Light Logic and Polynomial Time Computation

A Thesis

Submitted to

Faculty of Letters

Keio University

In Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy

by

Kazushige Terui

Acknowledgments

First of all, the author would like to express his gratitude to Dr. Max Kanovitch. During his visit at Keio University in 1998, he kindly spent much time for discussions, from which the author benefited considerably. Indeed, it was through the discussion on Elementary Linear Logic with him that the author first gained the core insight into the polytime strong normalization theorem, one of the main results of this thesis. The discussion concerning phase semantics was also helpful; it is reflected in the presentation of Chapter 7.

Second, the author would like to express his gratitude to the members of Mita Logic Seminar, especially Dr. Misao Nagayama and Mr. Ken Shiotani, for helpful suggestions and stimulating discussions.

Professor Harry Mairson gave valuable comments on my presentation at the 16th Annual Symposium on Logic in Computer Science; his comments are reflected in Chapter 4. Professor Takashi Iida and Professor Andre Scedrov made a number of helpful suggestions on the manuscript of this thesis. Dr. Masaru Shirahata kindly replied to the author's questions on contraction-free set theory. The author would like to thank them sincerely.

Last but not least, the author would like to express his special thanks to his invaluable advisor, Professor Mitsuhiro Okada, for helpful suggestions and constant encouragements; without his help, this thesis would never be completed.

Contents

Chapter 1

Introduction

This thesis is intended to be a thorough investigation of the family of formal logical systems, called *Light Logic*, which has been introduced recently and expected to shed a new light on the nature of the *feasible computation* (i.e., computation executable in the real world) from a logical perspective. Although our investigation below will be mostly technical, we shall first explain the main issues of this thesis in an informal way.

In Section 1.1, we shall give the background, informally explaining the key concepts such as proofs, programs, feasibility and the polynomial time computation. Then, after having mentioned Linear Logic, in which our basic methodology originates, we shall arrive at the main theme of this thesis, Light Logic. In Section 1.2, we shall summarize the main results of this thesis. In Section 1.3, we shall discuss related works and compare them with our Light Logic approach. In Section 1.4, we shall outline the contents of this thesis.

1.1 Background

Proofs and programs. Although proofs and programs are conceptually distinct notions which are concerned with two different aspects of human/machine intelligence, that is reasoning and computation, it is not to be denied that these two are closely related in certain concrete situations. For example, consider the following mathematical proposition:

(*) There are infinitely many prime numbers; namely, for any natural number n, there exists a prime number m which is larger than n.

The well-known Euclid's proof to (*) [EucBC] goes, roughly, as follows:

> *Given a natural number n, let $m_0 = n! + 1$. If m_0 is prime, then m_0 satisfies the condition. Otherwise, let m_1 be the smallest divisor of m_0. Then this m_1 satisfies the condition.* Indeed m_1 is prime, since any number which is the smallest divisor of another number is always prime. Moreover, m_1 is larger than n, since no number $\leq n$ can divide $m_0 = n! + 1$.

This is a typical example of *constructive proofs*, which, for an existential statement $\exists x A(x)$, provide an effective means to find an object m as well as a proof of $A(m)$ (see [TvD88]). Constructive proofs have algorithmic content. Indeed, the italicized part in the above proof may well be construed as describing an *algorithm* for obtaining a desired prime number. The rest of the proof then verifies

Logical Notions	Computational Notions
Proofs	Programs
Formulas	Types (Specifications)
Cut-Elimination (Normalization)	Computation (Execution)

Table 1.1: Proofs-as-Programs Correspondence

that the number obtained is really a desired one. Thus the above proof consists of two parts, namely the algorithmic part and the verification part. The algorithmic part can be formally described as a *program* in a usual programming language.

On the other hand, consider the following program specification:

(**) Given a natural number n as an input, return a prime number m which is larger than n.

The primary task of a programmer who is given this specification is to write a program for it, typically a formal description of the algorithmic part of the above proof, but that is not all what he/she has to do. After having written a program, the programmer may be asked to *certify* its correctness, especially when reliability of the program is crucial. A certification is, ideally, given by means of a mathematical *proof*, and it is likely that such a proof coincides with the verification part of the above proof. In this way, we can observe a tight connection between proofs and programs, at least when constructive proofs and certified programs are concerned.

The purest form of this proofs-programs connection can be found in a formal setting, which is now widely known as the *proofs-as-programs correspondence*[1] [CF58, How80]. According to this correspondence, proofs are not just related to, but even *identified* with programs. To be more specific, proofs of certain logical systems, e.g., Intuitionistic Logic, are interpretable as programs of certain (models of) programming languages, e.g., typed λ-calculus; the formulas are then interpreted as types, i.e., specifications of programs, and the cut-elimination procedure in a sequent calculus system [Gen35] (or equivalently the normalization procedure in a natural deduction system [Pra65]) is considered as computation, i.e., execution of programs (see Table 1.1).

The proofs-as-programs correspondence is theoretically interesting in formally relating two conceptually different aspects of intelligence, i.e., reasoning and computation. It is practically advantageous in providing a unified framework for programming, deduction and verification.

The proofs-as-programs correspondence is usually discussed within a framework of very high computational complexity. For example, the simplest form of the correspondence may be found between propositional Intuitionistic Logic and simply typed λ-calculus, but normalization (i.e., execution of a program) in the latter is already hyper-exponential (i.e., requires of towers of exponentials) [Sta79]. However, hyper-exponential time computability is merely a theoretical notion which has little to do

[1]It is also known as the *Curry-Howard isomorphism* (or the *formulas-as-types interpretation*). We prefer the word "proofs-as-programs" for the following reasons. First, the word "Curry-Howard isomorphism" sometimes refers to a specific relation, rather than a general paradigm, between propositional Intuitionistic Logic and simply typed λ-calculus, while we would like to apply the word to other logical/computational systems as well. Second, the Curry-Howard isomorphism is usually discussed in the context of Church-style typed calculi, while in this thesis we deal with type-free calculi with Curry-style type assignment systems. Third, in our framework, quantifiers are interpreted implicitly, i.e., the logical inference rules for quantifiers do not appear explicitly in the computational languages. Namely, the interpretation of proofs is not isomorphic, but just homomorphic. Therefore, it is at least debatable whether our framework really falls under the Curry-Howard isomorphism in its most strict sense. Instead of discussing this point further, we adopt a more neutral terminology "proofs-as-programs" which does not seem to have such a specific meaning.

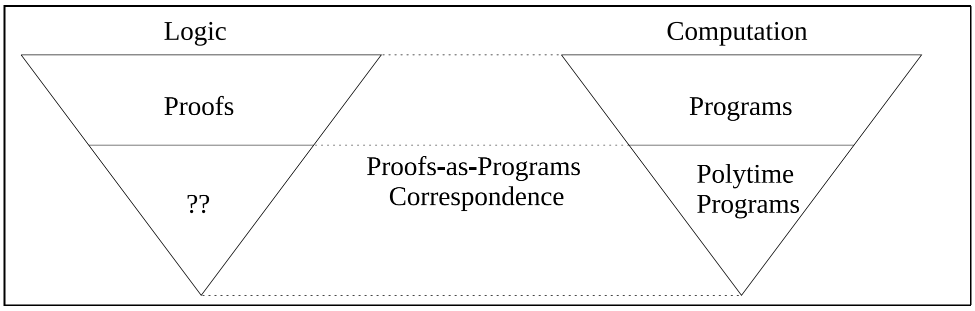


Figure 1.1: What corresponds to polytime programs?

with the real world computation, since the time required for computation may be far larger than the lifespan of the universe (see, e.g., [GJ78]). Hence a natural question is whether this paradigm can be accommodated to a framework of lower computational complexity as well. In this thesis, we shall discuss the proofs-as-programs correspondence in the context of the *feasible computation*, which is explained below.

Feasible computation and polynomial time. In the real life, programmers write computer programs for solving daily computational tasks, such as calculating wages of employees, sorting a list of customers, finding the most efficient way of transportation, and so on. Such programs are required not only to be correct, but also to be executable within a reasonable amount of time and space. In short, such programs are required to be *feasibly* executable. There is wide agreement among researchers in the field of computer science that feasible algorithms are identified with polynomial time algorithms, i.e., those which are computable by Turing machines within a polynomial number of steps (see, e.g., [GJ78] for an argument for this). Thus the feasible computation is identified with the polynomial time computation, or shortly the *polytime* computation, and feasibly computable functions are identified with polytime functions. Since the notion of feasibility is so crucial in the real world computation, it has been one of the central issues in computer science to understand the nature of the polytime computation.

In view of the proofs-as-programs correspondence which allows us to analyze various aspects of computation from a logical perspective, it is natural to ask what the polytime computation amounts to in terms of logic. Is it possible to find a purely logical notion which does not presuppose polytime but does capture it according to the proofs-as-programs paradigm? To put it in other words, is it possible to find a well-delimited subset of logical proofs which precisely correspond to polytime programs (see Figure 1.1)? There are various ways to approach this problem. For instance, one could seek such a logical notion in the complexity of formulas (i.e., degree of quantifier alternation) or in the structure of the induction inference rule in an arithmetical system. Our approach, by contrast, focuses on the structure of logical inference rules. Among those, we are particularly interested in the *structural inference rules*, which arise in Gentzen's sequent calculus most naturally. (Other approaches are mentioned in Section 1.3.)

Control of Contraction: Linear Logic. Apart from the logical inference rules and the cut rule, Gentzen's sequent calculus[2] [Gen35] contains structural inference rules, Weakening and Contraction[3]:

[2]Although Gentzen also considered sequent calculus for Classical Logic, we shall exclusively deal with sequent calculus for Intuitionistic Logic below in order to simplify the argument.

[3]In what follows, the Exchange inference rule of [Gen35]

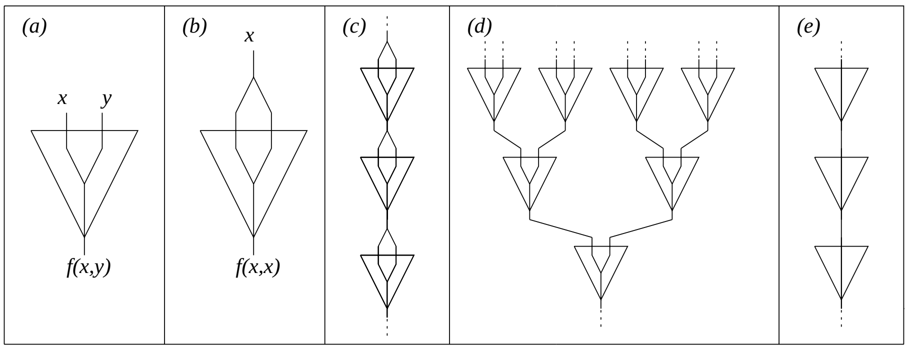


Figure 1.2: Iteration of Functions

$$\frac{\Gamma \vdash C}{A, \Gamma \vdash C}\ (Weak) \qquad \frac{A, A, \Gamma \vdash C}{A, \Gamma \vdash C}\ (Contr).$$

Weakening means that a redundant assumption may be added harmlessly, while Contraction means that two assumptions of a formula A is identified with a single assumption of A. Of these two, we are particularly interested in the latter, i.e., Contraction. Although it seems quite harmless from a viewpoint of logical reasoning, where it is irrelevant how many times one uses an assumption, it actually causes a disastrous effect, namely an *exponential explosion*, when it comes to the complexity of computation. For illustration, let us consider a proof of conclusion $A, A \vdash A$. According to the proofs-as-programs paradigm, such a proof corresponds to a function $f(x, y)$ with two inputs and one output all of which are of type A. The function f is schematically drawn in Figure 1.2(a). By Contraction, we can obtain a proof of conclusion $A \vdash A$ which corresponds to function $f(x, x)$ (Figure 1.2(b)). If this function is iterated (Figure 1.2(c)), it immediately gives rise to an exponentially growing computational tree (Figure 1.2(d)). Since iteration is such a basic mechanism that it cannot be dispensed with in most realistic computational systems, a reasonable way to avoid this exponential explosion is to restrict the use of Contraction in one way or another.

At this point, it is suggestive to pay a visit to *Linear Logic* [Gir87, Gir95], which embodies an elegant means to control the use of structural inference rules on the object level. Linear Logic is often said to be a resource-sensitive logic, and this is precisely because it takes special care of Contraction and Weakening. In Linear Logic, a formula is not allowed to be contracted or weakened unconditionally. Rather, a formula must be *authorized* by means of a modal operator ! in advance of being contracted or weakened. Formally, (the intuitionistic version of) Linear Logic is obtained from Intuitionistic Logic in the following three steps: (i) remove Contraction and Weakening, (ii) enrich Intuitionistic Logic with an S4-modal operator ! (called an *exponential*), (iii) re-introduce Contraction and Weakening, but this time only for !-prefixed formulas[4]:

$$\frac{\Gamma \vdash C}{!A, \Gamma \vdash C}\ Weak \qquad \frac{!A, !A, \Gamma \vdash C}{!A, \Gamma \vdash C}\ Contr.$$

In this way the exponential modal operator controls the use of Contraction and Weakening. These modifications result in a new logical system which is constructive (in the sense that proofs have

$$\frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C}\ (Exch)$$

will be assumed implicitly.

[4]A more detailed exposition is given in Chapter 2.

algorithmic content) and which inherently possesses control over resources. It helps us analyze Intuitionistic and Classical Logics and give a deep insight into the nature of cut-elimination (see in particular [Gir91, DJS95, DJS97]). Practical applications of Linear Logic are abundant, for which we refer to [Sce93, GLR95].

Linear Logic is not a restriction, but a refinement of Intuitionistic and Classical Logics. As such a logic, it has roughly the same expressive power as Intuitionistic and Classical Logics, thus cut-elimination is still hyper-exponential. For instance, from a proof of conclusion $A, A \vdash A$ which represents $f(x, y)$, we may obtain a proof of conclusion $!A \vdash !A$

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\\ A, A \vdash A\end{array}}{\dfrac{}{!A, !A \vdash A}}\ !l}{!A, !A \vdash !A}\ !r}{!A \vdash !A}\ Contr$$

which represents $f(x, x)$ and by iteration causes an exponential explosion as before. Therefore, Linear Logic itself does not capture polytime. Nevertheless, the idea of controlling Contraction by means of a modal operator is quite attractive, and indeed it is this idea which, after a pioneering attempt of Bounded Linear Logic [GSS92], leads Girard to an intrinsically polytime system: Light Linear Logic.

Taming of Contraction: Light Logic. In [Gir98], Girard introduced *Light Linear Logic* (**LLL**) which is intended to capture the polytime computation in the proofs-as-programs paradigm. It was proved that every polynomial time function is representable by a proof of (the second order) **LLL**, and conversely that every **LLL** proof is normalizable via cut-elimination in polynomial time. Thus the representable functions in **LLL** are precisely polytime. Later on, in [Asp98], Asperti introduced a simplified system, called *Light Affine Logic*, by adding the full (unrestricted) Weakening rule to **LLL**. The intuitionistic versions of these systems, *Intuitionistic Light Linear Logic* (**ILLL**) and *Intuitionistic Light Affine Logic* (**ILAL**), were also introduced by Girard and Asperti, respectively. Since these systems, **LLL**, **LAL**, **ILLL** and **ILAL**, are just variations on the same theme, namely taming of Contraction, we shall collectively call them systems of *Light Logic.*

While Linear Logic is concerned with control of Contraction, Light Logic is concerned with taming of Contraction. The basic idea is to replace the exponential modality ! of Linear Logic, which controls Contraction, with two more tamed ones, called *light exponentials.* Now we have two modal operators ! and § with the following inference rules:

$$\frac{B \vdash A}{!B \vdash !A}\ (!) \qquad\qquad \frac{B_1, \ldots, B_m, C_1, \ldots, C_n \vdash A}{!B_1, \ldots, !B_m, \S C_1, \ldots, \S C_n \vdash \S A}\ (\S)\quad m, n \geq 0 .$$

Now assume that a proof of conclusion $A, A \vdash A$ is given and it represents a function $f(x, y)$ as before. In contrast to the case of Linear Logic, one can no more produce a proof of conclusion $!A \vdash !A$ from that, since rule (!) only applies to those sequents which have at most one assumption formula. One could produce a proof of $\S A, \S A \vdash \S A$, but it is useless because Contraction does not apply to §-prefixed formulas. The only possibility left to us is to produce a proof of $!A \vdash \S A$ as follows:

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ A, A \vdash A\end{array}}{!A, !A \vdash \S A}\ \S}{!A \vdash \S A}\ Contr .$$

Although this proof surely represents $f(x,x)$, it cannot be iterated anymore, since the input type $!A$ is different from the output type $\S A$. Actually rule (!) of Light Logic allows a form of iteration, but it no more yields an exponential computational tree like Figure 1.2 (d) but it just yields a linear one like Figure 1.2 (e); iteration is not exponential, but linear in Light Logic. In general, what can be obtained by several applications of linear iteration is at best a polytime function. Thus exponential time functions are successfully ruled out[5].

Since its inception, Light Logic has been investigated by various authors. Kanovitch, Okada and Scedrov [KOSar] considered phase semantics for **LLL** and gave a semantic proof to the cut-elimination theorem for **LLL**. Baillot [Bai00] considered coherent semantics for **LLL**, while Murawski and Ong [MO00b] proposed a game semantics for a multiplicative-exponential fragment of **LAL** and proved a full completeness theorem. Roversi [Rov99] fixed a gap which was found in the proof of the polytime representability theorem in [Gir98], and described a very precise encoding of polytime Turing machines in **ILAL**. He also attempted to design a functional programming language based on **ILAL** with ML-like polymorphism in [Rov00]. Asperti and Roversi [AR00] presented a proofnet syntax for **ILAL** and investigated some syntactic issues. Murawski and Ong [MO00a] investigated a relationship between Light Logic and Bellantoni-Cook's safe recursion formalism (see below). As for extensions of Light Logic, there is a variant of **LLL**, called *Elementary Linear Logic*, which captures the elementary recursive complexity; it was introduced in the appendix of [Gir98] and later reformulated in [DJ99]. On the other hand, we gave a characterization of the polynomial space functions based on the Light Logic approach in [Ter00]. There are also a considerable number of related works, which are discussed in Section 1.3.

1.2 Main Results of This Thesis

In spite of those past studies mentioned above, we have not yet arrived at a comprehensive understanding of Light Logic. First, the computational aspect (i.e., cut-elimination) of Light Logic is not fully understood; it has been known that proofs are polytime normalizable, but has not been known if polytime normalizability here is in the strong sense or not (see below). Second, the reasoning aspect of Light Logic is not completely explored; it has not been fully examined what kind of reasoning is possible and what kind of formal theories can be developed in Light Logic. Third, the semantic aspect of Light Logic is to be studied further. Moreover, certain basic properties of a logical system, such as decidability and finite model property, are not known for Light Logic. In this thesis, we investigate Light Logic with the aim of clarifying these three aspects. In doing so, we mainly deal with the simplest system of Light Logic, **ILAL**. Our main results are stated as follows.

1. In order to clarify the computational aspect of Light Logic, we introduce a term calculus λLA, called *Light Affine Lambda Calculus*, which embodies the essence of the proof system of **ILAL**. Proofs of **ILAL** are structurally representable as terms of λLA. Then we prove the main theorem:

Polytime Strong Normalization Theorem. Terms of λLA are normalizable in polynomial time regardless of which reduction strategy we take.

2. In order to examine the reasoning aspect of Light Logic, we develop Cantor's naive set theory based on **ILAL**, called *Light Set Theory*. Our main result is:

[5]This is merely a rough sketch of the idea, however. The actual situation is more complicated, and we need another important mechanism, which we call *stratification of proofs*. We shall explain the latter in Subsection 2.4.2 of Chapter 2.

Provable Totality of Polytime Functions. A function is polytime computable if and only if it is provably total in Light Set Theory.

It confirms that a sensible mathematical theory which has a truly polytime character can be developed on the basis of Light Logic.

3. As a semantic means to investigate various properties of Light Logic, we introduce a sound and complete semantics for **ILAL**, called *light affine phase semantics* (as well as a slightly generalized version of it). Then we prove:

The Finite Model Property for ILAL. A formula is provable in **ILAL** if and only if it is satisfied in all *finite* (generalized) light affine phase models.

It follows that **ILAL** is decidable. A more detailed account of these results is given in Section 1.4.

Light Logic captures the polytime computation through the proofs-as-programs paradigm. Hence to study Light Logic is at the same time to study the nature of the polytime computation. We hope that our study will lead to a better understanding of the feasible computation from a logical perspective.

1.3 Other Approaches to Characterization of Polytime

Light Logic provides a logical characterization of the polytime functions in the paradigm of proofs-as-programs and cut-elimination-as-computation. The characterization is *machine-independent* (it does not mention Turing machines or other machine models) and *resource-free* (the syntax does not contain explicit polynomial bounds). Since similar characterizations are abundant in the literature, we need to compare Light Logic with them in order to clarify the characteristic of the Light Logic approach. For convenience, we classify other approaches into the following categories: (1) recursion theory, (2) functional programming, (3) proof theory, and (4) finite model theory.

(1) Recursion Theory
The seminal work in this area is Cobham's recursion-theoretic characterization of the polytime functions by *bounded recursion on notation* [Cob65]. This is the first machine-independent characterization of a complexity class and has influenced later work so much. There is, however, an unpleasant point in his result that in applying recursion one needs an explicit bounding function in addition to the usual base and step functions, and one also needs an *ad hoc* initial function $2^{|x|\cdot|y|}$, called the *smash function*, sorely to provide a large enough bounding function. In some appropriate sense, Cobham's characterization does not presuppose polynomial time but *does* presuppose polynomial growth rate, hence his characterization is not resource-free.

The *ad hoc* character of [Cob65] has been later remedied in the *safe recursion* (or *ramified recurrence*) approach. Bellantoni and Cook [BC92] provide a resource-free characterization of polytime which does away with Cobham's bounding condition. It is based on a conceptual distinction between two ways that data objects are used in computing. For instance, a 0-1 word can be used *locally* as a collection of 0-1 data each of which is accessed bitwise, or it may be used *globally* as a template for function iteration. According to this distinction, [BC92] classifies arguments of functions into two, *safe* arguments (for local use of arguments) and *normal* arguments (for global use of arguments), and imposes a constraint that recursion can be applied only on safe arguments. The resulting system precisely captures the polytime functions in a machine-independent and resource-free way. Leivant [Lei93, LM94] provides a similar characterization based on a more general concept of data ramification and ramified recurrence.

These characterizations are purely recursion-theoretic while Light Logic is proof-theoretic. Nevertheless, a partial relationship between them is established by Murawski and Ong [MO00a] based on the idea of identifying **bint**, which is a type for binary integers in **ILAL**, with normal inputs and $\S^k$**bint** with safe inputs. It is then proved that safe recursion with *non-contractible* safe variables is interpretable in the second order **ILAL**.

(2) Functional Programming
The idea of safe recursion is also available in the functional systems, as first exhibited by [LM93]. Later on, a typical polytime functional system is described by Hofmann [Hof97], where he considers simply typed λ-calculus with data objects, distinguishes safe and normal arguments by means of S4-modal types and then incorporates safe recursion of [BC92] as a recursor of base type. The system is extended with higher type recursors in [BNS99, Hof98, Hofar], by imposing some linearity constraint on the use of variables of higher type. See [Hof] for a good survey.

Closely related, but conceptually different characterizations of polytime are given based on *applicative control.* The idea is to restrict the form of programs syntactically (rather than type-theoretically) so as to guarantee their computational complexity. Jones [Jon97] characterizes the polytime functions by recursion with *read-only variables.* Leivant [Lei99] extends this applicative control approach to a calculus with higher type recursion, by appealing to some linearity constraint on variables of higher type as in [BNS99, Hof98, Hofar].

These functional systems based on safe recursion or applicative control bear some similarity with Light Logic, as suggested by the above mentioned result of [MO00a] and by the fact that linearity constraint is crucial in both approaches. Nevertheless, these functional systems are formally distinguished from Light Logic in several ways. First, their underlying frameworks are different; all functional systems mentioned above are based on λ-calculus with data objects and a recursor (like Gödel's system T [Göd58]), while the proof systems of Light Logic are polymorphic calculi without built-in data objects and a recursor (like Girard's system F [Gir72]). Second, in the above functional systems, the input/output dependencies are polytime only for *base types*, and not for *higher types.* Indeed, normalization may be of hyper-exponential complexity at higher types. This shows that the global structures of these systems are not necessarily of polytime character. On the other hand, normalization (cut-elimination) is polytime at *any type* in Light Logic; in this respect one could say that Light Logic is more adequate for characterizing polytime in the paradigm of normalization-as-computation, since normalization (cut-elimination) in Light Logic is globally polytime. Third, there is a significant difference even at the base types. The above functional systems are at best *weakly* polytime; they admit a base type program whose normalization requires of exponential time when one takes a bad reduction strategy. On the other hand, all programs expressible in Light Logic are *strongly* polytime.

(3) Proof Theory
As for proof-theoretic investigations of the polytime complexity, the first work to be mentioned is Cook [Coo75], where an equational theory PV is introduced and the notion of feasibly constructive proof is analyzed based on it. Although PV is interesting in its own right, it involves an iteration schema similar to bounded recursion on notation of [Cob65] as well as certain *ad hoc* initial functions. Thus it is not resource-free. In [CU93], system PV is extended to higher type functionals PV^ω and

to logical systems IPV with quantifiers.

Another seminal work in this area is Buss's *Bounded Arithmetic* [Bus86], where a deep connection between bounded fragments of arithmetic and polynomial time hierarchy is revealed. Among systems of Bounded Arithmetic, S^1_2 captures the polytime functions. The relationship with Cook's PV and its extensions is examined in [Bus86, Bus93, CU93]. Bounded Arithmetic is now one of the central subjects in proof theory and being studied quite extensively (see [HP93] and [Kra95]). However, the insight it offers to polytime is quite different from that Light Logic offers. First, syntax of S^1_2 contains Cobham's smash function as a primitive, and induction formulas have to be bounded; i.e., the characterization is not resource-free. Second, Buss [Bus86] introduces a general method of *witnessing*, which allows us to extract a polytime algorithm (in terms of Turing machines) from a given proof of totality of a function in S^1_2. But the witnessing method applies only to *cut-free* proofs, while cut-elimination itself is of hyper-exponential complexity. System S^1_2 proof-theoretically captures the polytime computation, but not in the paradigm of proofs-as-programs and cut-elimination-as-computation.

Resource free characterizations are given in [Lei94, Lei95, HBar] by developing the idea of safe recursion/ramified recurrence in proof-theoretic settings. But in those works, again, the inner structure of proofs forced by the cut-elimination procedure is not of polytime character.

