master's Thesis, Imperial College, London, 1991.

[113] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Trans. on Prog. Lang. and Systems*, 4(2), Feb. 1982.

[114] N. Martí-Oliet and J. Meseguer. An algebraic axiomatization of linear logic models. Technical Report SRI-CSL-89-11, SRI International, 1989.

[115] N. Martí-Oliet and J. Meseguer. From Petri nets to linear logic. In: Springer LNCS 389, ed. by D.H. Pitt et al., 1989. 313-340.

[116] E. Mayr and A. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in Mathematics*, 46:305–329, 1982.

[117] E. W. Mayr. An algorithm for the general petri net reachability problem. *SIAM Journal on Computing*, 13(3):441–460, 1984.

[118] E.W. Mayr. An algorithm for the general Petri net reachability problem. In *Proc. 13-th ACM Symp. on Theory of Computing, Milwaukee*, pages 238–246, 1981.

[119] R. Milner. The Standard ML core language. *Polymorphism*, 2(2), 1985. 28 pages. An earlier version appeared in Proc. 1984 ACM Symp. on Lisp and Functional Programming.

[120] M. Minsky. Recursive unsolvability of post's problem of 'tag' and other topics in the theory of turing machines. *Annals of Mathematics*, 74:3:437–455, 1961.

[121] G. Mints. Gentzen-type systems and resolution rules. Part I. Propositional logic. In P. Martin-Löf and G. Mints, editors, *COLOG-88*, pages 198–231. *Lecture Notes in Computer Science vol. 417, Springer*, 1990.

[122] G. Mints. Some Information on Linear Logic. Draft, 1991.

[123] G. Mints. Natural deduction in the intuitionistic linear logic. Draft, 1995.

[124] H. Ono. Structural rules and a logical hierarchy. In: Mathematical Logic, ed by P. Petkov.

[125] H. Ono. Phase structures and quantales — a semantical study of logics without structural rules. Manuscript, 1991.

[126] M. Pentus. Lambek grammars are context free. In *Proc. 8-th IEEE Symp. on Logic in Computer Science, Montreal*, June 1993.

[127] C. Petri. Kommunikation mit automaten. Technical Report IMM-2-1962, Institut fur Instrumentelle Mathematick (Bonn), 1962.

[128] R. Pliushkevichus. On a version of the constructive predicate calculus without structural rules. *Soviet Math. Doklady*, 6:416–419, 1965.

[129] E.L. Post. Recursive unsolvability of a problem of thue. *Journal of Symbolic Logic*, 12:1–11, 1947.

[130] V. Pratt. Event spaces and their linear logic. In *Second International Conference on Algebraic Methodology and Software Technology*, 1992.

[131] D. Prawitz. *Natural Deduction.* Almquist and Wiksell, Stockholm, 1965.

[132] D. Roorda. Resource logics: proof-theoretical investigations. Dissertation, University of Amsterdam, September 1991.

[133] A. Scedrov. A brief guide to linear logic. *Bulletin of the European Assoc. for Theoretical Computer Science*, 41:154–165, June 1990.

[134] R.A.G. Seely. Linear logic, *-autonomous categories, and cofree algebras. In: Contemporary Math. 92, Amer. Math. Soc., 1989.

[135] R.A.G. Seely. Linear logic, *-autonomous categories, and cofree coalgebras. In *Categories in Computer Science and Logic*, pages 371–382. Contemporary Math., vol. 92, American Math. Soc., Providence, RI, 1989.

[136] N. Shankar. Proof search in the intuitionistic sequent calculus. In D. Kapur, editor, *Automated Deduction: CADE-11*, volume 607 of *lncs*, pages 522–536, Berlin, 1992. Springer-Verlag.

[137] E.Y. Shapiro. Alternation and the computational complexity of logic programs. *Journal of Logic Programming*, 1:19–33, 1984.

[138] R. Statman. Intuitionistic propositional logic is polynomial-space complete. *Theoretical Computer Science*, 9:67–72, 1979.

[139] L. Stockmeyer. Classifying the computational complexity of problems. *Journal of Symbolic Logic*, 52:1–43, 1987.

[140] T. Tammet. Proof search strategies in linear logic. Programming Methodology Group Report 70, Chalmers University, 1993.

[141] A. Troelstra. *Lectures on Linear Logic.* Center for the Study of Language and Information, 1991.

[142] A.S. Troelstra. *Lectures on Linear Logic.* CSLI Lecture Notes No. 29. Center for the Study of Language and Information, Stanford University, 1992.

[143] A.S. Troelstra. Natural deduction for intuitionistic linear logic. Technical Report ML-93-09, Institute for Logic, Language, and Computation, University of Amsterdam, May 1993.

[144] David N. Turner, Phil Wadler, and Christian Mossin. Once upon a type. In *Proceedings of the Conference on Functional Programming Languages and Computer Architecture*, La Jolla, CA, June 1995.

[145] A.M. Ungar. *Normalization, Cut-Elimination and the Theory of Proofs.* CSLI Press (Chicago), 1992. CSLI Lecture Notes Number 28.

[146] A. Urquhart. The undecidability of entailment and relevant implication. *Journal of Symbolic Logic*, 49:1059–1073, 1984.

[147] A. Urquhart. The complexity of decision procedures in relevance logic. Technical Report 217/89, Department of Computer Science, University of Toronto, 1989.

[148] N. Vorobjev. New derivability algorithm in the constructive propositional calculus. (In Russian). In *Proceedings of the Steklov institute of Mathematics (Trudy), v.52*, pages 193–225, 1958.

[149] 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.

[150] P. Wadler. Is there a use for linear logic? In *Proceedings of the Symposium on Partial Evaluations and Semantics-Based Program Manipulation*, pages 255–273, New Haven, Connecticut, June 1991. Also in SIGPLAN Notices; vol.26, no.9; September 1991.

[151] P. Wadler. There's no substitute for linear logic. Manuscript. Presented at Workshop on Programming Language Semantics, Oxford, April 1992, December 1992.

[152] P. Wadler. A syntax for linear logic. In S. Brookes, M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Proceedings of the Ninth Conference on Mathematical Foundations of Programming Semantics*, pages 513–529, New Orleans, Louisiana, April 1993. Springer-Verlag LNCS 647.

[153] P. Wadler. A taste of linear logic. In A. M. Borzyszkowski and S. Sokolowski, editors, *Mathematical Foundations of Computer Science*, pages 185–210, Gdansk, Poland, 1993. Springer-Verlag LNCS 781.

[154] M. Wajsberg. Untersuchungen über den Aussagenkalkül von A. Heyting. *Wiadomości Matematyczne*, 46:45–101, 1938.

[155] D. Wakeling and C. Runciman. Linearity and laziness. In *5th Conf. on Functional Programming Languages and Computer Architecture, Lecture Notes in Computer Science 523*, New York, 1991. Springer-Verlag.

[156] L. Wallen. *Automated Proof search in non-classical logics.* MIT Press, 1990.

[157] D.N. Yetter. Quantales and (noncommutative) linear logic. *Journal of Symbolic Logic*, 55:41–64, 1990.

C Index