Light Logic has a precursor, namely *Bounded Linear Logic* [GSS92], which provides a proof-theoretic characterization of polytime in the proofs-as-programs paradigm. The basic idea is to replace the exponential modality $!A$ of Linear Logic, which means an unlimited supply of A, with a bounded one $!_x A$, which means a supply of A up to x times. The resulting system has a polytime cut-elimination procedure and is enough expressive to encode all polytime functions. There is, however, a drawback that polynomial resource parameters explicitly appear in the syntax; Bounded Linear Logic surely captures polytime, but in doing so it crucially refers to polynomials. It should be noted, however, that Bounded Linear Logic has been recently recast by a notable work [HS00], where a realizability model for Bounded Linear Logic is introduced and the polytime upperbound for the representable functions is shown based on it. It would be exciting if one could import the realizability method developed there to Light Logic. Another work to be noted is Lafont's new polytime system, called *Soft Linear Logic* [Laf01], which can be seen as an elegant simplification of Bounded Linear Logic. We shall mention the latter in Section 4.5 of Chapter 4.

(4) Finite Model Theory
Finally, we should mention that there are many nontrivial characterizations of polytime in the finite model theory (and database queries) approach, such as [Saz80, Var82, Gur83, Pap85, GS86, Imm86, Imm87, Lei90]. Typically in [Imm86], polytime predicates are characterized by formulas of first order Classical Logic with *inflationary fixpoints* in a model-theoretic way (see [EF99] for the general background). Those model-theoretic characterizations, however, have very little to do with the proof-theoretic characterization of Light Logic; what plays an essential role there is *models*. On the other hand, we intend to capture polytime in terms of *proofs*.

1.4 Outline of This Thesis

We have already stated our main results in Section 1.2. Here we explain the main issues of this thesis in more detail and from a broader perspective.

Syntax of Light Logic (Chapter 2). We are concerned with the above problems for Light Logic in general, but it would be tedious and fruitless to consider all systems of Light Logic at once. Rather, we shall pick out **ILAL** as a sample system and address the above problems mainly for **ILAL**. The reason why we select **ILAL** is that it is arguably simpler than the other systems. The simplicity of **ILAL** will allow us to concentrate on the critical issues, leaving unnecessary complications aside.

In Chapter 2, we shall describe syntax of **ILLL** and then syntax of **ILAL**. It will be explained how the latter simplifies the former, and why the classical counterpart of **ILAL**, i.e. **LAL**, is not appropriate. **LLL** is as complex as **ILLL**. In addition, we shall mention undecidability of **ILLL**, which is in contrast to decidability of **ILAL**. These results witness the relative simplicity of reasoning in **ILAL** compared with others, thus justify our choice of **ILAL**.

Polytime strong normalizability of Light Logic proofs (Chapter 3 and Chapter 4)[6]. The intrinsically polytime nature of Light Logic is best witnessed by the polytime normalization theorem, which states that every proof can be normalized into a cut-free one in polynomial time. Although the theorem has been shown for **LLL** by Girard and for **ILAL** by Asperti, there still remains an important problem. What is actually shown in [Gir98, Asp98] is the polytime *weak* normalizability, namely, that there is a *specific* reduction strategy which normalizes a given **LLL** proof in polytime. It has been left unsettled whether the polytime *strong* normalizability holds for these systems of Light Logic, namely, whether *any* reduction strategy normalizes a given proof in polytime. In Chapters 3 and 4, we shall address this problem and give it an affirmative answer.

Having such a property will be theoretically important in that it gives further credence to Light Logic as an intrinsically polytime system. It will be practically important, too. Through the proofs-as-programs correspondence, each proof of Light Logic may be considered as a *feasible program*, which is executable in polytime, and whose bounding polynomial is specified by its type (formula). In this context, the property will assure that the polytime executability of such a program is not affected by the choice of an evaluation strategy.

In Chapter 3, we shall introduce a new term calculus, called *Light Affine λ-Calculus* (λLA), which embodies the essential mechanisms of Light Logic in an untyped setting. It amounts to a simple modification of λ-calculus with modal and let operators. The calculus λLA is untyped, but remarkably, all its *well-formed* terms are polytime normalizable; here well-formedness is a type-free syntactic condition defined on terms. The second order version of **ILAL**, which is denoted as **ILAL2**, is then re-introduced as a Curry-style type assignment system for λLA in Chapter 4, and the subject reduction theorem is proved. This basically means that proofs of **ILAL2** can be embedded in λLA, and the cut-elimination procedure of **ILAL2** is compatible with the reduction rules of λLA under that embedding. In this way, the proofs-as-programs interpretation is demonstrated for **ILAL2**.

Several term calculi for **ILAL2** have been introduced before (for example, [Asp98, Rov00, Rov99, AR00]). However, those calculi either have a complicated notion of reduction defined by more than 20 rewriting rules [Asp98, Rov00], or involve notational ambiguity [Rov99, AR00].[7] It is often the case that such complication and ambiguity make the computational intuition behind a calculus less clear. On the other hand, λLA has very simple operational behavior defined by just 5 reduction rules each of which has a clear meaning, and yet it is free from ambiguity.

Another difference from those existing term calculi is that λLA is introduced as an untyped calculus from the outset. There are a number of reasons in doing so:

[6]This part is written based on our recent work [Ter01].

[7]See the remark in Section 9.1 of [AR00]. Instead, the latter paper presents a proofnet syntax for **ILAL**, based on which several computational properties are investigated.

1. First of all, to design a truly polytime (rather than just polystep) polymorphic calculus, one must give up a Church-style term syntax with embedded types; a universal quantifier may bind an arbitrary number of type variable occurrences, and thus iterated type instantiations (Λ reductions) may easily cause exponential growth in the size of types.[8]

2. An untyped polytime calculus deserves investigation in its own right. (This program was advocated in the appendix of [Gir98], but has not been developed so far.)

3. The notion of well-formedness, rather than typability, neatly captures the syntactic conditions for being polytime normalizable.

4. Last but not least, typability in **ILAL2** is presumably intractable,[9] while well-formedness is checked very easily (in quadratic time).

We believe that our approach is more transparent than the above mentioned approaches in modeling the computational mechanism of Light Logic[10].

In this setting, we prove the polytime strong normalization theorem: every reduction strategy (given as a function oracle) induces a normalization procedure which terminates in time polynomial in the size of a given term (of fixed depth). It follows that every term typable in **ILAL2**, which can be viewed as a structural representation of an **ILAL2** proof (with formulas erased), is polytime strongly normalizable. We shall argue that essentially the same holds for **LLL**.

Development of naive set theory in Light Logic (Chapter 5 and Chapter 6). Being a logical system, Light Logic may be seen as a formal system of reasoning, and therefore it can be employed as a basis for concrete mathematical theories by enriching it with non-logical axioms and/or non-logical inference rules. There are various possibilities as to what mathematical theory we develop. For example, we could enrich Light Logic with the axioms of Peano Arithmetic to obtain a feasible version of first order arithmetic, or we could think of its second order extension, i.e., a feasible version of analysis. Among those, the most interesting, most well-principled, and most radical option is perhaps to enrich Light Logic with naive set theory:

- In enriching Light Logic, the least requirement is that axioms and inference rules newly introduced should be compatible with cut-elimination, since otherwise the paradigm of proofs-as-programs could not be maintained. In this respect, the inference rules of naive set theory are perfectly suitable; they by no means disturb the cut-elimination procedure of Light Logic.

- Naive set theory is a powerful specification language. The descriptive expressivity of naive set theory allows us to define various mathematical objects such as natural numbers, words and functions in it. Indeed, it is so rich that we can fairly think of naive set theory based on Light Logic as a general framework for polytime mathematics, a framework in which many branches of mathematics can be developed in so far as the polytime concepts and functions are concerned.

[8]Proofnets (of **LLL**) contain formulas. Hence proofnets themselves are not polytime normalizable. A solution suggested by [Gir98] is to work with untyped proofnets (with formulas erased) in the actual computation. When the conclusion is lazy, the formulas can be automatically recovered after normalization, and such formulas are not exponentially large. Our approach is essentially the same, but we start by a type-free setting, then consider typing afterwards.

[9]The problem is undecidable for System F in the Curry style [Wel94].

[10]A problem is, however, that the connection between proofs and terms is loosened in our approach. In particular, proving the subject reduction property is a bit difficult.

The main difficulty of naive set theory is that it is inconsistent with Classical and Intuitionistic Logics, as witnessed by so-called Russell's paradox. It is, however, known that the existence of Russell's formula does not necessarily imply contradiction if the use of Contraction is somehow limited. Indeed, naive set theory is consistent with Light Logic, where the use of Contraction is severely restricted by means of light exponentials.

In the appendix of [Gir98], Girard initiated the study of *Light Set Theory*, that is naive set theory based on Light Logic, aiming at a general framework for feasible mathematics. In that paper, he showed the fundamental theorem, called the fixpoint theorem, and in view of this theorem he claimed that various sets and functions are definable in Light Set Theory. This is, however, merely the very beginning of the story, and much has to be done to convince ourselves that Light Set Theory is a suitable general framework for feasible mathematics. First, he did not carry out the full development of Light Set Theory. That is no doubt a tedious work, but someone must carry it out. Second, he considered naive set theory based on **LLL**, but things become much simpler if we consider it based on **ILAL**, since we can freely use Weakening in **ILAL**. Third, function definability itself does not reflect the intrinsically feasible character of Light Set Theory, since all recursive functions are definable in it [Shi99]. The aim of Chapter 5 is to supplement Girard's work in order to confirm the truly polytime character of Light Set Theory. We shall introduce **LST**, which is naive set theory based on **ILAL** (rather than **LLL**), and show that the polytime functions are not only definable, but also *provably total* in **LST**.

One of the main advantages of developing a mathematical theory based on a constructive logical system is that one can automatically extract algorithmic content from a given proof. In Chapter 6, we shall demonstrate the *program extraction* method for **LST**. In our case, what we can extract is a term of λLA. Since any term of λLA is polytime computable, this means that we can extract a polytime program. The program extraction method is obtained by extending the proofs-as-programs interpretation, which is demonstrated for **ILAL2** in Chapter 4, to **LST**. In particular, from a proof of the totality of function f in **LST**, we can automatically extract a term of λLA which represents f. Therefore, in some sense, proving a theorem which states a function's totality is equivalent to writing a (certified) program which computes that function.

Phase semantics and the finite model property for systems of Light Logic (Chapter 6). Phase semantics was originally introduced by Girard [Gir87] as a sound and complete semantics for Linear Logic. Later on, it was modified and adapted for various related logical systems; just to mention a few examples, it was adapted for Intuitionistic Linear Logic in [Abr90, Tro92, Ono94, Oka96, Oka99], and for Affine Logic in [Ono94, Laf97, Oka01]. Higher order versions were introduced in [Oka96, Oka99].

Phase semantics, which was initially thought as "abstract nonsense" (see [Gir99a]), has later found many interesting applications. For example, it was employed to give a semantic proof to the cut-elimination theorem in [Oka96, Oka99, Oka01], to show undecidability of the second order Linear Logic without exponentials in [Laf96]. Various finite model property results were given in [Laf97] and [OT99]. A mixture of phase semantics and coherent semantics gave rise to the denotational completeness for Linear Logic [Gir99a]. Phase semantics is also helpful in understanding certain interesting syntactic phenomena such as polarity and focalization (see [And92], [Gir99b]).

In view of these applications, it is important to investigate phase semantics and its possible applications for Light Logic. Phase semantics for **LLL** and **ILLL** have already been introduced by Kanovitch, Okada and Scedrov [KOSar] (and called *fibred phase semantics*), but not yet for **ILAL**. In Chapter 7, we shall introduce phase semantics for **ILAL**, called *light affine phase semantics*. It is naturally obtained from those for Intuitionistic Linear Logic, Affine Logic and Light Linear Logic. Although the interpretation of light exponentials directly comes from [KOSar], it is considerably simplified in our case, since our target syntax **ILAL** is much simpler than **LLL** and **ILLL**. As an example of applications, we shall show the finite model property for **ILAL**, which implies decidability of **ILAL**. The latter is particularly important in the context of **ILAL** as a type assignment system for λLA, since it means that the type inhabitation problem for **ILAL** is decidable.

Chapter 2

Syntax of Light Logic

In this chapter, we shall describe syntax of Light Logic. Among systems of Light Logic, we are mainly concerned with Intuitionistic Light Affine Logic (**ILAL**) [Asp98, AR00] and its extensions, which we think to be most suitable as a basic framework for feasible computation and reasoning. **ILAL** is a simplification of the intuitionistic version of Light Linear Logic (**ILLL**) [Gir98], while the latter may be understood as a subsystem of the intuitionistic version of Linear Logic (**ILL**). And also, affinity of **ILAL** comes from the intuitionistic version of Affine Logic (**IAL**). Being such a system, **ILAL** inherits many ideas, notions and properties from **ILLL**, **ILL** and **IAL**. Hence for systematic presentation, it is appropriate to begin with these precursors of **ILAL** and then to proceed step by step towards **ILAL**. In addition, we shall state decidability results for these systems. Most of them are already known or immediate consequences of known ones. Nevertheless, we think it important to mention them explicitly for the following reasons:

- These decidability results clearly show that reasoning in systems of Light Logic is as complex as systems of Linear/Affine Logic; replacing exponentials of Linear Logic with light exponentials does not make easier to prove or refute statements.
- As we shall show below, **ILLL** is undecidable while **ILAL** is decidable. These facts confirm the relative simplicity of **ILAL** compared with **ILLL** on the theoretical ground.
- Decidability results have implications on type inhabitation problems.

In Section 2.1, we shall recall syntax of **ILL** as well as its second order extension **ILL2**. In Section 2.2, we shall describe syntax of **IAL**, and investigate the effects of adding Unrestricted Weakening to the inference rules. In Section 2.3, we move on to **ILLL**, and investigate the most important modal connectives of Light Logics, called light exponentials. Undecidability of **ILLL** is shown, too. In Section 2.4, we shall arrive at our target logical system, **ILAL**. In addition to its syntax, the central idea of Light Logic, stratification of proofs, is informally explained. Section 2.5 concludes the present chapter.

2.1 Preliminary 1: Intuitionistic Linear Logic

In this section we shall recall syntax and basic ideas of **ILL**. Our explanation below is essentially based on [Gir87, Gir95]. The undecidability result (Theorem 2.2) is due to [LMSS92]. See also [Tro92].

Identity and Cut:

$$\frac{}{A \vdash A}\ Id \qquad \frac{\Gamma_1 \vdash A \quad A, \Gamma_2 \vdash \Delta}{\Gamma_1, \Gamma_2 \vdash \Delta}\ Cut$$

Structural Rules:

$$\frac{\Gamma \vdash C}{!A, \Gamma \vdash C}\ Weak \qquad \frac{!A, !A, \Gamma \vdash C}{!A, \Gamma \vdash C}\ Contr$$

Multiplicatives:

$$\frac{A, B, \Gamma \vdash C}{A \otimes B, \Gamma \vdash C}\ \otimes l \qquad \frac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \otimes B}\ \otimes r \qquad \frac{\Gamma \vdash C}{\mathbf{1}, \Gamma \vdash C}\ \mathbf{1}l \qquad \frac{}{\vdash \mathbf{1}}\ \mathbf{1}r$$

$$\frac{\Gamma_1 \vdash A \quad B, \Gamma_2 \vdash C}{A \multimap B, \Gamma_1, \Gamma_2 \vdash C}\ \multimap l \qquad \frac{A, \Gamma \vdash B}{\Gamma \vdash A \multimap B}\ \multimap r$$

Additives:

$$\frac{A, \Gamma \vdash C \quad B, \Gamma \vdash C}{A \oplus B, \Gamma \vdash C}\ \oplus l \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B}\ \oplus r_1 \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \oplus B}\ \oplus r_2 \qquad \frac{}{\mathbf{0}, \Gamma \vdash C}\ \mathbf{0}l$$

$$\frac{A, \Gamma \vdash C}{A \,\&\, B, \Gamma \vdash C}\ \& l_1 \qquad \frac{B, \Gamma \vdash C}{A \,\&\, B, \Gamma \vdash C}\ \& l_2 \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}\ \& r \qquad \frac{}{\Gamma \vdash \top}\ \top r$$

Exponentials:

$$\frac{A, \Gamma \vdash C}{!A, \Gamma \vdash C}\ !l \qquad \frac{!\Gamma \vdash A}{!\Gamma \vdash !A}\ !r$$

Here $!\Gamma$ stands for a multisets of formulas of the form $!A_1, \ldots, !A_n$.

Figure 2.1: Inference Rules of Intuitionistic Linear Logic (**ILL**)

2.1.1 Syntax of ILL

Definition 2.1 The *formulas of Intuitionistic Linear Logic* (**ILL**) are defined as follows;

- Propositional variables $\alpha, \beta, \gamma, \ldots$ are formulas of **ILL**.
- $\mathbf{1}$, $\top$, $\mathbf{0}$ are formulas of **ILL**.
- If A and B are formulas of **ILL**, then so are $A \otimes B$, $A \multimap B$, $A \,\&\, B$, $A \oplus B$ and $!A$.

Throughout this thesis, formulas are denoted by capital letters $A, B, C, \ldots$, and multisets of formulas are by Greek capital letters $\Gamma, \Delta, \Sigma, \ldots$. The multiset union of Γ and Δ is simply denoted by Γ, Δ. If $\Gamma \equiv A_1, \ldots, A_n$, then $!\Gamma$ denotes the multiset $!A_1, \ldots, !A_n$. A *sequent* is of the form $\Gamma \vdash C$. Since Γ is a multiset, sequents are considered up to Exchange, namely the exchange rule may be applied implicitly. The *inference rules of* **ILL** are those given in Figure 2.1. The notions of *proof* and *provability* are just as usual, for which we refer to [Tak87, ST96].

Apart from atomic formulas, the formulas of **ILL** are classified into three groups:

Multiplicatives: $A \otimes B$ (multiplicative conjunction), $A \multimap B$ (implication), $\mathbf{1}$ (one).

Additives: $A \,\&\, B$ (additive conjunction), $A \oplus B$ (additive disjunction), $\top$ (top), $\mathbf{0}$ (zero).

Exponentials: $!A$ (of course).

The distinction between multiplicatives and additives reflects the two ways of dealing with context formulas in the traditional sequent calculi (for Classical and Intuitionistic Logics). For example, in Intuitionistic Logic, conjunction $A \wedge B$ can be introduced on the right hand side of a sequent in the following two ways:

$$(1)\ \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \wedge B} \qquad (2)\ \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$$

In (1), two contexts Γ and Δ in the premise sequents may be arbitrary and are concatenated in the conclusion sequent, while in (2), the contexts must be identical in the two premises and are shared in the conclusion. The first style of context management is called *multiplicative*, and the second style is called *additive*. These two are equivalent in the presence of the structural rules Contraction and Weakening. However, they are strictly distinguished in **ILL** since **ILL** is sensitive as to the use of the structural rules. Therefore, the conjunction $\wedge$ of Classical/Intuitionistic Logic splits into two connectives, multiplicative $\otimes$ and additive $\&$, in **ILL**.

It is possible to consider additive implication (see, e.g., [Sch94]) too, but it is customarily omitted. On the other hand, multiplicative disjunction $\invamp$ and its unit $\bot$ are authentic to Classical Linear Logic, and does not arise naturally in **ILL**.

Let us give some examples of provable formulas in **ILL**. Below, $A \multimapboth B$ abbreviates $(A \multimap B) \otimes (B \multimap A)$.

- Connectives $\otimes$, $\&$ and $\oplus$ are commutative, associative and have units $\mathbf{1}$, $\top$ and $\mathbf{0}$ respectively:

Commutativity: $A \otimes B \multimapboth B \otimes A$, $A \,\&\, B \multimapboth B \,\&\, A$, $A \oplus B \multimapboth B \oplus A$.

Associativity: $(A \otimes B) \otimes C \multimapboth A \otimes (B \otimes C)$, $(A \& B) \& C \multimapboth A \& (B \& C)$, $(A \oplus B) \oplus C \multimapboth A \oplus (B \oplus C)$.

Unit: $\mathbf{1} \otimes A \multimapboth A$, $\top \,\&\, A \multimapboth A$, $\mathbf{0} \oplus A \multimapboth A$.

The associativity allows us to omit some parentheses; for example we may write $A \otimes B \otimes C$ in place of $A \otimes (B \otimes C)$ and $(A \otimes B) \otimes C$.

- Multiplicatives satisfy:

Adjointness: $(A \otimes B \multimap C) \multimapboth (A \multimap (B \multimap C))$.

- Additives distribute over multiplicatives in the following ways:

Distributivity: $A \otimes (B \oplus C) \multimapboth (A \otimes B) \oplus (A \otimes C)$; $A \multimap (B \,\&\, C) \multimapboth (A \multimap B) \,\&\, (A \multimap C)$.

There are plenty of formulas which are provable in Intuitionistic Logic but whose counterparts in **ILL** are not provable. Of particular importance are the following formulas corresponding to Weakening and Contraction:

Unrestricted Weakening: $A \multimap \mathbf{1}$.

Unrestricted Contraction: $A \multimap A \otimes A$.

These are not provable in **ILL**.

Let us come to the exponential modality. The crux of Linear Logic is to control the use of structural rules, Contraction and Weakening, on the object level. This is achieved by putting the modal operator ! onto those formulas to which structural rules are to be applied. Thus, to apply Contraction to a formula A, we first need to put ! upon A by rule $(!l)$ (called Dereliction). Formulas introduced by Weakening are also marked with !. The right rule $(!r)$ is designed so that it is compatible with Dereliction, Contraction and Weakening with respect to the cut-elimination procedure:

$(!r{-}!l)$:

$$\dfrac{\dfrac{\overset{\vdots}{!\Gamma \vdash A}}{!\Gamma \vdash !A}\,(!r) \qquad \dfrac{\overset{\vdots}{A, \Delta \vdash C}}{!A, \Delta \vdash C}\,(!l)}{!\Gamma, \Delta \vdash C} \quad\longrightarrow\quad \dfrac{\overset{\vdots}{!\Gamma \vdash A} \qquad \overset{\vdots}{A, \Delta \vdash C}}{!\Gamma, \Delta \vdash C}$$

$(!r - Contr)$:

$$\dfrac{\dfrac{\overset{\vdots\ \pi}{!\Gamma \vdash A}}{!\Gamma \vdash !A}\,(!r) \qquad \dfrac{\overset{\vdots}{!A, !A, \Delta \vdash C}}{!A, \Delta \vdash C}\,(Contr)}{!\Gamma, \Delta \vdash C} \quad\longrightarrow\quad \dfrac{\dfrac{\dfrac{\overset{\vdots\ \pi}{!\Gamma \vdash A}}{!\Gamma \vdash !A}\,(!r) \qquad \dfrac{\dfrac{\overset{\vdots\ \pi}{!\Gamma \vdash A}}{!\Gamma \vdash !A}\,(!r) \qquad \overset{\vdots}{!A, !A, \Delta \vdash C}}{!A, !\Gamma, \Delta \vdash C}}{!\Gamma, !\Gamma, \Delta \vdash C}}{!\Gamma, \Delta \vdash C}\,(Contr)$$

$(!r - Weak)$:

$$\dfrac{\dfrac{\overset{\vdots}{!\Gamma \vdash A}}{!\Gamma \vdash !A}\,(!r) \qquad \dfrac{\overset{\vdots}{\Delta \vdash C}}{!A, \Delta \vdash C}\,(Weak)}{!\Gamma, \Delta \vdash C} \quad\longrightarrow\quad \dfrac{\overset{\vdots}{\Delta \vdash C}}{!\Gamma, \Delta \vdash C}\,(Weak)$$

Note that, in the second case, the subproof π is duplicated and n occurrences of $(Contr)$ are newly created, where $n = |!\Gamma|$.

The modality ! satisfies **S4** axioms of Modal Logic. In more detail, it is characterized by the following principles:

Functricity: $A \multimap B$ implies $!A \multimap !B$

Monoidalness 1: $!A \otimes !B \multimap !(A \otimes B)$

Monoidalness 2: $!\mathbf{1}$

Dereliction: $!A \multimap A$

Digging: $!A \multimap !!A$

Weakening: $!A \multimap \mathbf{1}$

Second Order Quantifiers:

$$\frac{A[B/\alpha], \Gamma \vdash C}{\forall\alpha.A, \Gamma \vdash C}\ \forall l \quad \frac{\Gamma \vdash A}{\Gamma \vdash \forall\alpha.A}\ \forall r$$

$$\frac{A, \Gamma \vdash C}{\exists\alpha.A, \Gamma \vdash C}\ \exists l \quad \frac{\Gamma \vdash A[B/\alpha]}{\Gamma \vdash \exists\alpha.A}\ \exists r$$

In rule ($\forall r$), α is not free in Γ. In rule ($\exists l$), α is not free in Γ and C.

Figure 2.2: Inference Rules for Second Order Quantifiers

Contraction: $!A \multimap !A \otimes !A$

Here, Monoidalness 1 and 2 correspond to axiom **K** (in the presence of Functricity), Dereliction to axiom **T**, and Digging to axiom **4**.

Since the exponential modality allows us to use structural rules, it is naturally expected that multiplicatives and additives which are previously distinguished are somehow identifiable in the presence of !. Indeed we have:

Exponential Isomorphism: $!A \otimes !B \circ\!\!-\!\!\circ\ !(A \,\&\, B)$,

which is in analogy with the equation $2^x \cdot 2^y = 2^{x+y}$ in number theory.

The modality-free fragment of **ILL** is called **IMALL** (the Multiplicative-Additive fragment of Intuitionistic Linear Logic).

ILL and its classical counterpart Classical Linear Logic are undecidable. Indeed, a considerably restricted fragment of **ILL** is already undecidable. Say that a sequent is *!-prenex* if it is of the form $!\Gamma, \Delta \vdash C$ where all formulas in Γ, Δ, C are !-free.

Theorem 2.2 (Lincoln et al. [LMSS92]) *The provability of !-prenex sequents in* **ILL** *is undecidable. Hence* **ILL** *is undecidable, too.*

The proof consists in encoding a sort of Minsky's counter machines, whose halting problem is known to be undecidable.

2.1.2 Second Order Quantification

The expressive power of **ILL** is surely limited under the proofs-as-programs interpretation. **ILL** is essentially a refinement of Intuitionistic Logic (see, e.g., [Abr93]), and the proofs of the latter may well be identified with the terms of simply typed λ-calculus (see, e.g., [GLT88]). It is well-known that simply typed λ-calculus cannot represent a nonmonotonic function such as predecessor [Sta79], and the same is true of the proofs of **ILL**. Hence, it is natural to look for a suitable extension of **ILL** which covers a sufficiently wide range of functions. The most preferable extension is with second order quantifiers.

Definition 2.3 The *formulas of the second order Intuitionistic Linear Logic* (**ILL2**) are defined analogously to those of **ILL**, with the following clause attached:

- If A is a formula of **ILL2** and α is a propositional variable, then $\forall\alpha.A$ and $\exists\alpha.B$ are formulas of **ILL2**.

Let $A[B/\alpha]$ denote the formula obtained by substituting B for the free occurrences of α in A. The inference rules of **ILL2** are those of **ILL** toghether with the rules in Figure 2.2.

In what follows, formulas are considered up to α-equivalence, hence two formulas which differ only in the names of bound variables are identified (see, e.g., [ST96]). Moreover, we adopt the so-called *variable convention* [Bar81], so that substitution of formulas never causes a variable to be newly bounded.

ILL2 is a refinement of the second order Intuitionistic Logic, which corresponds to Girard's System F [Gir72], also known as second order polymorphic typed lambda calculus. Hence the proofs of **ILL2** represent all the provably total functions of second order Peano Arithmetic (see [GLT88] for the background).

2.2 Preliminary 2: Intuitionistic Affine Logic

There is a variant of **ILL**, called *Intuitionistic Affine Logic*, denoted by **IAL** (see, e.g., [Ono94, Kop95, Laf97, Oka01]). In this section, we shall recall syntax and basic properties of **IAL**. The undecidability result (Theorem 2.4) for the second order **IAL** is essentially due to [LSS95].

2.2.1 Syntax of IAL

There is a variant of **ILL**, called *Intuitionistic Affine Logic* (**IAL**), which is obtained from **ILL** by allowing Weakening for It is obtained from **ILL** by allowing Weakening for arbitrary formulas without any restriction.

$$\frac{\Gamma \vdash C}{A, \Gamma \vdash C}\ (Weak).$$

In what follows, the rule will be referred to as *Unrestricted Weakening.* Adding this rule means that we give up the control over Weakening via the modality, and we concentrate on Contraction. In **IAL**, the following are provable:

- $\top \multimapboth \mathbf{1}$;
- $A \otimes B \multimap A \,\&\, B$.

The modality-free fragment is called **IMAAL**(the Multiplicative-Additive fragment of Intuitionistic Affine Logic).

Adding Unrestricted Weakening does not alter the expressive power of the system under the proofs-as-programs interpretation. On the other hand, it significantly simplifies reasoning in the system; indeed, **IAL** is decidable while **ILL** is not, as we shall see in Chapter 7.

However, once **IAL** is extended to the second order version **IAL2**, one immediately gets an undecidability result:

Theorem 2.4 (Lincoln et al. [LSS95]) IAL2 *is undecidable. Indeed, its multiplicative fragment* **IMAL2** *is already undecidable.*

The proof amounts to interpreting the second order Intuitionistic Logic, which is known to be undecidable [Löb76], in **IMAL2**. The basic idea is that Contraction can be simulated by means of three copies of the following second order formula C:

$$C \;\equiv\; \forall\alpha.(\alpha \multimap \alpha \otimes \alpha).$$

Hence, with the help of $C \otimes C \otimes C$, **IMAL2** can simulate second order Intuitionistic Logic.

In **IAL2**, all logical connectives and constants are definable from $\{\multimap, !, \forall\}$:

$$\begin{array}{rcl}
\exists\beta.A & \equiv & \forall\alpha.(\forall\beta.(A \multimap \alpha) \multimap \alpha);\\
A \otimes B & \equiv & \forall\alpha.((A \multimap B \multimap \alpha) \multimap \alpha);\\
\mathbf{1} & \equiv & \forall\alpha.(\alpha \multimap \alpha);\\
A \,\&\, B & \equiv & \exists\alpha.((\alpha \multimap A) \otimes (\alpha \multimap B) \otimes \alpha);\\
A \oplus B & \equiv & \forall\alpha.((A \multimap \alpha) \multimap (B \multimap \alpha) \multimap \alpha);\\
\mathbf{0} & \equiv & \forall\alpha.\alpha,
\end{array}$$

where α is a fresh variable which does not occur in A and B. Similar second order definitions are also available in **ILL2**, but an important difference is that none of the above definitions uses !, whereas the definitions of additives in **ILL2** crucially use !. For example, $A \oplus B$ is defined in **ILL2** as follows:

$$A \oplus B \;\equiv\; \forall\alpha.(!(A \multimap \alpha) \multimap !(B \multimap \alpha) \multimap \alpha).$$

In the presence of Unrestricted Weakening, the use of ! may be avoided. This simplicity of second order definitions is one of the reasons why we prefer an affine system rather than a linear system when it comes to Light Logic.

The main defect of adding Unrestricted Weakening is that it makes the cut-elimination procedure nondeterministic in the classical framework. Typically, in Classical Affine Logic, the proof

$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \pi_1\\ \vdash A\end{array}}{\vdash A, B}\,(Weak) \quad \dfrac{\begin{array}{c}\vdots\ \pi_2\\ \vdash A\end{array}}{B \vdash A}\,(Weak)}{\vdash A, A}$$

reduces to two entirely different proofs

$$\dfrac{\begin{array}{c}\vdots\ \pi_1\\ \vdash A\end{array}}{\vdash A, A}\,(Weak) \qquad \text{and} \qquad \dfrac{\begin{array}{c}\vdots\ \pi_2\\ \vdash A\end{array}}{\vdash A, A}\,(Weak)$$

which cannot be identified in any sense. Therefore the result of cut-elimination is not unique, which means that proofs cannot be seen as programs, since programs are supposed to evaluate to the unique outputs. Such a difficulty does not arise in **IAL**, which does not have Weakening on the right hand side.

2.3 Intuitionistic Light Linear Logic

Let us now come to the historically first system of Light Logic, that is Light Linear Logic [Gir98]. The idea of the new system is to constrain the use of Contraction in such a way that a proof containing Contraction never reduces to an exponentially large one via the cut-elimination procedure, while allowing all polytime functions to be represented as proofs. Since !-modality of Linear Logic controls Contraction, this is achieved by constraining !-modality. Girard introduced both Classical Light Linear Logic (**LLL**) and its restriction to the intuitionistic fragment, Intuitionistic Light Linear Logic (**ILLL**). Here we shall only describe the latter, whose second order extension is already sufficient for representing all the polytime functions. Moreover, we shall only give it an axiomatic definition, since its sequent calculus formulation is quite complicated.

The following exposition owes much to [Gir98]. However, the undecidability result (Corollary 2.7) seems to be original (though immediate).

2.3.1 Syntax of ILLL

ILLL is obtained by replacing the exponential modality of **ILL** with its light version; first, disregard Monoidalness 1 and 2, Dereliction and Digging, and add one direction of Exponential Isomorphism explicitly, to the effect that we have:

Functricity: $A \multimap B$ implies $!A \multimap !B$

Weakening: $!A \multimap \mathbf{1}$

Contraction: $!A \multimap !A \otimes !A$

Exponential Isomorphism: $!A \otimes !B \multimap !(A \,\&\, B)$

The other direction of Exponential Isomorphism is derivable by Functricity and Contraction. Second, introduce an auxiliary modality § with the following principles:

Functricity§: $A \multimap B$ implies $\S A \multimap \S B$

Monoidalness§: $\S A \otimes \S B \multimap \S(A \otimes B)$ and $\S\mathbf{1}$

Weak Dereliction: $!A \multimap \S A$

The new modalities ! and § thus introduced are called the *light exponentials.* The second order extension is denoted by **ILLL2**. Here are some remarks.

- The fundamental concept of Light Logic is *stratification of proofs.* This is achieved by rejecting Dereliction and Digging (see Subsection 2.4.2 below).
- Two Monoidalness principles are also rejected. Both are quite harmless with regard to stratification, but Monoidalness 1 causes an exponential explosion in the process of cut-elimination. On the other hand, Monoidalness 2 is not quite compatible with Exponential Isomorphism. Hence they are refused.
- The rejection of Monoidalness is too serious a restriction. To compensate for this, we need an extra connective §, which satisfies Monoidalness (i.e., axiom **K** of Modal Logic). A weak form of Dereliction is also included, which plays a role of correlating ! and §.

- Exponential Isomorphism is not derivable without Dereliction and Digging, while it is required for representing certain basic functions such as predecessor. Hence we need to add it explicitly.

It is not so easy to integrate these principles into a sequent calculus formalism which admits cut-elimination, with the main difficulty being Exponential Isomorphism. A solution given by Girard is a sort of *hybrid* sequent calculus which can express both multiplicative and additive combinations of formulas in one and the same sequent. The system works, but it is complicated. Moreover, it cannot accommodate Monoidalness 2, which is perfectly harmless with regard to the computational complexity of cut-elimination and quite useful in representing the polytime functions. Indeed, due to the lack of Monoidalness 2, representation of functions in **ILLL2** is somewhat awkward, in that it leaves a lot of garbages $!\mathbf{1}$ in conclusion sequents; typically predecessor is represented by a proof of conclusion $!\mathbf{1}, \mathbf{int} \vdash \mathbf{int}$, and the garbage $!\mathbf{1}$ cannot be removed.

A variant of **LLL** (and **ILLL**) with Monoidalness 1 and 2 is known as Elementary Linear Logic [Gir98, DJ99] (and Intuitionistic Elementary Linear Logic) whose second order extension precisely characterizes the elementary recursive functions.

2.3.2 Undecidability of ILLL

ILLL may be seen as a subsystem of **ILL** in the following sense:

Proposition 2.5 *If A is provable in* **ILLL**, *then A^- is provable in* **ILL**, *where A^- is a formula of* **ILL** *which is obtained by removing all §'s from A.*

Proof. **ILLL** and **ILL** share the modality-free fragment. Moreover, the axioms for light exponentials are mapped to provable formulas of **ILL** via the $^-$ translation. ■

Furthermore, **ILLL** and **ILL** are "equivalent" as far as the !-prenex sequents (see Section 2.1) are concerned. For $n \geq 0$, let Γ^n be the multiset $\underbrace{\Gamma, \ldots, \Gamma}_{n \text{ times}}$. If $\Gamma \equiv A_1, \ldots, A_n$, let $\Gamma \,\&\, \mathbf{1}$ be the multiset $A_1 \,\&\, \mathbf{1}, \ldots, A_n \,\&\, \mathbf{1}$.

Proposition 2.6 *Let $!\Gamma, \Delta \vdash C$ be a !-prenex sequent. Then the following are equivalent:*

(1) $!\Gamma, \Delta \vdash C$ *is provable in* **ILL**;

(2) $(\Gamma \,\&\, \mathbf{1})^n, \Delta \vdash C$ *is provable in* **IMALL** *for some n;*

(3) $!(\Gamma \,\&\, \mathbf{1}), \S\Delta \vdash \S C$ *is provable in* **ILLL**.

Proof.
Statement (1) implies statement (2). This is easily proved by induction on the length of the cut-free proof of $!\Gamma, \Delta \vdash C$. (2) implies (3) by rules (§) and (*Contr*) of **ILLL**. (3) implies (1) by Proposition 2.5, together with the fact that $!A \multimapboth !(A \,\&\, \mathbf{1})$ is provable in **ILL**. ■

In conjunction with Theorem 2.2 (undecidability of the !-prenex fragment of **ILL**), we conclude:

Corollary 2.7 **ILLL** *is undecidable.*

By a similar argument, we can also show that **LLL** is undecidable.

2.4 Intuitionistic Light Affine Logic

By inspecting the representation of the polytime functions in [Gir98], we see that Exponential Isomorphism is used only in the form:

$$!\mathbf{1} \otimes !B \multimap !(\mathbf{1} \mathbin{\&} B).$$

The leftmost $!\mathbf{1}$ is a garbage, hence it may be ignored. Therefore Exponential Isomorphism is, in essence, only required to derive $!B \multimap !(\mathbf{1} \mathbin{\&} B)$. Now suppose that we have Unrestricted Weakening, namely the underlying logic be affine rather than linear. Then $B \multimap \mathbf{1} \mathbin{\&} B$ is derivable, hence by Functricity $!B \multimap !(\mathbf{1} \mathbin{\&} B)$ is derivable without Exponential Isomorphism. This means that we can dispense with the problematic Exponential Isomorphism in the presence of Unrestricted Weakening. This observation leads us to **ILAL**. The definition of **ILAL** below is due to [Asp98]. See also [AR00] for a good exposition of the system.

2.4.1 Syntax of ILAL

ILAL consists of **IMAAL** endowed with the light exponentials $!, \S$. But this time $!$ satisfies Monoidalness 2 in addition to Functricity, Weakening and Contraction. $\S$ is just the same as in **ILLL**. Here are some remarks.

- As explained above, we no more need Exponential Isomorphism, since its weaker form is derivable and that is sufficient for representing the polytime functions. Hence we no more need to consider hybrid sequents; just standard sequents suffice.
- And also, we are allowed to have Monoidalness 2, since the only reason why we did not adopt it was that it was not compatible with Exponential Isomorphism. As a result, representation is considerably simplified; in particular, it is garbage-free.
- The cost of adding Unrestricted Weakening is, again, the nondeterminism of cut-elimination in the classical case, i.e. Classical Light Affine Logic **LAL**. Hence, in order to maintain the proofs-as-programs paradigm, we are compelled to stick to the intuitionistic version, in contrast to the case of Light Liner Logic, for which both classical and intuitionistic formulations are possible.

ILAL is formally defined as follows [Asp98].

Definition 2.8 The *formulas of Intuitionistic Light Affine Logic* (**ILAL**) are defined as follows;

- Propositional variables $\alpha, \beta, \gamma, \ldots$ are formulas of **ILAL**.
- $\mathbf{1}$, $\top$, $\mathbf{0}$ are formulas of **ILAL**.
- If A and B are formulas of **ILAL**, then so are $A \otimes B$, $A \multimap B$, $A \mathbin{\&} B$, $A \oplus B$, $!A$ and $\S A$.

The inference rules of **ILAL** are those given in Figure 2.3.

We may also consider the second order extension of **ILAL**, analogously to **ILL2**. The resulting system is denoted by **ILAL2**. **ILAL2** is undecidable, as an easy extension of the undecidability result for **IAL2** (Theorem 2.4).

Identity and Cut:

$$\frac{}{A \vdash A}\ Id \qquad \frac{\Gamma_1 \vdash A \quad A, \Gamma_2 \vdash C}{\Gamma_1, \Gamma_2 \vdash C}\ Cut$$

Structural Rules:

$$\frac{\Gamma \vdash C}{A, \Gamma \vdash C}\ Weak \qquad \frac{!A, !A, \Gamma \vdash C}{!A, \Gamma \vdash C}\ Contr$$

Multiplicatives:

$$\frac{A, B, \Gamma \vdash C}{A \otimes B, \Gamma \vdash C}\ \otimes l \qquad \frac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \otimes B}\ \otimes r \qquad \frac{\Gamma \vdash C}{\mathbf{1}, \Gamma \vdash C}\ \mathbf{1}l \qquad \frac{}{\vdash \mathbf{1}}\ \mathbf{1}r$$

$$\frac{\Gamma_1 \vdash A \quad B, \Gamma_2 \vdash C}{A \multimap B, \Gamma_1, \Gamma_2 \vdash C}\ \multimap l \qquad \frac{A, \Gamma \vdash B}{\Gamma \vdash A \multimap B}\ \multimap r$$

Additives:

$$\frac{A, \Gamma \vdash C \quad B, \Gamma \vdash C}{A \oplus B, \Gamma \vdash C}\ \oplus l \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B}\ \oplus r_1 \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \oplus B}\ \oplus r_2 \qquad \frac{}{\mathbf{0}, \Gamma \vdash C}\ \mathbf{0}l$$

$$\frac{A, \Gamma \vdash C}{A \,\&\, B, \Gamma \vdash C}\ \& l_1 \qquad \frac{B, \Gamma \vdash C}{A \,\&\, B, \Gamma \vdash C}\ \& l_2 \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}\ \& r \qquad \frac{}{\Gamma \vdash \top}\ \top r$$

Exponentials:

$$\frac{B \vdash A}{!B \vdash !A}\ ! \qquad \frac{\Gamma, \Delta \vdash A}{!\Gamma, \S\Delta \vdash \S A}\ \S$$

In rule (!), B can be absent. In rule (§), Γ and Δ can be empty.

Figure 2.3: Inference Rules of Intuitionistic Light Affine Logic (**ILAL**)

A nice aspect of **ILAL2** is that all logical connectives, in particular additives, are definable from $\{\multimap, !, \S, \forall\}$, just as in **IAL2**. This point is quite useful, especially when it comes to investigating the term calculus for **ILAL2** (in Chapter 4). Note, in contrast, that the second order definitions of additives are not available in **ILLL2**.

2.4.2 Stratification of Proofs

As often pointed out, the main cause of an exponential explosion in cut-elimination lies in an untamed use of Contraction. What is particularly problematic is the phenomenon that Contraction duplicates Contraction, and a copy of Contraction thus obtained by duplication then duplicates another copy, and so on. The fundamental idea of Light Logic is to avoid such a lawless process of duplication by enforcing a hierarchical structure on the occurrences of Contraction in a proof. This is achieved by *stratification of proofs* which now we shall explain.

We may imagine that each proof of **ILAL** is stratified into layers, with two layers separated by rules (!) and (§), as illustrated in Figure 2.4. To each layer, its *depth* $n \geq 0$ is associated, by setting the outermost layer as depth 0, the next layer as depth 1, and so on. The proof in Figure 2.4 consists of three layers, and Contraction is used at depth 1.

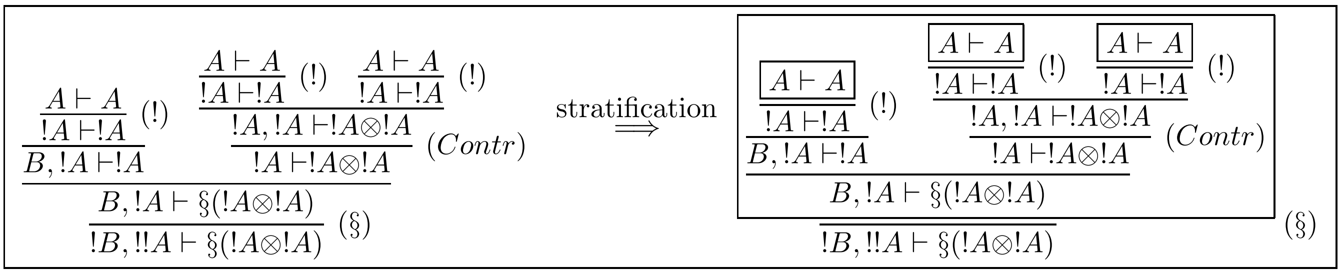
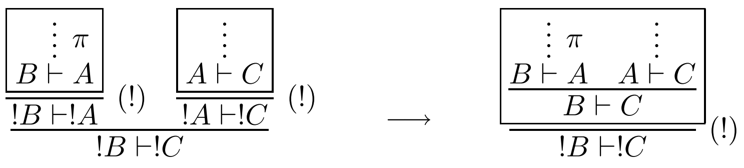


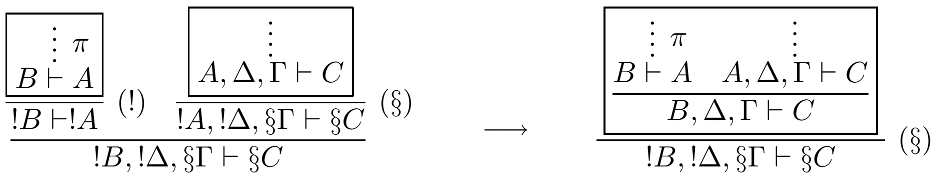
Figure 2.4: Stratification of Proofs

The basic idea is to preserve this stratified structure during cut-elimination. Indeed, the inference rules for light exponentials are carefully designed in such a way that the depth of a subproof is always preserved by a reduction step; see the following principal reduction steps of cut-elimination for light exponentials:

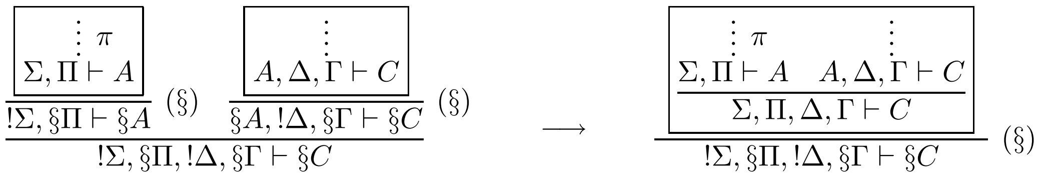
(!−!):



(! − §):

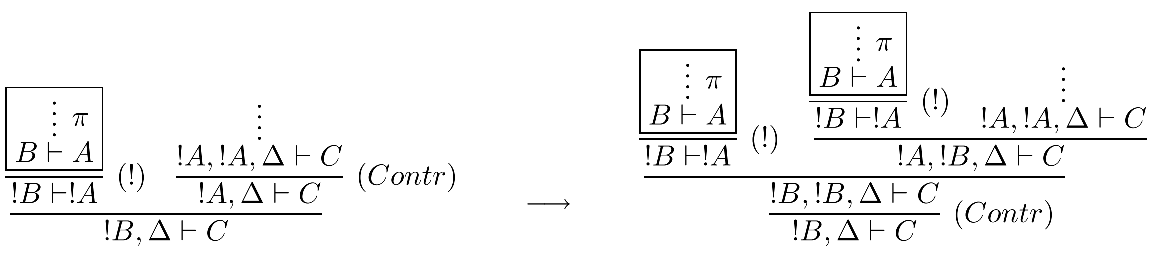


(§ − §):



In all cases, reduction preserves the depth of the subproof π. This means that an occurrence of Contraction cannot go up or down to another layer. On the other hand, an occurrence of Contraction can duplicate only a subproof π at a deeper layer:

$(! - Contr)$:



Linear and Affine Logics	Light Logic
ILL: no	**ILLL**: no
ILL2: no	**ILLL2**: no
IAL: yes	**ILAL**: yes
IAL2: no	**ILAL2**: no

Table 2.1: Decidability of Linear, Affine and Light Logics

Thus an occurrence of Contraction can duplicate another one at a deeper layer only. In this way, a hierarchical structure of Contraction is established, which has the effect of drastically lowering the complexity of normalization. Note that stratification is easily disturbed if we have Dereliction and Digging in addition. The effect of stratification will be formally investigated in terms of a term calculus in Chapter 3.

2.5 Remarks

In this chapter, we have described various systems of Linear, Affine and Light Logics. The decidability results for these systems are summarized in Table 2.1 (decidability of **IAL** and **ILAL** will be shown in Chapter 7). The situation is basically the same in the classical cases. To the best of our knowledge, however, it remains open whether the second order Classical Affine Logic and the second oder Classical Light Affine Logic (or even their multiplicative fragment) are decidable or not.

In view of these results, it is fair to say that replacing the exponentials of Linear Logic with the light ones does not alter the complexity of reasoning (it does not make easier to prove/refute a formula). The "light" character of Light Logic is entirely concerned with its computational aspect, according to the proofs-as-programs correspondence.

We have occasionally pointed out how **ILAL** simplifies **ILLL** (and **LLL**). To sum up:

- **ILAL** does not require hybrid sequents as **ILLL** does.
- **ILAL** admits second order definitions of additives, while **ILLL** does not.
- The representation of functions in **ILAL** is considerably simpler than in **ILLL**.
- **ILAL** is decidable while **ILLL** is not.

We have also pointed out that cut-elimination in Classical Light Affine Logic is nondeterministic, hence the proofs-as-programs paradigm cannot be maintained for it. These results should justify our choice of **ILAL** as the basic system for our investigation. In the rest of this thesis, we shall mainly deal with **ILAL** and mention other systems only when they matter.

Chapter 3

Light Affine Lambda Calculus and Polytime Strong Normalization

We shall now turn our attention to the computational aspects of Light Logic, namely those issues concerning the cut-elimination procedure. The intrinsically polytime character of Light Logic is best witnessed by the polytime normalizability of proofs, which means that every proof can be normalized into a cut-free one in polynomial time. It is, however, observed by Girard that the polytime normalizability is largely independent of formulas/types; indeed the proof of the polytime normalizability in [Gir98] does not use induction on the complexity of formulas. This suggests that we could have an untyped calculus which equally satisfies the polytime normalizability.

In this chapter, we shall introduce an untyped term calculus λLA, called *Light Affine λ-Calculus*, which embodies the essentials of Light Logic in an untyped setting. The calculus is a modification of λ-calculus (see [Bar81]), which has very simple operational behavior defined by just 5 reduction rules. We then prove:

- The Polystep Strong Normalization Theorem: every reduction sequence in λLA has a length bounded by a polynomial in the size of its initial term (of fixed depth).
- The Polytime Strong Normalization Theorem: every reduction strategy (given as a function oracle) induces a normalization procedure which terminates in time polynomial in the size of a given term (of fixed depth).

A corollary to the strong normalizability is the Church-Rosser property for λLA. (See Chapter 1 for the theoretical and practical values of these results.) In the next chapter, **ILAL2** will be introduced as a type assignment system for λLA. Thus the results in this chapter also have an impact on the typed setting, i.e., the proof systems of Light Logic.

The rest of this chapter is organized as follows. In Section 3.1, we shall introduce λLA. In Section 3.2 we shall prove the main part of the *polystep* strong normalization theorem. The theorem itself appears in Section 3.3, as well as its direct corollaries, namely the Church-Rosser property and the *polytime* strong normalization theorem.

3.1 Light Affine Lambda Calculus

We begin by giving the set $\mathcal{PT}$ of pseudo-terms (in 3.1.1). Our goal is to define the set $\mathcal{T}$ of well-formed terms (in 3.1.2) and the notion of reduction (in 3.1.3).

3.1.1 Pseudo-Terms

Let $x, y, z \ldots$ range over term variables.

Definition 3.1 The set $\mathcal{PT}$ of *pseudo-terms* is defined by the following grammar:

$$t, u ::= x \mid \lambda x.t \mid tu \mid !t \mid \mathsf{let}\ u\ \mathsf{be}\ !x\ \mathsf{in}\ t \mid \S t \mid \mathsf{let}\ u\ \mathsf{be}\ \S x\ \mathsf{in}\ t.$$

In addition to the standard constructs such as λ-abstraction and application, we have two modal operators ! and § and two let *operators* (called let-! and let-§operators). Pseudo-terms $!t$ and $\S t$ are called *boxes* in analogy with their counterparts in proof-nets. Two let operators are used to connect boxes, thus to make boxes mutually interacting.

It is suggestive to think of $!t$ and $\S t$ as "boxes" in the literal sense and depict them as:

$$!\boxed{t} \qquad \text{and} \qquad \S\boxed{t}.$$

With this intuition, each pseudo-term is *stratified* into layers; for example, $(\mathsf{let}\ !y\ \mathsf{be}\ !x\ \mathsf{in}\ \S!(\lambda z.zx))z$ is stratified into three layers as follows:

$$(\mathsf{let}\ !\boxed{y}\ \mathsf{be}\ !x\ \mathsf{in}\ \S\boxed{!\boxed{\lambda z.zx}})\,w.$$

In the sequel, symbol † stands for either ! or §. Pseudo-terms $(\lambda x.t)$ and $(\mathsf{let}\ u\ \mathsf{be}\ \dagger x\ \mathsf{in}\ t)$ *bind* each occurrence of x in t. As usual, pseudo-terms are considered up to α-equivalence, and the *variable convention* (see [Bar81]) is adopted for the treatment of free/bound variables (namely, the bound variables are chosen to be different from the free variables, so that variable clash is never caused by substitution). Notation $t[u/x]$ is used to denote the pseudo-term obtained by substituting u for the free occurrences of x in t. $FV(t)$ denotes the set of free variables in t. $FO(x,t)$ denotes the number of free occurrences of x in t and $FO(t)$ denotes the number of free occurrences of all variables in t.

As usual, each pseudo-term t is represented as a *term tree*, and each subterm occurrence u in t is pointed by its *address*, i.e., a word $w \in \{0,1\}^*$ which describes the path from the root to the node corresponding to u in the term tree. For example, the term tree for $(\lambda x.\mathsf{let}\ !x\ \mathsf{be}\ !y\ \mathsf{in}\ yy)$ and the addresses in it are illustrated in Figure 3.1.

The *size* $|t|$ of a pseudo-term t is the number of nodes in its term tree. Since our terms are untyped, $|t|$ is not significantly different from the length of its string representation. Given a pseudo-term t and an address w, the *depth* of w in t is the number of !-boxes and §-boxes enclosing the subexpression at w. The *depth* of t is the maximum depth of all addresses in it.

A *context* Φ (see, e.g., CH. 2, §1 of [Bar81]) is a pseudo-term-like expression with one hole $\bullet$. If Φ is a context and t is a pseudo-term, then $\Phi[t]$ denotes the pseudo-term obtained by substituting t for $\bullet$ in Φ.

We write $!^d t$ and $\S^d t$ to denote $\underbrace{!! \cdots !}_{d\ times}\ t$ and $\underbrace{\S\S \cdots \S}_{d\ times} t$, respectively.

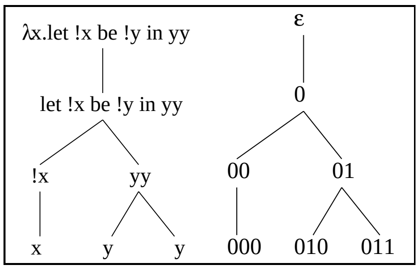


Figure 3.1: Term Tree and Addresses

3.1.2 Terms

Before giving a formal definition of well-formed terms, we shall informally discuss the critical issues.

The fundamental idea of Light Logic is to enforce a stratified structure on proofs/terms and to preserve it in the course of reduction. At the level of formulas/types, this is achieved by rejecting Dereliction and Digging principles of Linear Logic's !-modality. To achieve the same effect in our untyped framework, we first assume that variables are (conceptually) classified into three groups: undischarged, !-discharged, and §-discharged variables. These are to be bound by λ-abstraction, let-! operator and let-§ operator, respectively. Now consider the following conditions.

- *In default, a variable is undischarged, and a variable is made (either !- or §-) discharged when a box is built around it.*

For example, variable x in pseudo-term tx is undischarged, while it is !-discharged in $!(tx)$. To see the effect of this condition, consider the following pseudo-term corresponding to the Dereliction principle:

$$\mathit{dereliction}(x) := \mathsf{let}\ x\ \mathsf{be}\ !y\ \mathsf{in}\ y,$$

whose effect is to open a !-box:

$$\mathit{dereliction}(!\boxed{t}) \longrightarrow t.$$

It is ruled out by the above condition, since variable y is undischarged, but is illegally bound by a let-! operator. On the other hand, the following pseudo-term, which corresponds to the Weak Dereliction principle ($!A \multimap \S A$), is legitimated:

$$\mathsf{let}\ x\ \mathsf{be}\ !y\ \mathsf{in}\ \S\boxed{y},$$

since variable y is discharged in $\S\boxed{y}$, so it can be bound by the $\mathsf{let}_!$ operator.

- *A box may be built around a term only when it contains no free discharged variable.*

Consider the following pseudo-term which corresponds to the Digging principle:

$$\mathit{digging}(x) := \mathsf{let}\ x\ \mathsf{be}\ !y\ \mathsf{in}\ !\boxed{!\boxed{y}},$$

whose effect is to embed a !-box into another:

$$\mathit{digging}(!\boxed{t}) \longrightarrow !\boxed{!\boxed{t}}.$$

It is also ruled out by the above condition, since it attempts to build the outer box $!\boxed{!\boxed{y}}$ around the inner box $!\boxed{y}$, but the latter contains a discharged variable y.

The only duplicable entities in Light Logic are contents of !-boxes, just as in Linear Logic. This is maintained by the following condition:

- *Among three binders, only* let-! *may bind multiple occurrences of (!-discharged) variables.*

With this condition, duplication occurs only when a !-box meets a let-! operator; for example,

$$\mathsf{let}\ !t\ \mathsf{be}\ !x\ \mathsf{in}\ (\S xx)!x \longrightarrow (\S tt)!t.$$

Light Logic rejects Monoidalness 1: $!A \otimes !A \not\multimap !A$, in order to avoid potential exponential growth caused by iterated duplication of !-boxes. To achieve the same effect, our term syntax requires a further constraint on !-boxes:

- *A !-box may be built around a term only when it contains at most one free variable.*

Hence a term construction like

$$\lambda x.(\mathsf{let}\ x\ \mathsf{be}\ !y\ \mathsf{in}\ !yy)$$

which by iteration causes exponential growth is ruled out.

To compensate for this severe constraint, we need another kind of boxes, namely §-boxes. They are not duplicable. But instead, they may contain an arbitrary number of free variables, corresponding to Monoidalness§ of Light Logic.

All these design concepts are realized in the following formal definition, which is written in a style inspired by [Abr93].

Definition 3.2 Let X, Y, Z range over the finite sets of variables. Then the 4-ary relation $t \in \mathcal{T}_{X,Y,Z}$ (saying that t is a (well-formed) term with undischarged variables X, !-discharged variables Y and §-discharged variables Z) is defined as follows (in writing $t \in \mathcal{T}_{X,Y,Z}$, we implicitly assume that X, Y and Z are mutually disjoint):

1. $x \in \mathcal{T}_{X,Y,Z} \iff x \in X$.
2. $\lambda x.t \in \mathcal{T}_{X,Y,Z} \iff t \in \mathcal{T}_{X\cup\{x\},Y,Z},\ x \notin X,\ FO(x,t) \leq 1$.
3. $tu \in \mathcal{T}_{X,Y,Z} \iff t \in \mathcal{T}_{X,Y,Z},\ u \in \mathcal{T}_{X,Y,Z}$.
4. $!t \in \mathcal{T}_{X,Y,Z} \iff t \in \mathcal{T}_{Y,\emptyset,\emptyset},\ FO(t) \leq 1$.
5. $\S t \in \mathcal{T}_{X,Y,Z} \iff t \in \mathcal{T}_{Y\cup Z,\emptyset,\emptyset}$.
6. $\mathsf{let}\ t\ \mathsf{be}\ !x\ \mathsf{in}\ u \in \mathcal{T}_{X,Y,Z} \iff t \in \mathcal{T}_{X,Y,Z},\ u \in \mathcal{T}_{X,Y\cup\{x\},Z},\ x \notin Y$.
7. $\mathsf{let}\ t\ \mathsf{be}\ \S x\ \mathsf{in}\ u \in \mathcal{T}_{X,Y,Z} \iff t \in \mathcal{T}_{X,Y,Z},\ u \in \mathcal{T}_{X,Y,Z\cup\{x\}},\ x \notin Z,\ FO(x,u) \leq 1$.

Finally, t is a *(well-formed) term* ($t \in \mathcal{T}$) if $t \in \mathcal{T}_{X,Y,Z}$ for some X, Y and Z.

Definition 3.3

1. For each natural number n, we have *Church numeral* $\overline{n} \in \mathcal{T}$ defined by

$$\overline{n} \equiv \lambda x.(\mathsf{let}\ x\ \mathsf{be}\ !z\ \mathsf{in}\ \S\lambda y.\underbrace{(z \cdots (z}_{n\ times}\,y) \cdots)).$$

2. For each word $w \equiv i_0 \cdots i_n \in \{0,1\}^*$, we have $\overline{w} \in \mathcal{T}$ defined by

$$\overline{w} \equiv \lambda x_0 x_1.(\mathsf{let}\ x_0\ \mathsf{be}\ !z_0\ \mathsf{in}\ (\mathsf{let}\ x_1\ \mathsf{be}\ !z_1\ \mathsf{in}\ \S\lambda y.(z_{i_0} \cdots (z_{i_n} y) \cdots)).$$

   Observe that these $\overline{n}$'s and $\overline{w}$'s are all of depth 1.

Here are further examples:

Example 3.4

1. $\omega_{LA} \equiv \lambda x.(\mathsf{let}\ x\ \mathsf{be}\ !y\ \mathsf{in}\ \S yy) \in \mathcal{T}$.
2. $\Omega_{LA} \equiv \omega_{LA} ! \omega_{LA} \in \mathcal{T}$.
3. $Suc \equiv \lambda yx.\mathsf{let}\ x\ \mathsf{be}\ !x'\ \mathsf{in}\ (\mathsf{let}\ y!x'\ \mathsf{be}\ \S y'\ \mathsf{in}\ \S(\lambda z.x'(y'z))) \in \mathcal{T}$.

We have the following basic properties:

Lemma 3.5 *Let $t \in \mathcal{T}_{X,Y,Z}$.*

1. *If $X \subseteq X'$, $Y \subseteq Y'$ and $Z \subseteq Z'$, then $t \in \mathcal{T}_{X',Y',Z'}$.*
2. *If $x \notin FV(t)$, then $t \in \mathcal{T}_{X\setminus\{x\},Y\setminus\{x\},Z\setminus\{x\}}$.*
3. *Let $x \in FV(t)$. Then x occurs at depth 0 iff $x \in X$. x occurs at depth 1 iff $x \in Y \cup Z$. x never occurs at depth > 1.*

Proof. By induction on t. ■

Lemma 3.6 (Substitution)

1. *$t \in \mathcal{T}_{X\cup\{x\},Y,Z}$, $x \notin X$ and $u \in \mathcal{T}_{X,Y,Z} \Longrightarrow t[u/x] \in \mathcal{T}_{X,Y,Z}$.*
2. *$t \in \mathcal{T}_{X,Y\cup\{x\},Z}$, $x \notin Y$ $u \in \mathcal{T}_{Y,\emptyset,\emptyset}$ and $FO(u) \leq 1 \Longrightarrow t[u/x] \in \mathcal{T}_{X,Y,Z}$.*
3. *$t \in \mathcal{T}_{X,Y,Z\cup\{x\}}$, $x \notin Z$ and $u \in \mathcal{T}_{Y\cup Z,\emptyset,\emptyset} \Longrightarrow t[u/x] \in \mathcal{T}_{X,Y,Z}$.*

Proof. By induction on t. ■

Remark 3.7 As discussed in Section 3 of [Asp98], a naive use of a box notation causes ambiguity, and in conjunction with naive substitutions, it causes a disastrous effect on complexity.

Asperti fixed this problem by using a more sophisticated box notation like $\S(t)[u_1/x_1, \dots, u_n/x_n]$, while our solution is more implicit and is based on the conceptual distinction between discharged and undischarged variables.

Asperti's box $\S(tx_1x_2)[y/x_1, y/x_2]$ (with y of !-type) corresponds to ($\mathsf{let}\ y\ \mathsf{be}\ !x\ \mathsf{in}\ \S(txx)$) in our syntax. Observe that variable y, which is external to the §-box, is shared in the former, while variable x, which is internal to the §-box, is shared in the latter. This is parallel to the difference between the contraction inference rule of Asperti's **ILAL** and that of Girard's original formation of **LLL**; the former contracts !-formulas, while the latter contracts discharged formulas.

Remark 3.8 There is a quadratic time algorithm checking whether a given pseudo-term is well-formed: Let t be a pseudo-term, and X and Y be the sets of its free variables at depth 0 and at depth 1, respectively. Then t is well-formed iff $t \in \mathcal{T}_{X,Y,\emptyset}$ (by Lemma 3.5 and the fact that $t \in \mathcal{T}_{X,Y,Z}$ implies $t \in \mathcal{T}_{X,Y\cup Z,\emptyset}$). The latter can be recursively checked with at most $|t|$ recursive calls, and each call involves a variable occurrence check at most once (corresponding to Clauses 2, 4 and 7 of Definition 3.2). Thus the algorithm runs in time $O(n^2)$, given a term of size n.

Name	*Redex*	*Contractum*
(β)	$(\lambda x.t)u$	$t[u/x]$
$(\S)$	$\mathsf{let}\ \S u\ \mathsf{be}\ \S x\ \mathsf{in}\ t$	$t[u/x]$
$(!)$	$\mathsf{let}\ !u\ \mathsf{be}\ !x\ \mathsf{in}\ t$	$t[u/x]$
(com)	$(\mathsf{let}\ u\ \mathsf{be}\ \dagger x\ \mathsf{in}\ t)v$	$\mathsf{let}\ u\ \mathsf{be}\ \dagger x\ \mathsf{in}\ (tv)$
	$\mathsf{let}\ (\mathsf{let}\ u\ \mathsf{be}\ \dagger x\ \mathsf{in}\ t)\ \mathsf{be}\ \dagger y\ \mathsf{in}\ v$	$\mathsf{let}\ u\ \mathsf{be}\ \dagger x\ \mathsf{in}\ (\mathsf{let}\ t\ \mathsf{be}\ \dagger y\ \mathsf{in}\ v)$

Figure 3.2: Reduction Rules

3.1.3 Reduction

Definition 3.9 The *reduction rules* of λLA are those listed in Figure 3.2. We say that t *reduces to* u *at address* w *by rule* (r), and write as $t \xrightarrow{w,(r)} u$, if $t \equiv \Phi[v_1]$, $u \equiv \Phi[v_2]$, the hole $\bullet$ is located at w in Φ, and v_1 is an (r)-redex whose contractum is v_2.

Note that the address w uniquely determines the rule (r) to be used. When either the address w or the rule (r), or both, are irrelevant, we use notations $t \xrightarrow{(r)} u$, $t \xrightarrow{w} u$ and $t \longrightarrow u$. The *depth* of a reduction is the depth of its redex.

A finite sequence σ of addresses $w_0, \dots, w_{n-1}$ is said to be a *reduction sequence* from t_0 to t_n, written as $t_0 \xrightarrow{\sigma}{}^{*} t_n$, if there are pseudo-terms $t_0, \dots, t_n$ such that

$$t_0 \xrightarrow{w_0} t_1 \xrightarrow{w_1} \cdots \xrightarrow{w_{n-1}} t_n.$$

If every reduction in σ is the application of (r), then σ is called an *(r)-reduction sequence* and written as $t_0 \xrightarrow{\sigma,(r)}{}^{*} t_n$ (or simply as $t_0 \xrightarrow{(r)}{}^{*} t_n$). The length of σ is denoted by $|\sigma|$.

Remark 3.10 The stratified structure of a term is preserved by reduction. In particular, the depth of a term never increases, since in reduction rules (β), $(\S)$ and $(!)$ a subterm u is substituted for a variable x occurring at the same depth and (com) does not affect the stratified structure.

Reduction rules (β) and $(\S)$ are strictly size-decreasing, since they never involve duplication. (com) preserves a term's size. The only reduction rule which causes duplication is $(!)$. Observe, however, that it is also strictly size-decreasing at the depth of the redex. The size increases only at deeper layers.

The terms are closed under reduction:

Proposition 3.11 *If* $t \in \mathcal{T}_{X,Y,Z}$ *and* $t \longrightarrow u$*, then* $u \in \mathcal{T}_{X,Y,Z}$.

(1)

$$\begin{aligned}\Omega_{LA} \equiv \omega_{LA}!\omega_{LA} &\xrightarrow{(\beta)} (\mathsf{let}\ !\omega_{LA}\ \mathsf{be}\ !y\ \mathsf{in}\ \S yy)\\ &\xrightarrow{(!)} \S\omega_{LA}\omega_{LA}\\ &\xrightarrow{(\beta)} \S(\mathsf{let}\ \omega_{LA}\ \mathsf{be}\ !y\ \mathsf{in}\ \S yy).\end{aligned}$$

(2)

$$\begin{aligned}Suc\ \overline{2} &\xrightarrow{(\beta)} \lambda x.\mathsf{let}\ x\ \mathsf{be}\ !x'\ \mathsf{in}\ (\mathsf{let}\ (\lambda x.(\mathsf{let}\ x\ \mathsf{be}\ !z\ \mathsf{in}\ \S\lambda y.(z(zy))))!x'\ \mathsf{be}\ \S y'\ \mathsf{in}\ \S(\lambda z.x'(y'z)))\\ &\xrightarrow{(\beta)} \lambda x.\mathsf{let}\ x\ \mathsf{be}\ !x'\ \mathsf{in}\ (\mathsf{let}\ (\mathsf{let}\ !x'\ \mathsf{be}\ !z\ \mathsf{in}\ \S\lambda y.(z(zy)))\ \mathsf{be}\ \S y'\ \mathsf{in}\ \S(\lambda z.x'(y'z)))\\ &\xrightarrow{(!)} \lambda x.\mathsf{let}\ x\ \mathsf{be}\ !x'\ \mathsf{in}\ (\mathsf{let}\ \S\lambda y.(x'(x'y))\ \mathsf{be}\ \S y'\ \mathsf{in}\ \S(\lambda z.x'(y'z)))\\ &\xrightarrow{(\S)} \lambda x.\mathsf{let}\ x\ \mathsf{be}\ !x'\ \mathsf{in}\ \S(\lambda z.x'(\lambda y.(x'(x'y))z))\\ &\xrightarrow{(\beta)} \lambda x.\mathsf{let}\ x\ \mathsf{be}\ !x'\ \mathsf{in}\ \S(\lambda z.x'(x'(x'z))) \equiv \overline{3}\end{aligned}$$

Figure 3.3: Examples of normalization

Proof. By induction on Φ, we prove the following: if $\Phi[t] \in \mathcal{T}_{X,Y,Z}$, $\Phi[t] \to \Phi[u]$ and t is the redex of the reduction, then $\Phi[u] \in \mathcal{T}_{X,Y,Z}$ and

(*) $FO(x, \Phi[u]) \leq FO(x, \Phi[t])$ for each $x \in X \cup Z$.

If $\Phi \equiv \bullet$ then $u \in \mathcal{T}_{X,Y,Z}$ is easily checked by inspecting each reduction rule. For example, if t is a (!) redex $\mathsf{let}\ !v_1\ \mathsf{be}\ !x\ \mathsf{in}\ v_2$, then $v_2 \in \mathcal{T}_{X,Y\cup\{x\},Z}$, $v_1 \in \mathcal{T}_{Y,\emptyset,\emptyset}$ and $FO(v_1) \leq 1$. Hence $u \equiv v_2[v_1/x] \in \mathcal{T}_{X,Y,Z}$ by Lemma 3.6. Property (*) is easily examined.

If $\Phi \equiv \lambda y.\Phi'$, then $\Phi'[t] \in \mathcal{T}_{X\cup\{y\},Y,Z}$, $y \notin X$ and $FO(y, \Phi'[t]) \leq 1$. By the induction hypothesis, $\Phi'[u] \in \mathcal{T}_{X\cup\{y\},Y,Z}$, $y \notin X$ and $FO(y, \Phi'[u]) \leq 1$. Therefore $\lambda y.\Phi'[u] \in \mathcal{T}_{X,Y,Z}$. Property (*) is obvious.

The other cases are similar. ■

Example 3.12

1. The term Ω_{LA} in Example 3.4 is a light affine analogue of $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$, which is not normalizable in λ-calculus. However, Ω_{LA} is normalizable; see Figure 3.3 (1).
2. The term *Suc* in Example 3.4 represents successor for numerals $\overline{n}$. For example the normalization of *Suc* $\overline{2}$ is illustrated in Figure 3.3 (2).

3.2 Proving the Polystep Strong Normalization Theorem

In this section we shall prove the main part of the proof of the polystep (and polytime) strong normalization theorem for λLA. The theorem itself will appear in the next section. The key step toward the polystep strong normalization theorem is a sort of *standardization*, i.e., transformation of a reduction sequence into a outer-layer-first one. However, standardization in λLA may undesirably shorten the length of a reduction sequence. To avoid shortening, we first need to extend λLA with *explicit* weakening and to give a translation of reduction sequences in λLA into this extended calculus (in 3.2.1). Then we show the standardization theorem for the extended calculus (in 3.2.2), where shortening of reduction sequences is successfully avoided. Finally, we show that the length of a standard reduction sequence thus obtained is polynomially bounded (in 3.2.3).

3.2.1 An Extended Calculus with Explicit Weakening

The set $\mathcal{PT}^w$ of *extended pseudo-terms* is defined analogously to $\mathcal{PT}$, but each extended pseudo-term may contain a subexpression of the form $\mathsf{let}\ t\ \mathsf{be}\ \_\ \mathsf{in}\ u$ (explicit weakening). To define the well-formedness, we give a new 4-ary relation $t \in \mathcal{T}^w_{X,Y,Z}$ by modifying Definition 3.2 as follows.

(1) Replace clauses 2, 6, and 7 with:

2' $\lambda x.t \in \mathcal{T}_{X,Y,Z} \iff t \in \mathcal{T}_{X\cup\{x\},Y,Z},\ x \notin X,\ FO(x,t) = 1.$

6' $\mathsf{let}\ t\ \mathsf{be}\ !x\ \mathsf{in}\ u \in \mathcal{T}_{X,Y,Z} \iff t \in \mathcal{T}_{X,Y,Z},\ u \in \mathcal{T}_{X,Y\cup\{x\},Z},\ x \notin Y, FO(x,u) \geq 1.$

7' $\mathsf{let}\ t\ \mathsf{be}\ \S x\ \mathsf{in}\ u \in \mathcal{T}_{X,Y,Z} \iff t \in \mathcal{T}_{X,Y,Z},\ u \in \mathcal{T}_{X,Y,Z\cup\{x\}},\ x \notin Z,\ FO(x,u) = 1.$

Namely, we require that each binder must bind at least one (and exactly one, in case of λ and let-§) variable occurrence.

(2) Add the following clause:

8' $\mathsf{let}\ t\ \mathsf{be}\ \_\ \mathsf{in}\ u \in \mathcal{T}_{X,Y,Z} \iff t \in \mathcal{T}_{X,Y,Z},\ u \in \mathcal{T}_{X,Y,Z}.$

We say that t is a *(well-formed) extended term* ($t \in \mathcal{T}^w$) if $t \in \mathcal{T}^w_{X,Y,Z}$ for some X, Y, Z.

The reduction rules in Figure 3.2 are extended to $\mathcal{PT}^w$ with the following modifications:

- Generalize (*com*) so that it is also applicable to the new let operator for explicit weakening.
- Add a new reduction rule (_):
  $\mathsf{let}\ u\ \mathsf{be}\ \_\ \mathsf{in}\ t \longrightarrow t.$

Reduction rules other than (_) are called *proper*. A reduction sequence is *proper* if every reduction in it is proper.

Lemmas 3.5 and 3.6 hold for $\mathcal{T}^w$, too. In addition, we have:

Proposition 3.13 *If $t \in \mathcal{T}^w_{X,Y,Z}$, $t \xrightarrow{(r)} u$ and (r) is proper, then $u \in \mathcal{T}^w_{X,Y,Z}$.*

Now we consider a translation of λLA into the extended calculus.

Lemma 3.14 *For each term t, there is an extended term t^w such that $t^w \xrightarrow{(\_)}{}^{*} t$ and $|t^w| \leq 4|t|$.*

Proof. By induction on t. If $t \equiv \lambda x.u$ and $FO(x,u) = 0$, let $t^w \equiv \lambda x.(\mathsf{let}\ x\ \mathsf{be}\ \_\ \mathsf{in}\ u^w)$. If $t \equiv (\mathsf{let}\ v\ \mathsf{be}\ \dagger x\ \mathsf{in}\ u)$ and $FO(x,u) = 0$, let $t^w \equiv \mathsf{let}\ v\ \mathsf{be}\ \dagger x\ \mathsf{in}\ (\mathsf{let}\ \S x\ \mathsf{be}\ \_\ \mathsf{in}\ u^w)$. ■

Theorem 3.15 (Translation into the extended calculus) *Let t_0 be a term and let*

$$t_0 \xrightarrow{\sigma}{}^{*} t_1$$

be a reduction sequence in λLA. *Then there are extended terms t_0', t_1' and a proper reduction sequence τ such that $|\sigma| \leq |\tau|$, $|t_0'| \leq 4|t_0|$ and*

$$\begin{tikzcd} t_0 \arrow[r, "\sigma *"] & t_1 \\ t_0' \arrow[u, "* (\_)"] \arrow[r, "\tau *"] & t_1'. \arrow[u, "* (\_)"'] \end{tikzcd}$$

The proof idea is as follows. By Lemma 3.14, there is an extended term t_0^w such that

$$t_0^w \xrightarrow{(\_)}{}^{*} t_0 \xrightarrow{\sigma}{}^{*} t_1.$$

By permuting it suitably, we can obtain

$$t_0^w \xrightarrow{\tau}{}^{*} t_1' \xrightarrow{(\_)}{}^{*} t_1,$$

such that τ is proper and $|\tau| \geq |\sigma|$. For example, a reduction sequence of the form

$$(\mathsf{let}\ v\ \mathsf{be}\ \_\ \mathsf{in}\ (\lambda x.t))u \xrightarrow{(\_)} (\lambda x.t)u \xrightarrow{(\beta)} t[u/x]$$

can be transformed into the following longer one:

$$\begin{aligned} (\mathsf{let}\ v\ \mathsf{be}\ \_\ \mathsf{in}\ (\lambda x.t))u &\xrightarrow{(com)} \mathsf{let}\ v\ \mathsf{be}\ \_\ \mathsf{in}\ ((\lambda x.t)u) \\ &\xrightarrow{(\beta)} \mathsf{let}\ v\ \mathsf{be}\ \_\ \mathsf{in}\ t[u/x] \xrightarrow{(\_)} t[u/x]. \end{aligned}$$

For a precise proof, we need the following two lemmas.

Lemma 3.16 *Let $t_0 \in \mathcal{PT}^w$. If $t_0 \xrightarrow{(\_)} t_1 \xrightarrow{(com)} t_2$, then*

$$t_0 \xrightarrow{\sigma,(com)}{}^{*} t_1' \xrightarrow{(\_)} t_2$$

for some t_1' and $|\sigma| \geq 1$.

Proof. Let w_0 be the address of the contractum of the (_) reduction and w_1 be the address of the redex of the (com) reduction in t_1. We consider the following cases.

(Case 1) w_0 and w_1 are incomparable. Then the reduction sequence must be of the form:

$$\Phi[u_0, u_1] \xrightarrow{(\_)} \Phi[u_0', u_1] \xrightarrow{(com)} \Phi[u_0', u_1'],$$

where Φ is a context with two holes, the first one at w_0 and the second one at w_1. In this case, it can be permuted into:

$$\Phi[u_0, u_1] \xrightarrow{(r)} \Phi[u_0, u_1'] \xrightarrow{(\_)} \Phi[u_0', u_1'].$$

(Case 2) $w_0 \sqsubseteq w_1$. This means that the (*com*) redex is embedded in the (_) contractum, namely the reduction sequence must be of the form:

$$\Phi_0[\mathsf{let}\ t\ \mathsf{be}\ \_\ \mathsf{in}\ \Phi_1[u]] \xrightarrow{(\_)} \Phi_0[\Phi_1[u]] \xrightarrow{(com)} \Phi_0[\Phi_1[u']].$$

Then it can be permuted into:

$$\Phi_0[\mathsf{let}\ t\ \mathsf{be}\ \_\ \mathsf{in}\ \Phi_1[u]] \xrightarrow{(com)} \Phi_0[\mathsf{let}\ t\ \mathsf{be}\ \_\ \mathsf{in}\ \Phi_1[u']] \xrightarrow{(\_)} \Phi_0[\Phi_1[u']].$$

(Case 3) $w_0 = w_1 0$. This means that two reductions overlap. We apply the permutation depicted in Figure 3.4(a), which indicates that the reduction sequence in solid lines may be replaced with one in broken lines.

(Case 4) $w_0 \sqsupseteq w_1 00$ or $w_0 \sqsupseteq w_1 1$. This means that the (_) contractum is embedded in the (*com*) redex. In this case we can first perform the (*com*) reduction, then the (_) reduction. Since the (*com*) reduction neither duplicate nor erase the (_) redex, the latter (_) reduction takes place exactly once. ■

Lemma 3.17 *Let $t_0 \in \mathcal{PT}^w$. If $t_0 \xrightarrow{(\_)} t_1 \xrightarrow{(r)} t_2$, where (r) is neither (com) nor $(\_)$, then*

$$t_0 \xrightarrow{(com)}{}^{*} t_1' \xrightarrow{(r)} t_1'' \xrightarrow{(\_)}{}^{*} t_2$$

for some t_1' and t_1''.

Proof. Let w_0 be the address of the contractum of the (_) reduction and w_1 be the address of the redex of the (r) reduction. We consider the following cases.

(Case 1) w_0 and w_1 are incomparable. Similar to (Case 1) of the previous lemma. (Case 2) $w_0 \sqsubseteq w_1$. Similar to (Case 2) of the previous lemma. (Case 3) $w_0 = w_1 0$. This means that two reductions overlap. We apply the permutations depicted in Figure 3.4 (b) and (c).

For the rest, we only treat the case where (r) is (!).
(Case 4) $w_0 \sqsupseteq w_1 00$. We have:

$$\Phi[\mathsf{let}\ !\Phi_1[\mathsf{let}\ t\ \mathsf{be}\ \_\ \mathsf{in}\ u]\ \mathsf{be}\ !x\ \mathsf{in}\ v] \xrightarrow{(\_)} \Phi[\mathsf{let}\ !\Phi_1[u]\ \mathsf{be}\ !x\ \mathsf{in}\ v] \xrightarrow{(!)} \Phi[v[\Phi_1[u]/x]],$$

which can be transformed into:

$$\Phi[\mathsf{let}\ !\Phi_1[\mathsf{let}\ t\ \mathsf{be}\ \_\ \mathsf{in}\ u]\ \mathsf{be}\ !x\ \mathsf{in}\ v] \xrightarrow{(!)} \Phi[v[\Phi_1[\mathsf{let}\ t\ \mathsf{be}\ \_\ \mathsf{in}\ u]/x]] \xrightarrow{\sigma,(\_)}{}^{*} \Phi[v[\Phi_1[u]/x]].$$

(Case 5) $w_0 \sqsupseteq w_1 1$. We have:

$$\Phi[\mathsf{let}\ !u\ \mathsf{be}\ !x\ \mathsf{in}\ \Phi_1[\mathsf{let}\ t\ \mathsf{be}\ \_\ \mathsf{in}\ v]] \xrightarrow{(\_)} \Phi[\mathsf{let}\ !u\ \mathsf{be}\ !x\ \mathsf{in}\ \Phi_1[v]] \xrightarrow{(!)} \Phi[\Phi_1[v][u/x]],$$

which can be transformed into:

$$\Phi[\mathsf{let}\ !u\ \mathsf{be}\ !x\ \mathsf{in}\ \Phi_1[\mathsf{let}\ t\ \mathsf{be}\ \_\ \mathsf{in}\ v]] \xrightarrow{(!)} \Phi[\Phi_1[\mathsf{let}\ t\ \mathsf{be}\ \_\ \mathsf{in}\ v][u/x]] \xrightarrow{(\_)} \Phi[\Phi_1[v][u/x]].$$

(a)

$$\begin{tikzcd}
\Psi[\mathsf{let}\ u_1\ \mathsf{be}\ \_\ \mathsf{in}\ (\mathsf{let}\ u_2\ \mathsf{be}\ *\ \mathsf{in}\ t)] \arrow[r, "(\_)"] \arrow[d, dotted, "(com)"] & \Psi[\mathsf{let}\ u_2\ \mathsf{be}\ *\ \mathsf{in}\ t] \arrow[r, "(com)"] & \mathsf{let}\ u_2\ \mathsf{be}\ *\ \mathsf{in}\ \Psi[t] \\
\mathsf{let}\ u_1\ \mathsf{be}\ \_\ \mathsf{in}\ \Psi[\mathsf{let}\ u_2\ \mathsf{be}\ *\ \mathsf{in}\ t] \arrow[rr, dotted, "(com)"] & & \mathsf{let}\ u_1\ \mathsf{be}\ \_\ \mathsf{in}\ \mathsf{let}\ u_2\ \mathsf{be}\ *\ \mathsf{in}\ \Psi[t] \arrow[u, dotted, "(\_)"']
\end{tikzcd}$$

where Ψ is of the form $(\bullet)v$ or $\mathsf{let}\ \bullet\ \mathsf{be}\ *\ \mathsf{in}\ v$.

(b)

$$\begin{tikzcd}
(\mathsf{let}\ t\ \mathsf{be}\ \_\ \mathsf{in}\ (\lambda y.u))v \arrow[r, "(\_)"] \arrow[d, dotted, "(com)"] & (\lambda y.u)v \arrow[r, "(\beta)"] & u[v/y] \\
\mathsf{let}\ t\ \mathsf{be}\ \_\ \mathsf{in}\ ((\lambda y.u)v) \arrow[rr, dotted, "(\beta)"] & & \mathsf{let}\ t\ \mathsf{be}\ \_\ \mathsf{in}\ (u[v/y]) \arrow[u, dotted, "(\_)"']
\end{tikzcd}$$

(c)

$$\begin{tikzcd}
\mathsf{let}\ (\mathsf{let}\ t\ \mathsf{be}\ \_\ \mathsf{in}\ \dagger u)\ \mathsf{be}\ \dagger y\ \mathsf{in}\ v \arrow[r, "(\_)"] \arrow[d, dotted, "(com)"] & \mathsf{let}\ \dagger u\ \mathsf{be}\ \dagger y\ \mathsf{in}\ v \arrow[r, "(\dagger)"] & v[u/x] \\
\mathsf{let}\ t\ \mathsf{be}\ \_\ \mathsf{in}\ (\mathsf{let}\ \dagger u\ \mathsf{be}\ \dagger y\ \mathsf{in}\ v) \arrow[rr, dotted, "(\dagger)"] & & \mathsf{let}\ t\ \mathsf{be}\ \_\ \mathsf{in}\ (v[u/x]) \arrow[u, dotted, "(\_)"']
\end{tikzcd}$$

(d)

$$\begin{tikzcd}
(\mathsf{let}\ !t\ \mathsf{be}\ !x\ \mathsf{in}\ (\lambda y.u))v \arrow[r, "(!)"] \arrow[d, dotted, "(com)"] & (\lambda y.u[t/x])v \arrow[r, "(\beta)"] & u[t/x][v/y] \arrow[d, dash, "\equiv"] \\
\mathsf{let}\ !t\ \mathsf{be}\ !x\ \mathsf{in}\ (\lambda y.u)v \arrow[r, dotted, "(\beta)"] & \mathsf{let}\ !t\ \mathsf{be}\ !x\ \mathsf{in}\ u[v/y] \arrow[r, dotted, "(!)"] & u[v/y][t/x]
\end{tikzcd}$$

Figure 3.4: Permutation Rules

■

Proof of Theorem 3.15.
We argue step by step as follows:
(1) Every reduction sequence of the form

$$t_0 \xrightarrow{(\_)} t_1 \xrightarrow{\nu,(com)}{}^{*} t_2$$

may be transformed into

$$t_0 \xrightarrow{\nu',(com)}{}^{*} t_1' \xrightarrow{(\_)} t_2.$$

where $|\nu| \leq |\nu'|$. This is proved by induction on $|\nu|$, using Lemma 3.16.
(2) Every reduction sequence of the form

$$t_0 \xrightarrow{\tau,(\_)}{}^{*} t_1 \xrightarrow{\nu,(com)}{}^{*} t_2$$

may be transformed into

$$t_0 \xrightarrow{\nu',(com)}{}^{*} t_1' \xrightarrow{\tau',(\_)}{}^{*} t_2,$$

where $|\tau| = |\tau'|$ and $|\nu| \leq |\nu'|$. This is proved by induction on $|\tau|$, using (1).
(3) Every reduction sequence of the form

$$t_0 \xrightarrow{\tau,(\_)}{}^{*} t_1 \xrightarrow{(r)} t_2,$$

where (r) is proper, may be transformed into

$$t_0 \xrightarrow{\nu'}{}^{*} t_1' \xrightarrow{\tau',(\_)}{}^{*} t_2,$$

where ν' is proper and $|\nu'| \geq 1$. This is proved by induction on $|\tau|$, using Lemma 3.17 and (2).
(4) Every reduction sequence of the form

$$t_0 \xrightarrow{\tau,(\_)}{}^{*} t_1 \xrightarrow{\nu}{}^{*} t_2,$$

where ν is proper, may be transformed into

$$t_0 \xrightarrow{\nu'}{}^{*} t_1' \xrightarrow{\tau',(\_)}{}^{*} t_2,$$

where ν' is proper and $|\nu| \leq |\nu'|$. This is proved by induction on $|\nu|$, using (3).
(5) Finally, given a term t_0 and a reduction sequence $t_0 \xrightarrow{\sigma}{}^{*} t_1$, we apply Lemma 3.14 to obtain a suitable extended term t_0'. Then the theorem immediately follows from (4). ■

3.2.2 Standardization Theorem

A reduction sequence σ is *standard* if it can be partitioned into subsequences $\sigma_0; \sigma_1; \ldots; \sigma_{2d}$, such that, for $i \leq d$, σ_{2i+1} consists of (!)-reductions at depth i and σ_{2i} consists of other reductions at depth i.

Theorem 3.18 (Standardization) *Let t_0 be an extended term and σ be a proper reduction sequence*

$$t_0 \xrightarrow{\sigma}{}^{*} t_1.$$

Then there is a standard proper reduction sequence τ

$$t_0 \xrightarrow{\tau}{}^{*} t_1$$

such that $|\sigma| \leq |\tau|$.

The proof is again based on permutation of reduction sequences. For example, let u be a (β) redex and u' be its contractum, and consider the following nonstandard reduction sequence:

$$\mathsf{let}\ !u\ \mathsf{be}\ !x\ \mathsf{in}\ v \xrightarrow{(\beta)} \mathsf{let}\ !u'\ \mathsf{be}\ !x\ \mathsf{in}\ v \xrightarrow{(!)} v[u'/x].$$

Here the first reduction is at depth 1 and the second at depth 0. It can be standardized as follows:

$$\mathsf{let}\ !u\ \mathsf{be}\ !x\ \mathsf{in}\ v \xrightarrow{(!)} v[u/x] \xrightarrow{\sigma,(\beta)}{}^{*} v[u'/x].$$

Since ($\mathsf{let}\ !u\ \mathsf{be}\ !x\ \mathsf{in}\ v$) is an extended term, we have $FO(x, v) \geq 1$. Hence $v[u/x]$ contains at least one occurrence of the (β) redex u, so $|\sigma| \geq 1$. Therefore the length of a reduction sequence never decreases by this permutation. For a precise proof, we need the following two lemmas.

Lemma 3.19 *Let*

$$t_0 \xrightarrow{w_0,(r_0)} t_1 \xrightarrow{w_1,(r_1)} t_2,$$

and suppose that the first reduction is at deeper depth than the second. Then

$$t_0 \xrightarrow{(r_1)} t_1' \xrightarrow{(r_0),\sigma}{}^{*} t_2,$$

such that the first reduction is at depth $dp(w_1, t_1)$ and all reductions in σ are at depth $dp(w_0, t_0)$.

Proof. The case when w_0 and w_1 are incomparable is similar to (Case 1) of the proof of Lemma 3.17. It is impossible to have $w_0 \sqsubseteq w_1$, since $dp(w_0, t_1) > dp(w_1, t_1)$. It is also impossible that two reductions overlap. For the other cases, we only consider the critical case when w_1 points at a (!)-redex.

If $w_0 \sqsupseteq w_1 0$, we have

$$\Phi_0[\mathsf{let}\ !\Phi_1[u]\ \mathsf{be}\ !x\ \mathsf{in}\ v] \xrightarrow{w_0,(r)} \Phi_0[\mathsf{let}\ !\Phi_1[u']\ \mathsf{be}\ !x\ \mathsf{in}\ v] \xrightarrow{w_1,(!)} \Phi_0[v[\Phi_1[u']/x]],$$

which can be transformed into:

$$\Phi_0[\mathsf{let}\ !\Phi_1[u]\ \mathsf{be}\ !x\ \mathsf{in}\ v] \xrightarrow{w_1,(!)} \Phi_0[v[\Phi_1[u]/x]] \xrightarrow{\sigma,(r)}{}^{*} \Phi_0[v[\Phi_1[u']/x]].$$

Note that $FO(x, v) \geq 1$ by the definition of the extended terms, therefore we always have $|\sigma| \geq 1$.

If $w_0 \sqsupseteq w_1 1$, we have

$$\Phi_0[\mathsf{let}\ !u\ \mathsf{be}\ !x\ \mathsf{in}\ \Phi_1[v]] \xrightarrow{w_0,(r)} \Phi_0[\mathsf{let}\ !u\ \mathsf{be}\ !x\ \mathsf{in}\ \Phi_1[v']] \xrightarrow{w_1,(!)} \Phi_0[\Phi_1[v'][u/x]],$$

which can be transformed into:

$$\Phi_0[\mathsf{let}\ !u\ \mathsf{be}\ !x\ \mathsf{in}\ \Phi_1[v]] \xrightarrow{w_1,(!)} \Phi_0[\Phi_1[v][u/x]] \xrightarrow{w_0',(r)} \Phi_0[\Phi_1[v'][u/x]].$$

■

Lemma 3.20 *Let t_0 be an extended term and let a proper reduction sequence*

$$t_0 \xrightarrow{w_0,(!)} t_1 \xrightarrow{w_1,(r)} t_2,$$

be given, where (r) is not $(!)$ and two reductions are at the same depth d. Then we have a reduction sequence

$$t_0 \xrightarrow{(com)}{}^{*} t_1' \xrightarrow{w_1',(r)} t_1'' \xrightarrow{w_0',(!)} t_2,$$

such that every reduction in it is at depth d.

$$\boxed{\begin{array}{rcl@{\qquad}rcl}
s_0(x) & = & 1 & s_i(x) & = & 0 \\
s_0(\lambda x.t) & = & s_0(t)+1 & s_i(\lambda x.t) & = & s_i(t) \\
s_0(tu) & = & s_0(t)+s_0(u)+1 & s_i(tu) & = & s_i(t)+s_i(u) \\
s_0(\dagger t) & = & FO(t)+1 & s_i(\dagger t) & = & s_{i-1}(t) \\
s_0(\mathsf{let}\ t\ \mathsf{be}\ \dagger x\ \mathsf{in}\ u) & = & s_0(t)+s_0(u)+1 & s_i(\mathsf{let}\ t\ \mathsf{be}\ \dagger x\ \mathsf{in}\ u) & = & s_i(t)+s_i(u) \\
s_0(\mathsf{let}\ t\ \mathsf{be}\ \_\ \mathsf{in}\ u) & = & s_0(t)+s_0(u)+1 & s_i(\mathsf{let}\ t\ \mathsf{be}\ \_\ \mathsf{in}\ u) & = & s_i(t)+s_i(u)
\end{array}}$$

Figure 3.5: Partial Sizes

Proof. The critical case is when (r) is (β) and two reductions overlap. In this case, we apply the permutation depicted in Figure 3.4(d). The equivalence of $u[t/x][v/y]$ and $u[v/y][t/x]$ is easily checked (recall that we are adopting the variable convention, so that x does not occur in v and y does not occur in t). ■

Proof of Theorem 3.18.
(1) Every proper reduction sequence $t_0 \longrightarrow^* u$ can be transformed into the following form without decreasing the length:

$$t_0 \xrightarrow{\tau_0}{}^* t_1 \xrightarrow{\tau_1}{}^* \cdots t_n \xrightarrow{\tau_n}{}^* u,$$

where τ_i consists of reductions at depth i $(0 \leq i \leq n)$. This is proved by a step-by-step argument similar to the proof of Theorem 3.15, using Lemma 3.19.
(2) Every reduction sequence $t_0 \xrightarrow{\tau}{}^* u$ which consists of reductions at the same depth can be transformed into

$$t_0 \xrightarrow{\tau_0}{}^* t_1 \xrightarrow{\tau_1}{}^* u,$$

where τ_1 consists of (!) reductions and τ_0 consists of other reductions. This is proved by induction on the number of (!) reductions in τ, using Lemma 3.20. ■

3.2.3 Bounding Lengths of Standard Reduction Sequences

For each extended term t its *partial size $s_i(t)$ at depth i* is defined in Figure 3.5 (where i ranges over the numbers ≥ 1). We define $s(t)$ to be $\sum_{i=0}^{\infty} s_i(t)$. The only difference between $|t|$ and $s(t)$ is that the size of a box $\dagger t$ in the latter sense also counts the number of free variable occurrences in t, i.e., that variables may be counted twice in the latter. Therefore it holds that $|t| \leq s(t) \leq 2|t|$.

The theorem below is essentially due to [Gir98, Asp98]. In our case, however, the length of a reduction sequence may slightly exceed the size of its final term, since we have the commuting reduction rule (*com*).

Theorem 3.21 (Polynomial bounds for standard reduction sequences) *Let t_0 be an extended term of depth d and σ be a standard proper reduction sequence $t_0 \xrightarrow{\sigma}{}^* u$. Then $s(u) \leq s(t_0)^{2^d}$ and $|\sigma| \leq s(t_0)^{2^{d+1}}$.*

This is shown with the help of the following lemmas.

Lemma 3.22 *Let σ be a reduction sequence $t \xrightarrow{\sigma}{}^* t'$ which consists of (!) reductions at depth i. Then $s_j(t') \leq s_j(t) \cdot s_i(t)$ for each $j > i$.*

Proof. For simplicity, let us assume $i = 0$. To estimate the potential size growth caused by (!) reductions, we make the following definition. For each extended term t, its *unfolding* $\sharp t \mathcal{PT}^w$ is defined as follows:

$$\begin{array}{rcl}
\sharp x & \equiv & x \\
\sharp(tu) & \equiv & \sharp t \sharp u \\
\sharp(\lambda x.t) & \equiv & \lambda x.\sharp t \\
\sharp(\dagger t) & \equiv & \dagger t \\
\sharp(\mathsf{let}\ !t\ \mathsf{be}\ !x\ \mathsf{in}\ u) & \equiv & \mathsf{let}\ \underbrace{!t!t\cdots!t}_{n\ times}\ \mathsf{be}\ !x\ \mathsf{in}\ \sharp u,\ \text{where } n = FO(x, \sharp u). \\
\sharp(\mathsf{let}\ t\ \mathsf{be}\ \dagger x\ \mathsf{in}\ u) & \equiv & \mathsf{let}\ \sharp t\ \mathsf{be}\ \dagger x\ \mathsf{in}\ \sharp u,\ \text{if } t \not\equiv !t' \text{ or } \dagger x \not\equiv !x.
\end{array}$$

We claim:

(1) $FO(\sharp v) \leq s_0(v)$.

(2) $s_j(v) \leq s_j(\sharp v) \leq s_0(v) \cdot s_j(v)$.

(3) if $v \xrightarrow{(!)} v'$ at depth 0. then $FO(x, \sharp v') \leq FO(x, \sharp v)$ for any x.

(4) if $v \xrightarrow{(!)} v'$ at depth 0, then $s_j(\sharp v') \leq s_j(\sharp v)$.

The lemma follows from (2) and (4):

$$s_j(t') \leq s_j(\sharp t') \leq s_j(\sharp t) \leq s_0(t) \cdot s_j(t).$$

Claim (1) is proved by induction on v. If $v \equiv \dagger u$, then $FO(\sharp v) = FO(\dagger u) \leq s_0(\dagger u)$. If $v \equiv$ $\mathsf{let}\ !u_1\ \mathsf{be}\ !x\ \mathsf{in}\ u_2$, then

$$FO(\sharp v) = FO(!u_1) \cdot FO(x, \sharp u_2) + FO(\sharp u_2) - FO(x, \sharp u_2) \leq FO(\sharp u_2) \leq s_0(u_2),$$

since $FO(!u_1) \leq 1$ by well-formedness. Other cases are easier.

The first half of Claim (2) is obvious. The second half is proved by induction on v, using (1). If $v \equiv \dagger u$, then $s_j(\sharp v) = s_j(\dagger u) \leq s_0(\dagger u) \cdot s_j(\dagger u)$. If $v \equiv \mathsf{let}\ !u_1\ \mathsf{be}\ !x\ \mathsf{in}\ u_2$, then

$$\begin{array}{rcl}
s_j(\sharp v) & = & s_j(!u_1) \cdot FO(x, \sharp u_2) + s_j(\sharp u_2) \\
& \leq & s_j(!u_1) \cdot s_0(u_2) + s_0(u_2) \cdot s_j(u_2) \\
& \leq & s_0(u_2) \cdot (s_j(!u_1) + s_j(u_2)) \\
& \leq & s_0(v) \cdot s_j(v)
\end{array}$$

Claims (3) is proved by induction on Φ, and (4) is also proved by induction on Φ, using Claim (3). ■

Lemma 3.23 *Let σ be a reduction sequence $t \xrightarrow{\sigma}{}^{*} t'$ which consists of reductions at depth i. Then $|\sigma| \leq s_i(t)^2$.*

Proof. For simplicity, assume that $i = 0$. For an extended term v and its subterm u at depth 0 of the form ($\mathsf{let}\ u_1\ \mathsf{be}\ *\ \mathsf{in}\ u_2$), we define

$$com(u, v) := s_0(v) - s_0(u_2).$$

Define $com(v)$ to be the sum of all $com(u, v)$'s with u ranging over all such occurrences of let-expressions in v. Then we claim:

(1) $s_0(v) + com(v) \leq s_0(v)^2$.

(2) If $v \xrightarrow{(r)} v'$ by a reduction at depth 0, then $s_0(v') + com(v') < s_0(v) + com(v)$.

The lemma follows from these two.

To show (1), observe that v contains at most $s_0(v) - 1$ let-expressions and that $com(u, v) \leq s_0(v)$ for each u. Hence

$$s_0(v) + com(v) \leq s_0(v) + (s_0(v) - 1) \cdot s_0(v) \leq s_0(v)^2.$$

Now let us show (2). In case that (r) is (β), $(\S)$ or $(!)$, $s_0(v)$ strictly decreases and $com(v)$ never increases. In case that (r) is (com), $s_0(v)$ does not change and $com(v)$ strictly decreases: let the reduction be

$$\Psi[\mathsf{let}\ u_1\ \mathsf{be}\ *\ \mathsf{in}\ u_2] \xrightarrow{(com)} \mathsf{let}\ u_1\ \mathsf{be}\ *\ \mathsf{in}\ \Psi[u_2],$$

where Ψ is of the form $(\bullet)v$ or $\mathsf{let}\ \bullet\ \mathsf{be}\ *\ \mathsf{in}\ v$. Then

$$com((\mathsf{let}\ u_1\ \mathsf{be}\ *\ \mathsf{in}\ \Psi[u_2]), v') < com((\mathsf{let}\ u_1\ \mathsf{be}\ *\ \mathsf{in}\ u_2), v),$$

while $com(u, v)$ remains unchanged for other u's. ■

Proof of Theorem 3.21. Let σ be partitioned as

$$t_0 \xrightarrow{\sigma_0}{}^* \xrightarrow{\sigma_1}{}^* t_1 \xrightarrow{\sigma_2}{}^* \xrightarrow{\sigma_3}{}^* \cdots t_d \xrightarrow{\sigma_{2d}}{}^* u.$$

Let $s_0 = s_0(t_0), \ldots, s_d = s_d(t_0)$. Let also $s = s_0 + \cdots s_d = s(t)$. By applying Lemma 3.22 repeatedly, we obtain the bounds for the partial sizes of t_i illustrated in the following table (taken from [Asp98]):

	depth 0	*depth* 1	*depth* 2	$\cdots$	*depth* d
t_0	s_0	s_1	s_2	$\cdots$	s_n
t_1	s_0	$s_0 s_1$	$s_0 s_2$	$\cdots$	$s_0 s_n$
t_2	s_0	$s_0 s_1$	$s_0^2 s_1 s_2$	$\cdots$	$s_0^2 s_1 s_n$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
s_d	s_0	$s_0 s_1$	$s_0^2 s_1 s_2$	$\cdots$	$s_0^{2^d} s_1^{2^{d-1}} \cdots s_{d-2}^2 s_{d-1} s_d$

Here, the polynomial at row i and column j bounds $s_i(t_j)$.

Therefore we see that $s_j(t_i)$ is bounded by $S_j = s_0^{2^j} s_1^{2^{j-1}} \cdots s_{j-1} s_j$, for each $j \leq d$ (regardless of i), and the final size s is bounded by $S_0 + S_1 + \cdots + S_d \leq s^{2^d}$.

On the other hand, $|\sigma_{2j}| + |\sigma_{2j+1}| \leq S_j^2$ for each $j \leq d$ by Lemmas 3.23. Therefore, $|\sigma|$ is bounded by

$$(S_0 + \cdots + S_d)^2 \leq s^{2^d \cdot 2} = s^{2^{d+1}}.$$

■

```
input $t$
loop
query to oracle $f$ to obtain $f(t)$
if $f(t)$ is defined
then let $t := t'$ such that $t \xrightarrow{f(t)} t'$
else output $t$ and halt
end loop.
```

Figure 3.6: Algorithm $\mathsf{normalize}_f$

3.3 Main Results

Now we are in a position to state the main results of this chapter. From Theorems 3.15, 3.18 and 3.21, it follows:

Theorem 3.24 (Polystep strong normalization) *For every term t_0 of size s and depth d, the following hold:*

1. *Every reduction sequence from t_0 has a length bounded by $O(s^{2^{d+1}})$.*
2. *Every term to which t_0 reduces has a size bounded by $O(s^{2^d})$.*

Corollary 3.25 (Church-Rosser property) *If t_0 is a term and $t_1 \longleftarrow^* t_0 \longrightarrow^* t_2$, then $t_1 \longrightarrow^* t_3 \longleftarrow^* t_2$ for some term t_3.*

Proof. By showing local confluence, which is straightforward. ■

To make precise what we mean by *polytime* strong normalization, we give the following definitions. A *reduction strategy for $\mathcal{T}$* is a partial function $f : \mathcal{T} \longrightarrow \{0,1\}^*$ such that $f(t)$ gives an address of a redex of t whenever t is reducible and is undefined otherwise. We can think of a Turing machine $\mathsf{normalize}_f$ with function oracle f, described in Figure 3.6.

Now we have:

Corollary 3.26 (Polytime strong normalization) *For any reduction strategy f for $\mathcal{T}$, $\mathsf{normalize}_f$ terminates in time $O(s^{2^{d+2}})$, given a term t_0 of size s and depth d as input. It outputs the unique normal form of t_0.*

Proof. Observe that each step of reduction $t \longrightarrow t'$ is carried out in quadratic time; the worst case, namely the case of (!)-reduction, consists in substituting a subterm of size $\leq |t|$ for at most $|t|$ variable occurrences. Therefore the total runtime is roughly estimated by $O(s^{2^d \cdot 2} \cdot s^{2^{d+1}}) = O(s^{2^{d+2}})$. ■

Chapter 4

Proofs-as-Programs Interpretation for ILAL2

In this chapter, we shall demonstrate the proofs-as-programs interpretation for proofs of **ILAL2**. More specifically, we shall show that proofs of **ILAL** can be embedded as terms of λLA in such a way that cut-elimination in **ILAL2** is in full accordance with normalization in λLA. Hence the main result of the previous chapter, the polytime strong normalization theorem, also applies to the proof system of **ILAL2**, too.

The proofs-as-programs interpretation has another consequence. It is known by Girard-Roversi's result [Gir98, Rov99] that all polytime functions are representable as proofs of **ILAL2**. Therefore, our interpretation, in conjunction with the main result of the previous chapter, establishes a characterization of the polytime functions in terms of λLA: A function is polytime if and only if it is representable as a term of λLA.

In Section 4.1, we shall introduce **ILAL2** as a type assignment system for λLA. The presentation there is in a sequent calculus style. It is, however, difficult to prove the subject reduction theorem directly for that sequent system. The main reason is that it does not satisfy the subterm typability. To overcome this, we shall introduce a variant typing system, called $\mathbf{ILAL2}_N$, in Section 4.2. $\mathbf{ILAL2}_N$ is written in a natural deduction style. It is equivalent to **ILAL2** as far as *closed* terms are concerned, and it does satisfy the subterm typability. Once having defined the suitable formalism $\mathbf{ILAL2}_N$, it is easy to prove the subject reduction theorem for $\mathbf{ILAL2}_N$, and the same theorem for **ILAL2** follows from it. This is achieved in Section 4.3. The above mentioned characterization of the polytime functions by λLA is stated in Section 4.4. Finally in Section 4.5, we mention the polytime strong normalizability of the proofnets of **LLL** and give a brief comparison with Lafont's Soft Linear Logic [Laf01]. We also mention some decidability results for type inference in **ILAL** and **ILAL2**.

4.1 ILAL2 as a Type Assignment System

In this section, we shall reformulate **ILAL2** (described in Chapter 2) as a type assignment system for λLA. The present formulation differs from the previous one in the use of Girard's discharged types [Gir98].

$$\frac{}{x\!:\!A \vdash x\!:\!A}\ Id \qquad \frac{\Gamma_1 \vdash u\!:\!A \quad x\!:\!A, \Gamma_2 \vdash t\!:\!C}{\Gamma_1, \Gamma_2 \vdash t[u/x]\!:\!C}\ Cut$$

$$\frac{\Gamma \vdash t\!:\!C}{\Delta, \Gamma \vdash t\!:\!C}\ Weak \qquad \frac{x\!:\![A]_!, y\!:\![A]_!, \Gamma \vdash t\!:\!C}{z\!:\![A]_!, \Gamma \vdash t[z/x, z/y]\!:\!C}\ Cntr$$

$$\frac{\Gamma_1 \vdash u\!:\!A_1 \quad x\!:\!A_2, \Gamma_2 \vdash t\!:\!C}{\Gamma_1, y\!:\!A_1 \multimap A_2, \Gamma_2 \vdash t[yu/x]\!:\!C}\ {\multimap}l \qquad \frac{x\!:\!A_1, \Gamma \vdash t\!:\!A_2}{\Gamma \vdash \lambda x.t\!:\!A_1 \multimap A_2}\ {\multimap}r$$

$$\frac{x\!:\!A[B/\alpha], \Gamma \vdash t\!:\!C}{x\!:\!\forall\alpha.A, \Gamma \vdash t\!:\!C}\ \forall l \qquad \frac{\Gamma \vdash t\!:\!A}{\Gamma \vdash t\!:\!\forall\alpha.A}\ \forall r,\quad (\alpha \text{ is not free in } \Gamma)$$

$$\frac{x\!:\![A]_!, \Gamma \vdash t\!:\!C}{y\,:!A, \Gamma \vdash \mathsf{let}\ y\ \mathsf{be}\ !x\ \mathsf{in}\ t\!:\!C}\ !l \qquad \frac{x\!:\!B \vdash t\!:\!A}{x\!:\![B]_! \vdash !t\,:!A}\ !r$$

$$\frac{x\!:\![A]_\S, \Gamma \vdash t\!:\!C}{y\!:\!\S A, \Gamma \vdash \mathsf{let}\ y\ \mathsf{be}\ \S x\ \mathsf{in}\ t\!:\!C}\ \S l \qquad \frac{\Gamma, \Delta \vdash t\!:\!A}{[\Gamma]_!, [\Delta]_\S \vdash \S t\!:\!\S A}\ \S r$$

In rule $(!r)$, $x\!:\!B$ can be absent. In rule $(\S r)$, Γ and Δ can be empty.

Figure 4.1: Type Assignment System **ILAL2**

Definition 4.1 The *types* of **ILAL2** are given by the following grammar:

$$A, B ::= \alpha \mid A \multimap B \mid \forall\alpha.A \mid !A \mid \S A.$$

A !-*discharged* type is an expression of the form $[A]_!$. A §-*discharged* type is an expression of the form $[A]_\S$.

The connectives and constants of **ILAL2** other than $\{\multimap, \forall, !, \S\}$ are definable from $\{\multimap, \forall\}$ (see Section 2.2 of Chapter 2):

$$\begin{array}{rcl}
\exists\beta.A & \equiv & \forall\alpha.(\forall\beta.(A \multimap \alpha) \multimap \alpha); \\
A \otimes B & \equiv & \forall\alpha.((A \multimap B \multimap \alpha) \multimap \alpha); \\
\mathbf{1} & \equiv & \forall\alpha.(\alpha \multimap \alpha); \\
A \,\&\, B & \equiv & \exists\alpha.((\alpha \multimap A) \otimes (\alpha \multimap B) \otimes \alpha); \\
A \oplus B & \equiv & \forall\alpha.((A \multimap \alpha) \multimap (B \multimap \alpha) \multimap \alpha); \\
\mathbf{0} & \equiv & \forall\alpha.\alpha.
\end{array}$$

We write $!^d A$ and $\S^d A$ to denote $\underbrace{!! \cdots !}_{d\ times} A$ and $\underbrace{\S\S \cdots \S}_{d\ times} A$.

A *declaration* is an expression of the form $x\!:\!A$ or $x\!:\![A]_\dagger$. A finite set of declarations is denoted by Γ, Δ, etc. Let Γ be a set of declarations $x_1\!:\!A_1, \ldots, x_n\!:\!A_n$ where all types in it are non-discharged. Then $[\Gamma]_\dagger$ denotes $x_1 : [A_1]_\dagger, \ldots, x_n : [A_n]_\dagger$. If Γ contains a declaration with a discharged type, then $[\Gamma]_\dagger$ is undefined.

Definition 4.2 The *type inference rules* of **ILAL** are those given in Figure 4.1. We say that a pseudo-term t is *typable* in **ILAL** if $\Gamma \vdash t\!:\!A$ is derivable for some Γ and A by those inference rules.

Since the type inference rules are in full accordance with the logical inference rules of **ILAL2**, it is straightforward that a formula A is provable in **ILAL2** (as a logical system) if and only if there is a pseudo-term t such that $\vdash t\!:\!A$ is derivable in **ILAL2** (as a type assignment system).

Remark 4.3 Observe that if $x\!:\!A, \Gamma \vdash t\!:\!C$, namely x is of undischarged type, then it occurs at most once in t. Therefore, no duplication is caused by the substitutions used in (Cut) and $(\multimap l)$ rules, which always operate on undischarged types. Note the simplicity of our (Cut) rule as compared with the Cut rule of [Asp98] which requires a complicated substitution operation to avoid illegal duplication.

Another effect of using discharged types is that they act as a *barrier* to substitution into boxes. This is reminiscent of the use of *patterns* in Wadler's syntax for Intuitionistic Linear Logic [Wad93]. Indeed, we could alternatively use Wadler's patters to obtain the same effect.

As expected, we have:

Theorem 4.4 *Every typable pseudo-term is a term. More exactly, if* $\vec{x}\!:\!\vec{A}, \vec{y}\!:\![\vec{B}]_!, \vec{z}\!:\![\vec{C}]_\S \vdash t\!:\!D$, *then* $t \in \mathcal{T}_{\{\vec{x}\},\{\vec{y}\},\{\vec{z}\}}$.

Proof. By induction on the length of the typing derivation. In the cases of (Cut) and $(\multimap l)$, apply Lemma 3.6(1). ■

Terms for unary integers and words (or binary integers) are defined in Definition 3.3. To recall the definition, we have

$$\overline{n} \equiv \lambda x.(\mathsf{let}\ x\ \mathsf{be}\ !z\ \mathsf{in}\ \S\lambda y.\underbrace{(z \cdots (z\,y) \cdots)}_{n\ times}),$$

for each integer n. We also have

$$\overline{w} \equiv \lambda x_0 x_1.(\mathsf{let}\ x_0\ \mathsf{be}\ !z_0\ \mathsf{in}\ (\mathsf{let}\ x_1\ \mathsf{be}\ !z_1\ \mathsf{in}\ \S\lambda y.(z_{i_0} \cdots (z_{i_n} y) \cdots)),$$

for each $w \equiv i_0 \cdots i_n \in \{0,1\}^*$. To these terms the following types are to be assigned:

Definition 4.5

$$\begin{aligned} \mathbf{int} &\equiv \forall\alpha.!(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha) \\ \mathbf{bint} &\equiv \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha) \end{aligned}$$

There is, however, a minor problem which is related to the existence of *η-variants.* For instance, we have a term such that

$$\vdash \lambda x.(\mathsf{let}\ x\ \mathsf{be}\ !z\ \mathsf{in}\ \S z) : \mathbf{int},$$

which is not a Church-numeral. Actually, this term is η-equivalent to Church-numeral $\overline{1}$. Similarly, **bint** contains some terms which are η-equivalent to $\overline{0}$ and $\overline{1}$. In what follows, we shall just *identify* these η-variants with their origins. This is a quite harmless convention since the η-equivalence relation is compatible with the reduction relation $\longrightarrow$. It simplifies our presentation considerably.

With this convention, types **int** and **bint** surely serve as data types:

Proposition 4.6 *Let t be a normal term.*

1. *$\vdash t\!:\!\mathbf{int}$ is derivable if and only if $t \equiv \overline{n}$ for some natural number n (or t is an η-variant of $\overline{1}$).*
2. *$\vdash t\!:\!\mathbf{bint}$ is derivable if and only if $t \equiv \overline{w}$ for some $w \in \{0,1\}^*$ (or t is an η-variant of $\overline{0}$ or $\overline{1}$).*

This can be proved through a careful analysis of cut-free derivations; the fact that every typable normal term has a cut-free typing derivation can be easily verified[1]. In a similar way, we can show:

Proposition 4.7 *If $\vdash t : \S A$ is derivable and t is normal, then t is of the form $\S t'$ and $\vdash t' : A$ is derivable.*

An example of untypable terms is Ω_{LA} in Example 3.4. To see the reason, define the *erasure* of a term of λLA to be a λ-term obtained by applying the following operations as much as possible:

$$\begin{array}{rcl} \dagger u & \mapsto & u, \\ \mathsf{let}\ u\ \mathsf{be}\ \dagger x\ \mathsf{in}\ t & \mapsto & t[u/x]. \end{array}$$

If a term is typable in **ILAL**, then its erasure is typable in System F (in the Curry style, see [Bar92]). Now, Ω_{LA} cannot be typed in **ILAL**, since the erasure of Ω_{LA} is Ω, a term which cannot be typed in System F.

Definition 4.8 A function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ is λLA-*represented* by a term t if there is a natural number $d \geq 0$ such that $t\overline{w} \longrightarrow^* \S^d \overline{f(w)}$ for every $w \in \{0,1\}^*$.

By Girard-Roversi's result [Gir98, Rov99], every polytime function is representable as a proof of **ILAL2** (see [AR00] for a good exposition and [MO00a] for an independent proof). In our framework, that result is rephrased as follows:

Theorem 4.9 *Every function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ which is computable in polynomial time is λLA-represented by a term of type $\mathbf{bint} \multimap \S^d\mathbf{bint}$ for some d.*

The converse of this theorem will be taken up in Section 4.4 after the subject reduction theorem has been proved. Below is a remark on the significance of types in our framework.

Remark 4.10 Types are not necessary for the polytime normalizability, but still useful in several ways. Let us summarize the uses of types.

- Types are used to avoid *deadlocks*, such as $(\dagger t)u$ and $\mathsf{let}\ (\lambda x.t)\ \mathsf{be}\ \dagger x\ \mathsf{in}\ u$.
- Data types such as **int** and **bint**, constrain the shape of normal forms. By Propositions 4.6 and 4.7, all normal terms of type $\S^d\mathbf{int}$ are of the form $\S^d\overline{n}$ (or an η-variant of $\S^d\overline{1}$). Similarly, all normal terms of type $\S^d\mathbf{bint}$ are of the form $\S^d\overline{w}$ (or η-variants of $\S^d\overline{0}, \S^d\overline{1}$). .
- Lazy types, including $\S^k\mathbf{int}$ and $\S^k\mathbf{bint}$, tell us the depths of normal forms: say that a type is *lazy* if it does not contain a negative occurrence of $\forall$. If a term t is normal and of lazy type A, then it means that $\vdash t : A$ can be derived without using the $(\forall l)$ inference rule, which has an effect of hiding some information on derivations. Thus all uses of the ! and § inference rules in the derivation are recorded in A. Hence the depth of A immediately bounds the depth of t.

[1]A detailed proof is given in the case of **LST** in Lemma 6.8 (Section 6.3, Chapter 6), and it can be adapted for **ILAL2** straightforwardly (see Remark 6.10).

- The above suggests that in order to normalize a term of lazy type of depth d, we do not have to fire redices at depth $> d$, which will be removed by reductions at lower depths before arriving at the normal form. In this way, lazy types help us detect redundant redices.

4.2 Natural Deduction System $\mathbf{ILAL2}_N$

ILAL2 does not satisfy the subterm typability; for example, $!(xx)$ may be typed as $x:[A]_! \vdash !(xx) : !A$, but its subterm xx cannot be typed. This makes difficult to prove the subject reduction theorem directly for **ILAL2**.

In this section, we introduce a variant system $\mathbf{ILAL2}_N$ based on a natural deduction formalism. Then we show that **ILAL2** and $\mathbf{ILAL2}_N$ are equivalent as far as closed terms are concerned. $\mathbf{ILAL2}_N$ satisfies the subterm typability, and will be convenient for proving the subject reduction theorem in the next section.

The inference rules of $\mathbf{ILAL2}_N$ are those in Figure 4.2.

$$\frac{}{x:A, \Gamma \vdash x:A}\ Ax$$

$$\frac{\Gamma \vdash t:A \multimap B \quad \Gamma \vdash u:A}{\Gamma \vdash tu:B}\ {\multimap}E \qquad \frac{x:A, \Gamma \vdash t:B \quad FO(x,t) \leq 1}{\Gamma \vdash \lambda x.t:A \multimap B}\ {\multimap}I$$

$$\frac{\Gamma \vdash t:\forall\alpha.A}{\Gamma \vdash t:A[B/\alpha]}\ \forall E \qquad \frac{\Gamma \vdash t:A \quad \alpha \notin FV(\Gamma)}{\Gamma \vdash t:\forall\alpha.A}\ \forall I$$

$$\frac{\Gamma \vdash u : !A \quad x:[A]_!, \Gamma \vdash t:B}{\Gamma \vdash \mathsf{let}\ u\ \mathsf{be}\ !x\ \mathsf{in}\ t:B}\ !E \qquad \frac{\Gamma \vdash t:A \quad FO(t) \leq 1}{[\Gamma]_!, \Delta \vdash !t : !A}\ !I$$

$$\frac{\Gamma \vdash u : \S A \quad x:[A]_\S, \Gamma \vdash t:B \quad FO(x,t) \leq 1}{\Gamma \vdash \mathsf{let}\ u\ \mathsf{be}\ \S x\ \mathsf{in}\ t:B}\ \S E \qquad \frac{\Gamma, \Sigma \vdash t:A}{[\Gamma]_!, [\Sigma]_\S, \Delta \vdash \S t : \S A}\ \S I$$

Figure 4.2: Natural Deduction System $\mathbf{ILAL2}_N$

Lemma 4.11 *If $x:\mu, \Gamma \vdash t:A$ is derivable in $\mathbf{ILAL2}_N$ and $x \notin FV(t)$, where μ is either nondischarged or discharged type, then $\Gamma \vdash t:A$ is derivable in $\mathbf{ILAL2}_N$. The same property holds for* **ILAL2***, too.*

Proof. By induction on the derivation. ■

Lemma 4.12 *The following rules are derivable in $\mathbf{ILAL2}_N$:*

$$\frac{\Gamma \vdash t:C}{\Delta, \Gamma \vdash t:C}\ Weak \qquad \frac{x:[A]_!, y:[A]_!, \Gamma \vdash t:C}{z:[A]_!, \Gamma \vdash t[z/x, z/y]:C}\ Cntr \qquad \frac{\Gamma \vdash u:A \quad x:A, \Gamma \vdash t:C}{\Gamma \vdash t[u/x]:C}\ Cut$$

$$\frac{\Delta \vdash u:A \quad x:[A]_!, [\Delta]_!, \Gamma \vdash t:C \quad FO(u) \leq 1}{[\Delta]_!, \Gamma \vdash t[u/x]:C}\ Cut_! \qquad \frac{\Delta, \Sigma \vdash u:A \quad x:[A]_\S, [\Delta]_!, [\Pi]_\S, \Gamma \vdash t:C}{[\Delta]_!, [\Pi]_\S, \Gamma \vdash t[u/x]:C}\ Cut_\S$$

Proof.
($Weak$) By induction on the derivation.
($Cntr$) By induction on the derivation. Actually we show that contraction is derivable not only for !-discharged formulas but also for nondischarged and §-discharged formulas too.
(Cut) By induction on the derivation of $x\!:\!A, \Gamma \vdash t\!:\!C$.
($Cut_!$) By induction on the derivation of $x : [A]_!, [\Delta]_!, \Gamma \vdash t : C$. Let us consider the critical case. Suppose that the last inference rule used in the derivation is ($!I$) of the form:

$$\frac{x : A, \Delta \vdash t' : C' \quad FO(t') \leq 1}{x : [A]_!, [\Delta]_!, \Gamma \vdash !t' : !C'}$$

By (Cut), $\Delta \vdash t'[u/x] : C$ is derivable in $\mathbf{ILAL2}_N$. Moreover $FO(u) \leq 1$ and $FO(t') \leq 1$, hence $FO(t'[u/x]) \leq 1$. Therefore we can apply ($!I$), yielding $[\Delta]_!, \Gamma \vdash !t'[u/x] : !C'$.
($Cut_§$) By induction on the derivation of $x\!:\![A]_§, [\Delta]_!, [\Pi]_§, \Gamma \vdash t\!:\!C$. ■

Lemma 4.13 *If $\Gamma \vdash t\!:\!A$ is derivable in* **ILAL2**, *then it is also derivable in* $\mathbf{ILAL2}_N$.

Proof. By induction on the derivation, using derived rules ($Weak$), ($Cntr$) and (Cut) in Lemma 4.12. ■

A *variable substitution* is a function θ from the set of term variables to itself. $t\theta$ denotes the term obtained from t by replacing each free variable x in t with $\theta(x)$. $\Gamma\theta$ denotes the set of declarations obtained by replacing each variable x in it by $\theta(x)$. It may decrease the number of declarations due to unification. For example, If $\Gamma \equiv x\!:\!A, y\!:\!A$ and $\theta(x) = \theta(y) = z$, then $\Gamma\theta \equiv z\!:\!A$.

Lemma 4.14 *Let $\Gamma \vdash t\!:\!A$ be derivable in* $\mathbf{ILAL2}_N$. *Then there are Γ', t' and a variable substitution θ such that*

- $\Gamma' \vdash t'\!:\!A$ *is derivable in* **ILAL2**,
- $\Gamma'\theta \equiv \Gamma$ *and* $t'\theta \equiv t$.

Proof. By induction on the derivation. We shall only treat several critical cases.
(Case 1) The last inference is $\multimap E$ of the form:

$$\frac{\Gamma \vdash t\!:\!A \multimap B \quad \Gamma \vdash u\!:\!A}{\Gamma \vdash tu\!:\!B} \multimap E$$

By the induction hypothesis, there are Γ'_1, Γ'_2, t', u' and variable substitutions θ_1 and θ_2 such that

- $\Gamma'_1 \vdash t'\!:\!A \multimap B$ and $\Gamma'_2 \vdash u'\!:\!A$ are derivable in **ILAL2**,
- $\Gamma'_1\theta_1 \equiv \Gamma'_2\theta_2 \equiv \Gamma$, $t'\theta_1 \equiv t$ and $u'\theta_2 \equiv u$.

Without loss of generality, we may assume that $FV(\Gamma'_1)$ and $FV(\Gamma'_2)$ are disjoint. From these, we see that $\Gamma'_1, \Gamma'_2 \vdash t'u'\!:\!B$ is derivable in **ILAL2**, we can define a suitable variable substitution θ by

$$\begin{array}{rcl} \theta(x) & = & \theta_1(x) \text{ if } x \in FV(\Gamma'_1); \\ & = & \theta_2(x) \text{ otherwise.} \end{array}$$

(Case 2) The last inference is $\multimap I$ of the form:

$$\frac{x\!:\!A, \Gamma \vdash t\!:\!B \quad FO(x,t) \leq 1}{\Gamma \vdash \lambda x.t\!:\!A \multimap B} \multimap I$$

By the induction hypothesis, there are $x_1\!:\!A, \ldots, x_n\!:\!A$, Γ', t' and a variable substitution θ such that

- $x_1\!:\!A, \ldots, x_n\!:\!A, \Gamma' \vdash t'\!:\!B$ is derivable in **ILAL2**,
- $\theta(x_1) = \cdots = \theta(x_n) = x$, $\Gamma'\theta \equiv \Gamma$ and $t'\theta \equiv t$.

Since $FO(x,t) \leq 1$, there is at most one x_i such that $x_i \in FV(t')$. By Lemma 4.11, $x_i\!:\!A, \Gamma' \vdash t'\!:\!A$ is derivable in **ILAL2**. Rename x_i as x. Then we can apply ($\multimap r$) to derive $\Gamma' \vdash \lambda x.(t'[x/x_i])\!:\!A$ in **ILAL2**.

(Case 3) The last inference is $!E$ of the form:

$$\frac{\Gamma \vdash u :!A \quad x\!:\![A]_!, \Gamma \vdash t\!:\!B}{\Gamma \vdash \mathsf{let}\ u\ \mathsf{be}\ !x\ \mathsf{in}\ t\!:\!B} !E$$

By the induction hypothesis, there are Γ_1', $x_1\!:\![A]_!, \ldots, x_n\!:\![A]_!$, Γ_2', t', u' and variable substitutions θ_1 and θ_2 such that

- $\Gamma_1' \vdash u' :!A$ and $x_1\!:\![A]_!, \ldots, x_n\!:\![A]_!, \Gamma_2' \vdash t'\!:\!B$ are derivable in **ILAL2**,
- $\theta(x_1) = \cdots = \theta(x_n) = x$, $\Gamma_1'\theta_1 \equiv \Gamma_2'\theta_2 \equiv \Gamma$, $u'\theta_1 \equiv u$ and $t'\theta_2 \equiv t$.

We may assume that $FV(\Gamma_1')$, $FV(\Gamma_2')$ and $\{x_1, \ldots, x_n\}$ are disjoint. By ($Cntr$), ($!l$) and (Cut), $\Gamma_1', \Gamma_2' \vdash \mathsf{let}\ u'\ \mathsf{be}\ !x\ \mathsf{in}\ (t'[x/x_1, \ldots, x/x_n])$ is derivable in **ILAL2**. A suitable variable substitution θ is defined as in (Case 1). ■

As a consequence, we obtain:

Corollary 4.15 *$\vdash t\!:\!A$ is derivable in* **ILAL2** *if and only if it is derivable in* $\mathbf{ILAL2}_N$.

Proof. By Lemma 4.13 and Lemma 4.14. ■

4.3 Subject Reduction Theorem

In this section we prove the subject reduction theorem for $\mathbf{ILAL2}_N$, from which the same theorem for **ILAL2**easily follows. For the treatment of quantifiers, we use a technique which is exploited in [Bar92] to show the subject reduction theorem for System F in the Curry style.

Define a binary relation $>$ on types by

$$\begin{array}{rcl} A & > & \forall\alpha.A \\ \forall\alpha.A & > & A[B/\alpha]. \end{array}$$

Denote the reflexive transitive closure of $>$ by $\geq$.

Lemma 4.16 $A \multimap B \geq A' \multimap B' \Longrightarrow \exists\vec{\alpha}\exists\vec{C}\ A' \multimap B' \equiv (A \multimap B)[\vec{C}/\vec{\alpha}]$.

Proof. See Lemma 4.2.4 of [Bar92]. ■

Lemma 4.17 (Generation lemma) *The following hold for* $\mathbf{ILAL2}_N$*:*

1. $\Gamma \vdash x\!:\!A \Longrightarrow \Gamma \equiv x\!:\!B, \Gamma'$ *for some* $B \geq A$.
2. $\Gamma \vdash \lambda x.t\!:\!A \Longrightarrow x\!:\!B, \Gamma \vdash t\!:\!C$ *for some* $B \multimap C \geq A$*, and* $FO(x,t) \leq 1$.
3. $\Gamma \vdash tu\!:\!A \Longrightarrow \Gamma \vdash t\!:\!B \multimap C$ *and* $\Gamma \vdash u\!:\!B$ *for some* B *and* $C \geq A$.
4. $\Gamma \vdash \mathsf{let}\ u\ \mathsf{be}\ !x\ \mathsf{in}\ t\!:\!A \Longrightarrow \Gamma \vdash u :!B$ *and* $x\!:\![B]_!, \Gamma \vdash t\!:\!C$ *for some* B *and* $C \geq A$.
5. $\Gamma \vdash !t\!:\!A \Longrightarrow \Delta \vdash t\!:\!C$ *for some* $!C \geq A$ *and* $[\Delta]_! \subseteq \Gamma$*, and* $FO(t) \leq 1$.
6. $\Gamma \vdash \mathsf{let}\ u\ \mathsf{be}\ \S x\ \mathsf{in}\ t : A \Longrightarrow \Gamma \vdash u : \S B$ *and* $x : [B]_\S, \Gamma \vdash t : C$ *for some* B *and* $C \geq A$*, and* $FO(x,t) \leq 1$.
7. $\Gamma \vdash \S t\!:\!A \Longrightarrow \Delta, \Sigma \vdash t\!:\!C$ *for some* $\S C \geq A$ *and* $[\Delta]_!, [\Sigma]_\S \subseteq \Gamma$.

Proof. By induction on derivations. ■

Theorem 4.18 (Subject Reduction for $\mathbf{ILAL2}_N$) *If* $\Gamma \vdash t\!:\!A$ *is derivable in* $\mathbf{ILAL2}_N$ *and* $t \longrightarrow u$*, then* $\Gamma \vdash u\!:\!A$ *is derivable in* $\mathbf{ILAL2}_N$.

Proof. The proof is quite similar to that of Proposition 3.11. Let $\Gamma \vdash \Phi[t]\!:\!A$ be derivable in $\mathbf{ILAL2}_N$, $\Phi[t] \to \Phi[u]$ and t be the redex of the reduction. We prove that $\Gamma \vdash \Phi[u]\!:\!A$ and

(*) $FO(x, \Phi[u]) \leq FO(x, \Phi[t])$ for each x such that either $x\!:\!A \in \Gamma$ or $x\!:\![A]_\S \in \Gamma$ for some A.

by induction on Φ.

When $\Phi \equiv \bullet$, the above are shown using Generation Lemma and the derived rules (Cut), $(Cut_!)$ and $(Cut_\S)$ given in Lemma 4.12. Other cases are easy. ■

Corollary 4.19 (Subject Reduction for ILAL2) *If* $\Gamma \vdash t\!:\!A$ *is derivable in* **ILAL2** *and* $t \longrightarrow u$*, then* $\Gamma \vdash u\!:\!A$ *is derivable in* **ILAL2**.

In this chapter, **ILAL2** has been considered as a type assignment system for λLA. This means that every proof of **ILAL2** (as a logical system) is structurally representable by a term of λLA (*structurally* because formulas/types are erased and inference rules for second order quantifiers are ignored in terms of λLA). Corollary 4.19 basically means that cut-elimination in **ILAL2** (as a logical system) is in full accordance with normalization in λLA. In this context, it is important to mention a direct consequence of Corollary 3.26 in the previous chapter:

Corollary 4.20 (Polytime strong normalization for ILAL2 proofs) *Every structural representation of a proof of* **ILAL2** *(as a term of* λLA*) is polytime strongly normalizable in the sense of Corollary 3.26.*

4.4 Characterization of Polytime Functions

As a consequence of the subject reduction theorem and the polytime strong normalization theorem, we have the following result:

Theorem 4.21 *Every term t of type* $\mathbf{bint} \multimap \S^d\mathbf{bint}$ *λ-represents a function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ which is computable in time $O(n^{2^{d+3}})$.*

Proof. Recall that all $\overline{w}$'s are of depth 1, so that $t\overline{w}$ is of constant depth for every $w \in \{0,1\}^*$. Without loss of generality, we may assume that the depth is equal to the depth of $\S^d\mathbf{bint}$, i.e., $d+1$ (just ignore the deeper layers, which do not contribute to the normal form; see Remark 4.10). By Corollary 3.26, the normal form of $t\overline{w}$ is computed in time $O(|t\overline{w}|^{2^{d+3}})$, thus in time $O(|w|^{2^{d+3}})$ (by taking a reasonable reduction strategy of low complexity). The type of $t\overline{w}$ is $\S^d\mathbf{bint}$, hence so is the type of its normal form by the subject reduction theorem. By Proposition 4.6 (2) and Proposition 4.7, the normal form should be of the form $\S^d\overline{w'}$, and such w' is unique by the Church-Rosser property.■

Therefore we obtain a characterization of the polytime functions:

Corollary 4.22 (Characterization of the Polytime Functions) *A function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ is polytime computable if and only if it is λ-represented by a* λLA *term of type* $\mathbf{bint} \multimap \S^d\mathbf{bint}$ *for some d.*

4.5 Remarks

In this chapter, we have reformulated **ILAL2** as a type assignment system for λLA and proved the subject reduction theorem (Corollary 4.19). It has implied, in conjunction with the main result of the previous chapter, the polytime strong normalizability of (structural representations of) proofs of **ILAL2** (Corollary 4.20). It has also yielded a characterization of polytime in terms of λLA (Corollary 4.22). Some remarks are in order.

Polytime strong normalization for LLL. Let us discuss the polytime strong normalizability of **LLL** proofnets. As a preliminary, consider the following decompositions of the $(!)$ reduction rule:

$(!_1)$ $\mathsf{let}\ !u\ \mathsf{be}\ !x\ \mathsf{in}\ \Phi[x] \longrightarrow \mathsf{let}\ !u\ \mathsf{be}\ !x\ \mathsf{in}\ \Phi[u]$;

$(!_2)$ $\mathsf{let}\ !u\ \mathsf{be}\ !x\ \mathsf{in}\ t \longrightarrow t$, if $x \notin FV(t)$.

Clearly the $(!)$ reduction rule is simulated by these two. With this modification, we still have the polytime strong normalization theorem. Note that these rules are natural counterparts of Girard[Gir98]'s reduction rules for the exponential boxes: $(!_1)$ corresponds to the contraction reduction and $(!_2)$ to the weakening reduction.

Given this, it is quite plausible that we can apply our technique to **LLL** to show the strong polytime normalization theorem for the proofnets of **LLL** (with formulas erased). There is, however, a limitation that additives should be treated in a lazy way, because eager reductions for additive components cost exponential time.

	Type Checking	Typability	Inhabitation
ILAL	yes	yes	yes
ILAL2	?	?	no

Table 4.1: Decision Problems for Type Inference

Exact Runtime of normalization. In [AR00] it was shown that the functions computable in time $O(n^d)$ are representable by terms of type $\mathbf{bint} \multimap \S^{d+6}\mathbf{bint}$. Professor Mairson, on the other hand, pointed out to the author that the representation could be considerably remedied with respect to the depth; e.g., we can replace the depth $d+6$ above with log d multiplied by a constant.

The reason is as follows. The encoding of [AR00] consumes $d+3$ depths to represent polynomials of degree d. However, if we begin with a term for squaring n^2 of type $\mathbf{int} \multimap \S^2\mathbf{int}$ and compose it d times, we can obtain a term for n^{2^d} of type $\mathbf{int} \multimap \S^{2d}\mathbf{int}$. Hence it only requires depth 2log$d$ to represent polynomial n^d.

This fact induces a lowerbound $O(s^{2^{\frac{d-1}{2}}})$ for the time for normalizing a λLA term of size s and depth d. On the other hand, we have an upperbound $O(s^{2^d+2})$, as shown in the previous chapter. It seems possible to sharpen both of these two bounds so as to establish a finer characterization of the expressibility of λLA. We leave it to the future work.

Decision Problems for Type Inference. Given a type assignment system for some term calculus, it is important to know how complex type inference in it is. There are three questions which are frequently asked to estimate the complexity of type inference.

Type Checking: Given a term t and type A, does $\vdash t : A$ hold?

Typability: Given a term t, is there any type A such that $\vdash t : A$ holds?

Inhabitation: Given a type A, is there any term t such that $\vdash t : A$ holds?

The answers for these questions are summarized in Table 4.1.

Type checking and typability are decidable for **ILAL**, as (essentially) shown by Roversi [Rov00]. In fact, Roversi achieved more; he considered a type system **ILAL** with ML-like polymorphism and extended the standard Hindley-Milner algorithm for computing principal types [Hin69, Mil78] to his type system. His term calculus is different from ours, but there seems to be virtually no problem in repeating his argument for our term calculus.

Inhabitation is also decidable for **ILAL**; this follows from the fact that **ILAL** as a logical system is decidable. This result will be proved in Chapter 7 as a consequence to the finite model property for **ILAL**. On the other hand, inhabitation for **ILAL2** is undecidable, since **ILAL2** is undecidable as a logical system (see Section 2.4 of Chapter 2).

We strongly believe that type checking and typability for **ILAL2** are also undecidable, in view of the fact that the corresponding problems for System F in the Curry style are undecidable [Wel94]. It seems that essentially the same argument as [Wel94] goes through for **ILAL2**. But it still requires of a careful examination. We leave it to the future work.

Soft Linear Logic. Recently Lafont [Laf01] introduced yet another polytime system based on refinement of Linear Logic, which is called *Soft Linear Logic* (SLL). Like Light Logic, the refinement consists in a restriction on the exponential modality of Linear Logic, but remarkably SLL does not require any extra modality such as §, hence is much more elegant than Light Logic as a logical system. In addition, the (strong) normalization theorem is very easy to prove.

In spite of the emergence of this new elegant system, we still consider significant to continue the investigation of Light Logic for the following reasons:

- Certain basic functions are more naturally representable in Light Logic than in SLL. Typically, addition is slightly unnatural in SLL, because it is represented not of type $\mathbf{int} \multimap \mathbf{int} \multimap \mathbf{int}$, where $\mathbf{int} \equiv \forall\alpha.!(\alpha \multimap \alpha) \multimap (\alpha \multimap \alpha)$, but of type

  $$\mathbf{int} \multimap \mathbf{int} \multimap \forall\alpha.!(\alpha \multimap \alpha)\otimes!(\alpha \multimap \alpha) \multimap (\alpha \multimap \alpha).$$

  The reason is that the contraction rule of SLL is too restrictive. It seems that programming in Light Logic is more natural than in SLL in such cases.

- Although the modalities of two systems looks alike at first, they are actually quite different in their computational power; λLA terms of depth d are normalizable in $O(s^{2^{d+1}})$ steps, while SLL proofs of depth d are in $O(s^d)$ steps.

- Several semantic studies have already been made for Light Logic [KOSar, Bai00, MO00b]. On the other hand, it is not very obvious whether similar (or entirely different) semantics can be considered for SLL.

Chapter 5

Light Set Theory

Light Logic is not just a type system, but also a formal system of reasoning. To examine the reasoning aspect of Light Logic, we shall study naive set theory based on Light Logic, which we call *Light Set Theory*, in this chapter.

Naive set theory is characterized by the comprehension principle, saying that for any formula A and a variable x there is a set $\{x|A\}$ such that

$$A[t/x] \leftrightarrow t \in \{x|A\}.$$

The theory well-captures the basic intuition of sets as collections of elements satisfying certain properties. It is strong enough to develop almost all existing mathematical theory based on it, but unfortunately it is inconsistent with Classical and Intuitionistic Logics: in naive set theory, we can define a formula A which is equivalent to $\neg A$ (let R be the Russell's set defined as $\{x|x \notin x\}$ and let A be $R \in R$). In terms of sequent calculus, it means that both $A \vdash \neg A$ and $\neg A \vdash A$ are provable. From these two sequents, we can derive contradiction (i.e., the empty sequent) in Classical/Intuitionistic Logic:

$$\dfrac{\dfrac{\dfrac{A \vdash \neg A \qquad \dfrac{\dfrac{\neg A \vdash A}{\neg A, \neg A \vdash}}{\neg A \vdash}\,(Contr)}{A \vdash}}{\vdash \neg A} \qquad \dfrac{\dfrac{\neg A \vdash A}{\neg A, \neg A \vdash}}{\neg A \vdash}\,(Contr)}{\vdash}$$

The above proof contains Contraction. Since its use is so crucial, it is naturally expected that the existence of formula A above does not necessarily imply contradiction when the use of Contraction is somehow limited. It was Grishin [Gri74] who first observed that contraction-free logics are indeed consistent with the naive comprehension principle. Since then, naive set theory has been investigated in the framework of contraction-free logics (see, e.g., [Gri81, Whi93, Shi96, Shi99]). As observed in [Gir98], Light Logic is also consistent with naive comprehension, due to the tamed use of Contraction, hence it makes sense to investigate Light Set Theory.

Light Set Theory is introduced on the basis of **LLL** in [Gir98], but things become much more transparent if we base it on a simpler logical system. Hence we reformulate it on the basis of **ILAL**and call the resulting system **LST**. In Section 5.1, we shall describe syntax of **LST**. In Section 5.2, we shall mention several basic facts on it. Such basic facts include the cut-elimination theorem for **LST**,

Comprehension:

$$\frac{A[t/x], \Gamma \vdash C}{t \in \{x|A\}, \Gamma \vdash C} \in l \quad \frac{\Gamma \vdash A[t/x]}{\Gamma \vdash t \in \{x|A\}} \in r$$

Set Quantifiers:

$$\frac{A[t/x], \Gamma \vdash C}{\forall x.A, \Gamma \vdash C} \forall l \quad \frac{\Gamma \vdash A}{\Gamma \vdash \forall x.A} \forall r$$

$$\frac{A, \Gamma \vdash C}{\exists x.A, \Gamma \vdash C} \exists l \quad \frac{\Gamma \vdash A[t/\alpha]}{\Gamma \vdash \exists x.A} \exists r$$

In rule $(\forall r)$, x is not free in Γ. In rule $(\exists l)$, x is not free in Γ and C.

Figure 5.1: Inference Rules of Light Set Theory (**LST**)

availability of fixpoints, representability of all recursive functions and undecidability of **LST**. Section 5.3 is devoted to an investigation of natural numbers and numeric functions in **LST**. In particular, a restricted form of the induction schema is studied. In Section 5.4, we shall exhibit various mathematical structures and functions which are representable in **LST**. In Section 5.5, we shall give an encoding of polytime Turing machines, and prove the main result of this chapter that all polytime functions are provably total in **LST**.

5.1 Syntax of LST

Here we shall describe syntax of **LST**. It comes from the appendix of [Gir98], but we shall base it on **ILAL**.

Definition 5.1 The *terms and formulas of Light Set Theory* (**LST**) are defined simultaneously as follows:

- Term variables $x, y, z, \ldots$ are terms;
- If A is a formula and x is a term variable, then $\{x|A\}$ is a term;
- If t and u are terms, then $t \in u$ is a formula;
- $\mathbf{1}$, $\top$, $\mathbf{0}$ are formulas;
- If A and B are formulas, then so are $A \otimes B$, $A \multimap B$, $A \,\&\, B$, $A \oplus B$, $!A$ and $\S A$;
- If A is a formula and x is a term variable, then $\forall x.A$ and $\exists x.A$ are formulas.

The inference rules of **LST** are those of **ILAL** with the additional rules in Figure 5.1.

We use $A, B, C, \ldots$ to denote formulas, and $t, u, v, \ldots$ to denote terms. Notation $A[t/x]$ is used to denote the formula which is obtained from A by replacing all occurrences of term variable x with t. Similarly, we use substitution notations for terms and sequence of formulas, such as $u[t/x]$ and $\Gamma[t/x]$, with the obvious meaning.

Negation is defined by means of $\mathbf{0}$:

Definition 5.2

$$\begin{array}{rcl} \neg A & \equiv & A \multimap \mathbf{0} \\ t \notin u & \equiv & \neg t \in u \\ t \neq u & \equiv & \neg t = u \end{array}$$

This definition is compatible with the standard intuitionistic inference rules for negation:

Proposition 5.3 *The following formulas are provable in* **LST***;*

1. *$A, \Gamma \vdash \mathbf{0}$ implies $\Gamma \vdash \neg A$.*
2. *$\Gamma \vdash A$ implies $\neg A, \Gamma \vdash \mathbf{0}$.*
3. *$A, \neg A \vdash B$.*

5.2 Fundamentals of LST

In this section, we shall recall basic notions and properties of Light Set Theory, such as cut-elimination and its consequences (in 5.2.1), properties of equality (in 5.2.2), some basic set-theoretic operations (in 5.2.3), the fixpoint theorem (in 5.2.4) and numeralwise representability of all recursive functions and undecidability of **LST** (in 5.2.5). We owe most materials below to [Gir98], [Gri81] and [Shi99], except the undecidability result for (the modality-free fragment of) **LST**; the latter is immediately obtained from Shirahata's result, but it seems that this is the first place to mention it explicitly. All the results of this section hold for the modality-free fragment of **LST** as well.

5.2.1 Some Basic Facts

Throughout this chapter, we presuppose the cut-elimination theorem for **LST**:

Theorem 5.4 *If A is provable in* **LST***, then it is cut-free provable in* **LST***.*

A proof will be given in Chapter 6. As a consequence we have:

Corollary 5.5 **LST** *is consistent, i.e.,* $\mathbf{0}$ *is not provable in it.*

Corollary 5.6

1. *(Disjunction Property) If $A \oplus B$ is provable, then either A or B is provable.*
2. *(Existence Property) If $\exists x A$ is provable, then $A[t/x]$ is provable for some term t.*
3. *(Modality Property) If $!A$ or $\S A$ is provable, then A is provable.*

Another fact which will be frequently used without mentioning is that the provable sequents are closed under substitution:

Proposition 5.7 *Let t be a term. If $\Gamma \vdash C$ is provable in* **LST***, then $\Gamma[t/x] \vdash C[t/x]$ is also provable in* **LST***.*

Proof. By induction on the length of the proof, noting that formulas are considered up to α-equivalence, i.e., $\forall x A \equiv \forall y(A[y/x])$. ■

5.2.2 Equality

The first thing to do is to define an equality relation. Unfortunately, the standard *extensional equality* defined by:

$$t =_e u \;\equiv\; \forall x(x \in t \multimapboth x \in u)$$

is not appropriate, since it does not satisfy the basic properties of equality. Alternatively, Girard [Gir98] used *Leibniz equality*, which was defined as follows:

Definition 5.8

$$t = u \;\equiv\; \forall x(t \in x \multimap u \in x).$$

Note that Leibniz equality is very strong in that it equates only syntactically identical expressions:

Proposition 5.9 *$t = u$ is provable in* **LST** *iff t and u are syntactically identical.*

Proof. Since $\forall x(t \in x \multimap u \in x), t \in x \vdash u \in x$ is provable in **LST**, if we have $t = u$, then we also obtain $t \in x \vdash u \in x$. By the cut-elimination theorem, it should be an axiom. Hence t and u should be syntactically identical. The other direction is immediate. ■

For example, **LST** does not prove $\{x|A \oplus B\} = \{x|B \oplus A\}$.

The following are basic properties of Leibniz equality:

Proposition 5.10 *The following formulas are provable in* **LST***;*

1. $t = t$.
2. $t = u \multimap (A[t/x] \multimap A[u/x])$.
3. $t = u \multimap u = t$.
4. $t = u \otimes u = r \multimap t = r$.
5. $t = u \multimap t = u \otimes t = u$.
6. $t = u \multimap t =_e u$.

Proof.

1. Immediate.
2. We have $A[t/x] \vdash t \in \{x|A\}$ and $u \in \{x|A\} \vdash A[u/x]$.Hence

   $$t \in \{x|A\} \multimap u \in \{x|A\} \vdash A[t/x] \multimap A[u/x].$$

   Therefore, the claim holds by rule $(\in l)$.
3. By 1 and 2, taking $A \equiv x = t$.
4. By 1 and 2, it holds that $u = t \multimap u = r \multimap t = r$. Hence the claim holds by 3.

5. By 2, we have
$$t = u \multimap (t = t \otimes t = t \multimap t = u \otimes t = u).$$
Now use 1 twice.

6. By 2, we have $t = u \multimap (t =_e t \multimap t =_e u)$, while $t =_e t$ is easily proved.

■

Note that statement 5 above means that Contraction is freely available for equational formulas. The last statement says that Leibniz equality implies the extensional equality. We do not have the converse, however; indeed the converse is inconsistent:

Theorem 5.11 (Grishin[Gri81]) LST *with axiom $t =_e u \multimap t = u$ is inconsistent.*

Proof. We show that the axiom $t =_e u \multimap t = u$ implies Contraction for all formulas. Given a formula A, let $t \equiv \{x|\mathbf{1}\}$ and $u \equiv \{x|A\}$. Then we have $t =_e u \circ\!\!-\!\!\circ A$. On the other hand, $t = u \circ\!\!-\!\!\circ t =_e u$ by assumption. Since Contraction is freely available for $t = u$ by Proposition 5.10(5), it follows that Contraction is also available for A.

By taking Russell's formula in the beginning of this chapter as A, we can simulate the proof of contradiction given there. ■

We use the following abbreviations:

Definition 5.12
$$\begin{array}{rcl}
\forall x \in t.A & \equiv & \forall x(x \in t \multimap A); \\
\exists x \in t.A & \equiv & \exists x(x \in t \otimes A); \\
\exists^! x.A & \equiv & \exists x(A \otimes \forall y(A[y/x] \multimap y = x)); \\
\exists^! x \in t.A & \equiv & \exists x \in t(A \otimes \forall y(A[y/x] \multimap y = x)).
\end{array}$$

5.2.3 Set Theoretic Operations

Let us define some set theoretic operations.

Definition 5.13
$$\begin{array}{rcl}
\emptyset & \equiv & \{x|\mathbf{0}\} \\
\{t\} & \equiv & \{x|x = t\} \\
\{t, u\} & \equiv & \{x|x = t \oplus x = u\} \\
\{t_1, \ldots, t_n\} & \equiv & \{x|x = t_1 \oplus \cdots \oplus x = t_n\} \\
t \cup u & \equiv & \{x|x \in t \oplus x \in u\} \\
\langle t, u\rangle & \equiv & \{\{t\}, \{t, u\}\} \\
\langle t_1, \ldots, t_n\rangle & \equiv & \langle \cdots \langle\langle t_1, t_2\rangle, t_3\rangle \cdots, t_n\rangle
\end{array}$$

Proposition 5.14 *The following are provable in* **LST***;*

1. $t \notin \emptyset$.

2. $t \in \{u\} \multimap\!\circ\; t = u$.

3. $t \in \{u, v\} \multimap\!\circ\; t = u \oplus t = v$.

4. $\langle t, u\rangle = \langle r, s\rangle \multimap\!\circ\; t = r \otimes u = s$.

Proof.

1. From $\mathbf{0} \vdash \mathbf{0}$, we obtain $t \in \{x|\mathbf{0}\} \vdash \mathbf{0}$ by rule $(\in l)$.
2. By definition.
3. By definition.
4. The proof is familiar in the case of the standard axiomatic set theory, and we can repeat just the same argument in **LST**, since Contraction is available for all equational formulas. A complete proof can be found in [Shi99].

■

5.2.4 Fixpoint Theorem

One of the most interesting aspects of Light Set Theory is that any formula has a fixpoint:

Theorem 5.15 (Fixpoint Theorem, Girard[Gir98])

1. *For any formula A, there exists a term f such that*
$$t \in f \multimap\!\circ\; A[f/y, t/x]$$
is provable for any t.
2. *More generally, for any formula A, there exists a term f such that*
$$\langle t_1, \ldots, t_n\rangle \in f \multimap\!\circ\; A[f/y, t_1/x_1, \ldots, t_n/x_n]$$
is provable for any $t_1, \ldots, t_n$.

Proof. As for the first claim, define

$$\begin{array}{rcl} s & \equiv & \{z \mid \exists u \exists v(z = \langle u, v\rangle \otimes A[\{w \mid \langle w, v\rangle \in v/y, u/x])\}; \\ f & \equiv & \{w \mid \langle w, s\rangle \in s\}, \end{array}$$

where u, v and w are fresh variables. Then we can derive the desired property. A complete proof can be found in [Shi99]. The second claim is just a generalization of the first. ■

5.2.5 Undecidability of LST

In [Shi99], Shirahata defined the numeral $\underline{n}$ for each natural number n by:

$$\begin{array}{rcl} \underline{0} & = & \emptyset; \\ S(t) & = & t \cup \{t\}; \\ \underline{n} & \equiv & \underbrace{S(\cdots S(S}_{n\ times}(\underline{0}))\cdots), \end{array}$$

and proved the following by using the fixpoint theorem:

Theorem 5.16 (Shirahata[Shi99]) *Every total recursive function is numeralwise representable in (the modality-free fragment of)* **LST***; i.e., for every k-ary recursive function F, there exists a term f such that*

- *for any $\vec{n} \in N^k$, if $F(\vec{n}) = m$, then*

  $\vdash \langle \underline{\vec{n}}, \underline{m} \rangle \in f$ *and*
  $\vdash \forall x(\langle \underline{\vec{n}}, x \rangle \multimap x = \underline{m})$ *are provable.*

In what follows, we shall strengthen this result to weak numeralwise representability of all recursively enumerable predicates.

Let N^* be the fixpoint

$$x \in N^* \ \circ\!\!-\!\!\circ \ x = \underline{0} \oplus \exists y \in N^*(x = S(y)).$$

Then we have:

Lemma 5.17 $\vdash t \in N^*$ *is provable in* **LST** *if and only if t is a numeral $\underline{n}$.*

Proof. The "if" direction is proved by induction on n. If $n = 0$ then we have

$$\dfrac{\dfrac{\vdash \underline{0} = \underline{0}}{\vdash \underline{0} = \underline{0} \oplus \exists y \in N^*(\underline{0} = S(y))}}{\vdash \underline{0} \in N^*} \ .$$

If $n = m + 1$ then by the induction hypothesis, $\vdash \underline{m} \in N^*$ is provable in **LST**. Therefore we have:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash \underline{m} \in N^* \quad \vdash S(\underline{m}) = S(\underline{m})}{\vdash \underline{m} \in N^* \otimes S(\underline{m}) = S(\underline{m})}}{\vdash \exists y \in N^*(S(\underline{m}) = S(y))}}{\vdash S(\underline{m}) = \underline{0} \oplus \exists y \in N^*(S(\underline{m}) = S(y))}}{\vdash S(\underline{m}) \in N^*} \ .$$

The "only-if" direction is proved by induction on the size of term t (i.e., the number of symbols in t). Suppose that $\vdash t \in N^*$ is provable. Then either $\vdash t = \underline{0}$ or $\vdash \exists y \in N^*(t = S(y))$ is provable by the disjunction property.

In the former case, t is syntactically equivalent to $\underline{0}$ by Proposition 5.9.

In the latter case, there is some term u such that $\vdash u \in N^* \otimes t = S(u)$ is provable by the existence property. Therefore $\vdash u \in N^*$ and $\vdash t = S(u)$ are both provable. Thus t is syntactically equivalent to $S(u)$, and hence the induction hypothesis applies to u, which is of smaller size than t. Therefore we see that u is of the form $\underline{m}$ for some $m \in N$. Therefore $t \equiv \underline{m+1}$. ■

Using the above lemma, we can show:

Theorem 5.18 *Every recursively enumerable predicate is weakly numeralwise representable in (the modality-free fragment of)* **LST**. *Namely, for every k-ary predicate $R \subseteq N^k$ there exists a formula A of* **LST** *such that*

$$\langle n_1, \ldots, n_k\rangle \in R \iff \vdash A[\underline{n_1}/x_1, \ldots, \underline{n_k}/x_k]$$

for any $\langle n_1, \ldots, n_k\rangle \in N^k$.

Proof. Let $R \subseteq N^k$ be a recursively enumerable predicate. Without loss of generality, we may assume that there is a recursive predicate $Q \subseteq N^{k+1}$ such that

$$\langle \vec{n}\rangle \in R \iff \text{there exists } m \in N \text{ such that } \langle \vec{n}, m\rangle \in Q.$$

By Theorem 5.16, we have a formula B of **LST** which numeralwise represents Q. Now,

$$\begin{aligned} \langle \vec{n}\rangle \in R &\iff \text{there exists } m \in N \text{ such that } \langle \vec{n}, m\rangle \in Q \\ &\iff \text{there exists } m \in N \text{ such that } \vdash B[\underline{\vec{n}}/\vec{x}, \underline{m}/y] \\ &\iff \vdash \exists y \in N^* B[\underline{\vec{n}}/\vec{x}] \end{aligned}$$

(The previous lemma is used to derive the last equivalence.) Therefore, the formula $\exists y \in N^* B$ weakly numeralwise represents R. ■

Since the class of recursively enumerable predicates exceeds the class of recursive (decidable) predicates (see, e.g., [Sho67]), we conclude:

Corollary 5.19 **LST** *is undecidable.*

Remark 5.20 Note that this undecidability result for **LST** cannot be obtained as a simple modification of the undecidability result for **ILAL2** (Theorem 2.4); by using the encoding of [LSS95], we could simulate naive set theory based on Intuitionistic Logic. But it is not useful at all, since the latter is inconsistent, thus trivially decidable.

5.3 Natural Numbers

Now let us investigate natural numbers and their properties in **LST**. We shall define the set of natural numbers in 5.3.1, then introduce a restricted form of Induction, called Light Induction, which is available in **LST** (in 5.3.2). Using Light Induction, we shall show that addition and multiplication are provably total in **LST** (in 5.3.3).

5.3.1 Numerals

The previous definition of natural numbers based on unordered pairs is not satisfactory, because it does not yield $\mathsf{S}(x) = \mathsf{S}(y) \multimap x = y$. Alternatively, we define natural numbers based on ordered pairs:

Definition 5.21

$$\begin{array}{rcl} \mathsf{0} & \equiv & \emptyset \\ \mathsf{S}(t) & \equiv & \langle \emptyset, t \rangle \\ \mathsf{n} & \equiv & \underbrace{\mathsf{S}(\cdots \mathsf{S}(\mathsf{S}}_{n\ times}(\mathsf{0}))\cdots) \end{array}$$

This definition yields the desired properties:

Proposition 5.22 *The following are provable in* **LST***:*

1. $\mathsf{S}(t) \neq \mathsf{0}$.
2. $\mathsf{S}(t) = \mathsf{S}(u) \multimapboth t = u$.

Proof.

1. $\mathsf{S}(t) = \mathsf{0}$ implies $\langle \emptyset, t \rangle = \emptyset$. But $\{\emptyset\} \in \langle \emptyset, t \rangle$ whereas $\{\emptyset\} \notin \emptyset$, a contradiction.
2. $t = u \multimap \mathsf{S}(t) = \mathsf{S}(u)$ by Proposition 5.10 (2), while $\mathsf{S}(t) = \mathsf{S}(u) \multimap t = u$ by Proposition 5.14 (4).

■

Next we define the *set* of natural numbers in **LST**:

Definition 5.23

$$\mathsf{N} \equiv \{x | \forall \alpha . !\forall y (y \in \alpha \multimap \mathsf{S}(y) \in \alpha) \multimap \S(\mathsf{0} \in \alpha \multimap x \in \alpha)\}.$$

The term N surely represents the set of natural numbers in the usual sense:

Proposition 5.24

1. $\mathsf{0} \in \mathsf{N}$ *is provable in* **LST**.
2. $t \in \mathsf{N} \multimap \mathsf{S}(t) \in \mathsf{N}$ *is provable in* **LST**.
3. $t \in \mathsf{N}$ *is provable in* **LST** *if and only if* t *is a numeral* n.

Proof.
1.

$$\begin{array}{c} \mathsf{0} \in \alpha \vdash \mathsf{0} \in \alpha \\ \hline \vdash \mathsf{0} \in \alpha \multimap \mathsf{0} \in \alpha \\ \hline \vdash \S(\mathsf{0} \in \alpha \multimap \mathsf{0} \in \alpha) \\ \hline !\forall y (y \in \alpha \multimap \mathsf{S}(y) \in \alpha) \vdash \S(\mathsf{0} \in \alpha \multimap \mathsf{0} \in \alpha) \\ \hline \vdash !\forall y (y \in \alpha \multimap \mathsf{S}(y) \in \alpha) \multimap \S(\mathsf{0} \in \alpha \multimap \mathsf{0} \in \alpha) \\ \hline \vdash \forall \alpha . !\forall y (y \in \alpha \multimap \mathsf{S}(y) \in \alpha) \multimap \S(\mathsf{0} \in \alpha \multimap \mathsf{0} \in \alpha) \\ \hline \vdash \mathsf{0} \in \mathsf{N} \end{array}$$

2.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{t \in \alpha \vdash t \in \alpha \quad \mathsf{S}(t) \in \alpha \vdash \mathsf{S}(t) \in \alpha}{t \in \alpha \multimap \mathsf{S}(t) \in \alpha, t \in \alpha \vdash \mathsf{S}(t) \in \alpha}}{\forall y(y \in \alpha \multimap \mathsf{S}(y) \in \alpha), t \in \alpha \vdash \mathsf{S}(t) \in \alpha}}{\mathsf{0} \in \alpha, \forall y(y \in \alpha \multimap \mathsf{S}(y) \in \alpha), \mathsf{0} \in \alpha \multimap t \in \alpha \vdash \mathsf{S}(t) \in \alpha}}{\forall y(y \in \alpha \multimap \mathsf{S}(y) \in \alpha), \mathsf{0} \in \alpha \multimap t \in \alpha \vdash \mathsf{0} \in \alpha \multimap \mathsf{S}(t) \in \alpha}}{!\forall y(y \in \alpha \multimap \mathsf{S}(y) \in \alpha), \S(\mathsf{0} \in \alpha \multimap t \in \alpha) \vdash \S(\mathsf{0} \in \alpha \multimap \mathsf{S}(t) \in \alpha)}}{!\forall y(y \in \alpha \multimap \mathsf{S}(y) \in \alpha)^2, !\forall y(y \in \alpha \multimap \mathsf{S}(y) \in \alpha) \multimap \S(\mathsf{0} \in \alpha \multimap t \in \alpha) \vdash \S(\mathsf{0} \in \alpha \multimap \mathsf{S}(t) \in \alpha)}}{!\forall y(y \in \alpha \multimap \mathsf{S}(y) \in \alpha) \multimap \S(\mathsf{0} \in \alpha \multimap t \in \alpha) \vdash !\forall y(y \in \alpha \multimap \mathsf{S}(y) \in \alpha) \multimap \S(\mathsf{0} \in \alpha \multimap \mathsf{S}(t) \in \alpha)}}{\forall \alpha.!\forall y(y \in \alpha \multimap \mathsf{S}(y) \in \alpha) \multimap \S(\mathsf{0} \in \alpha \multimap t \in \alpha) \vdash \forall \alpha.!\forall y(y \in \alpha \multimap \mathsf{S}(y) \in \alpha) \multimap \S(\mathsf{0} \in \alpha \multimap \mathsf{S}(t) \in \alpha)}}{t \in \mathsf{N} \vdash \mathsf{S}(t) \in \mathsf{N}}$$

3. The "if" direction follows from 1 and 2 above. As for the "only-if" direction, observe that the last part of the cut-free proof of $t \in \mathsf{N}$ must be of the following form:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\mathsf{0} \in \alpha, \forall y(y \in \alpha \multimap \mathsf{S}(y) \in \alpha)^n \vdash t \in \alpha}}{\forall y(y \in \alpha \multimap \mathsf{S}(y) \in \alpha)^n \vdash \mathsf{0} \in \alpha \multimap t \in \alpha}}{!\forall y(y \in \alpha \multimap \mathsf{S}(y) \in \alpha) \vdash \S(\mathsf{0} \in \alpha \multimap t \in \alpha)}}{\vdash !\forall y(y \in \alpha \multimap \mathsf{S}(y) \in \alpha) \multimap \S(\mathsf{0} \in \alpha \multimap t \in \alpha)}}{\dfrac{\vdash \forall \alpha.!\forall y(y \in \alpha \multimap \mathsf{S}(y) \in \alpha) \multimap \S(\mathsf{0} \in \alpha \multimap t \in \alpha)}{\vdash t \in \mathsf{N}}}$$

for some $n \geq \mathsf{0}$. From this, we conclude that $t \equiv \mathsf{m}$ for some $m \leq n$. ■

5.3.2 Induction

With N defined above, a certain restricted form of induction is available:

Proposition 5.25 *The following inference rule, called Light Induction, is derivable in* **LST***:*

$$\dfrac{\Gamma \vdash A[\mathsf{0}/x] \quad B, A[y/x] \vdash A[\mathsf{S}(y)/x]}{\S\Gamma, !B, t \in \mathsf{N} \vdash \S A[t/x]},$$

where y does not occur in A and B, and B may be absent.

Proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{B, A[y/x] \vdash A[\mathsf{S}(y)/x]}{B, y \in \{x|A\} \vdash \mathsf{S}(y) \in \{x|A\}}}{B \vdash \forall y(y \in \{x|A\} \multimap \mathsf{S}(y) \in \{x|A\})}}{!B \vdash !\forall y(y \in \{x|A\} \multimap \mathsf{S}(y) \in \{x|A\})} \quad \dfrac{\dfrac{\dfrac{\Gamma \vdash A[\mathsf{0}/x]}{\Gamma \vdash \mathsf{0} \in \{x|A\}} \quad \dfrac{A[t/x] \vdash A[t/x]}{t \in \{x|A\} \vdash A[t/x]}}{\Gamma, \mathsf{0} \in \{x|A\} \multimap t \in \{x|A\}, \vdash A[t/x]}}{\S\Gamma, \S(\mathsf{0} \in \{x|A\} \multimap t \in \{x|A\}), \vdash \S A[t/x]}}{\S\Gamma, !B, !\forall y(y \in \{x|A\} \multimap \mathsf{S}(y) \in \{x|A\}) \multimap \S(\mathsf{0} \in \{x|A\} \multimap t \in \{x|A\}), \vdash \S A[t/x]}}{\S\Gamma, !B, \forall \alpha.!\forall y(y \in \alpha \multimap \mathsf{S}(y) \in \alpha) \multimap \S(\mathsf{0} \in \alpha \multimap t \in \alpha), \vdash \S A[t/x]}}{\S\Gamma, !B, t \in \mathsf{N} \vdash \S A[t/x]}$$

■

In what follows, we are particularly interested in sequents of the form $\vec{u} \in \mathsf{N} \vdash \S^p A$ ($p \geq 0$), where $\vec{u} \in \mathsf{N}$ stands for a sequence of the form $u_1 \in \mathsf{N}, \dots, u_n \in \mathsf{N}$. For such sequents, the following useful principles are available:

Proposition 5.26

1. *(Coercion)* $t \in \mathsf{N} \multimap \S^p !^q t \in \mathsf{N}$ *is provable for any* $p \geq 1$ *and* $q \geq 0$.
2. *(*N*-Contraction) The following inference rule is derivable in* **LST***:*
$$\frac{t \in \mathsf{N}, t \in \mathsf{N}, \vec{u} \in \mathsf{N} \vdash \S^p A}{t \in \mathsf{N}, \vec{u} \in \mathsf{N} \vdash \S^{p+1} A}$$
3. *(Lifting) The following inference rule is derivable in* **LST***:*
$$\frac{\vec{u} \in \mathsf{N} \vdash \S^p A}{\vec{u} \in \mathsf{N} \vdash \S^{p+q} A} \text{ for any } q \geq 0.$$

Proof.

1. For any $p \geq 1$ and $q \geq 0$, we have $\vdash \S^{p-1} !^q \mathsf{0} \in \mathsf{N}$ and $\S^{p-1} !^q x \in \mathsf{N} \vdash \S^{p-1} !^q \mathsf{S}(x) \in \mathsf{N}$. Hence the desired formula is obtained by Light Induction.
2. We have $\vdash \mathsf{0} \in \mathsf{N} \otimes \mathsf{0} \in \mathsf{N}$ and $x \in \mathsf{N} \otimes x \in \mathsf{N} \vdash \mathsf{S}(x) \in \mathsf{N} \otimes \mathsf{S}(x) \in \mathsf{N}$. Hence by Light Induction, we obtain $t \in \mathsf{N} \vdash \S(t \in \mathsf{N} \otimes t \in \mathsf{N})$. On the other hand, we have
$$\S(t \in \mathsf{N} \otimes t \in \mathsf{N}), \S\vec{u} \in \mathsf{N} \vdash \S^{p+1} A$$
by assumption. By (Cut) and Coercion $u_i \in \mathsf{N} \vdash \S u_i \in \mathsf{N}$ for each $u_i \in \{\vec{u}\}$, we obtain the desired sequent.
3. Apply rule ($\S$) q times, then apply (Cut) with Coercion $u_i \in \mathsf{N} \vdash \S^q u_i \in \mathsf{N}$ for each $u_i \in \{\vec{u}\}$.

■

5.3.3 Addition and Multiplication

The graphs of addition and multiplication are defined by fixpoint:

Definition 5.27 Let plus be a term which satisfies
$$\langle x, y, z \rangle \in \mathsf{plus} \multimapboth (y = \mathsf{0} \otimes x = z) \oplus \exists y' \exists z' (y = \mathsf{S}(y') \otimes z = \mathsf{S}(z') \otimes \langle x, y', z' \rangle \in \mathsf{plus}).$$
Such a term exists by the fixpoint theorem. Similarly, let mult be a term which satisfies
$$\langle x, y, z \rangle \in \mathsf{mult} \multimapboth (y = \mathsf{0} \otimes z = \mathsf{0}) \oplus \exists y' \exists z' (y = \mathsf{S}(y') \otimes \langle z', x, z \rangle \in \mathsf{plus} \otimes \langle x, y', z' \rangle \in \mathsf{mult}).$$

Lemma 5.28

1. $\langle x, \mathsf{0}, z\rangle \in \mathsf{plus} \multimap\!\!\circ\; x = z$.

2. $\langle x, \mathsf{S}(y), z\rangle \in \mathsf{plus} \multimap\!\!\circ\; \exists z'(z = \mathsf{S}(z') \otimes \langle x, y, z'\rangle \in \mathsf{plus})$.

3. $\langle x, \mathsf{0}, z\rangle \in \mathsf{mult} \multimap\!\!\circ\; z = \mathsf{0}$.

4. $\langle x, \mathsf{S}(y), z\rangle \in \mathsf{mult} \multimap\!\!\circ\; \exists z'(\langle z', x, z\rangle \in \mathsf{plus} \otimes \langle x, y, z'\rangle \in \mathsf{mult})$.

Proof.
1. We have $x = z \vdash \mathsf{0} = \mathsf{0} \otimes x = z$, hence $x = z \vdash \langle x, \mathsf{0}, z\rangle \in \mathsf{plus}$ is provable. On the other hand, we have $\mathsf{0} = \mathsf{0} \otimes x = z \vdash x = z$ and

$$\exists y'\exists z'(\mathsf{0} = \mathsf{S}(y') \otimes z = \mathsf{S}(z') \otimes \langle x, y', z'\rangle \in \mathsf{plus}) \vdash x = z.$$

(To see the latter, observe that $\mathsf{0} = \mathsf{S}(y')$ implies $\mathbf{0}$ by Proposition 5.22 (2), hence also implies $x = z$.) Therefore, $\langle x, \mathsf{0}, z\rangle \in \mathsf{plus} \vdash x = z$ is provable.

2. We have $z = \mathsf{S}(z') \otimes \langle x, y, z'\rangle \in \mathsf{plus} \vdash \mathsf{S}(y) = \mathsf{S}(y) \otimes z = \mathsf{S}(z') \otimes \langle x, y, z'\rangle \in \mathsf{plus}$, hence

$$\exists z'(z = \mathsf{S}(z') \otimes \langle x, y, z'\rangle \in \mathsf{plus}) \vdash \exists y'\exists z'(\mathsf{S}(y) = \mathsf{S}(y') \otimes z = \mathsf{S}(z') \otimes \langle x, y', z'\rangle \in \mathsf{plus}).$$

Therefore

$$\exists z'(z = \mathsf{S}(z') \otimes \langle x, y, z'\rangle \in \mathsf{plus}) \vdash \langle x, \mathsf{S}(y), z\rangle \in \mathsf{plus}$$

is provable. On the other hand, we have

$$\mathsf{S}(y) = \mathsf{0} \otimes x = z \vdash \exists z'(z = \mathsf{S}(z') \otimes \langle x, y, z'\rangle \in \mathsf{plus}) \tag{5.1}$$

(since $\mathsf{S}(y) = \mathsf{0} \vdash \mathbf{0}$) and

$$y = y' \otimes z = \mathsf{S}(z') \otimes \langle x, y', z'\rangle \in \mathsf{plus} \vdash z = \mathsf{S}(z') \otimes \langle x, y, z'\rangle \in \mathsf{plus},$$

(by Proposition 5.10 (2)). From the latter, we obtain

$$\mathsf{S}(y) = \mathsf{S}(y') \otimes z = \mathsf{S}(z') \otimes \langle x, y', z'\rangle \in \mathsf{plus} \vdash z = \mathsf{S}(z') \otimes \langle x, y, z'\rangle \in \mathsf{plus},$$

by Proposition 5.22 (4). Therefore

$$\exists y'\exists z'(\mathsf{S}(y) = \mathsf{S}(y') \otimes z = \mathsf{S}(z') \otimes \langle x, y', z'\rangle \in \mathsf{plus}) \vdash \exists z'(z = \mathsf{S}(z') \otimes \langle x, y, z'\rangle \in \mathsf{plus}) \tag{5.2}$$

is provable. From (5.1) and (5.2), we obtain

$$\langle x, \mathsf{S}(y), z\rangle \in \mathsf{plus} \vdash \exists z'(z = \mathsf{S}(z') \otimes \langle x, y, z'\rangle \in \mathsf{plus}).$$

3 and 4 are similarly shown. ■

The following two propositions show that addition and multiplication are provably total in **LST**.

Proposition 5.29

1. $\langle \mathsf{n}, \mathsf{m}, \mathsf{k}\rangle \in \mathsf{plus}$ *is provable in* **LST** *if* $n + m = k$.

2. $\langle \mathsf{n}, \mathsf{m}, \mathsf{k} \rangle \in \mathsf{mult}$ *is provable in* **LST** *if* $n \cdot m = k$.

Proof. Both are proved by (external) induction on m. ■

Proposition 5.30 *The following are provable in* **LST**:

1. $\forall x \in \mathsf{N}.\forall y \in \mathsf{N}.\S\exists^! z \in \mathsf{N}(\langle x, y, z\rangle \in \mathsf{plus})$.
2. $\forall x \in \mathsf{N}.\forall y \in \mathsf{N}.\S^3\exists^! z \in \mathsf{N}(\langle x, y, z\rangle \in \mathsf{mult})$.

Proof.
1. We prove

(i) $\vdash \forall x \in \mathsf{N}.\exists^! z \in \mathsf{N}(\langle x, 0, z\rangle \in \mathsf{plus})$ and

(ii) $\forall x \in \mathsf{N}.\exists^! z \in \mathsf{N}(\langle x, y, z\rangle \in \mathsf{plus}) \vdash \forall x \in \mathsf{N}.\exists^! z \in \mathsf{N}(\langle x, \mathsf{S}(y), z\rangle \in \mathsf{plus})$.

It then follows by Light Induction that

$$y \in \mathsf{N} \vdash \S(\forall x \in \mathsf{N}.\exists^! z \in \mathsf{N}(\langle x, y, z\rangle \in \mathsf{plus})). \tag{5.3}$$

By an easy manipulation, we obtain

$$\S(x \in \mathsf{N}), y \in \mathsf{N} \vdash \S\exists^! z \in \mathsf{N}(\langle x, y, z\rangle \in \mathsf{plus}),$$

hence by Coercion

$$x \in \mathsf{N}, y \in \mathsf{N} \vdash \S\exists^! z \in \mathsf{N}(\langle x, y, z\rangle \in \mathsf{plus}),$$

as required.

Now let us show (i). By Lemma 5.28(1),

$$\langle x, 0, x\rangle \in \mathsf{plus} \otimes \forall y(\langle x, 0, y\rangle \in \mathsf{plus} \multimap y = x)$$

is provable. From this,

$$x \in \mathsf{N} \vdash \exists^! z \in \mathsf{N}(\langle x, 0, z\rangle \in \mathsf{plus}),$$

i.e., (i) is provable.

As for (ii), argue informally as follows. Assume that $x \in \mathsf{N}$ and $\exists^! z \in \mathsf{N}(\langle x, y, z\rangle \in \mathsf{plus})$, i.e., $z \in \mathsf{N}$ be the unique element such that $\langle x, y, z\rangle \in \mathsf{plus}$. Then it follows that

$$\mathsf{S}(z) \in \mathsf{N} \tag{5.4}$$

by Proposition 5.24 (2), and

$$\langle x, \mathsf{S}(y), \mathsf{S}(z)\rangle \in \mathsf{N} \tag{5.5}$$

by Lemma 5.28 (2). It remains to show that $\mathsf{S}(z)$ is the unique element satisfying (5.5). So assume that $\langle x, \mathsf{S}(y), w\rangle \in \mathsf{N}$. Then by Lemma 5.28 (2), there exists w' such that $w = \mathsf{S}(w')$ and $\langle x, y, w'\rangle \in \mathsf{plus}$. By the uniqueness of z, we have $w' = z$. Therefore $w = \mathsf{S}(z)$ as required. It is easy to check that this informal proof can be formalized in **LST**.

2. We prove

(iii) $\vdash \S\exists^! w \in \mathsf{N}(\langle x, \mathsf{0}, z\rangle \in \mathsf{mult})$ and

(iv) $x \in \mathsf{N}, \S\exists^! z \in \mathsf{N}(\langle x, y, z\rangle \in \mathsf{mult}) \vdash \S\exists^! w \in \mathsf{N}(\langle x, \mathsf{S}(y), w\rangle \in \mathsf{mult})$.

Then by Light Induction,

$$!x \in \mathsf{N}, y \in \mathsf{N} \vdash \S^2\exists^! z \in \mathsf{N}(\langle x, y, z\rangle \in \mathsf{mult})$$

is provable, hence by Coercion, we obtain the desired formula.

As for (iii), observe that

$$\vdash \mathsf{0} \in \mathsf{N} \otimes \langle x, \mathsf{0}, \mathsf{0}\rangle \in \mathsf{mult} \otimes \forall w'(\langle x, \mathsf{0}, w'\rangle \in \mathsf{mult} \multimap w' = \mathsf{0})$$

is provable by Proposition 5.24 (1) and Lemma 5.28 (3).

As for (iv), first show

$$\langle x, y, z\rangle \in \mathsf{mult}, \langle z, x, w\rangle \in \mathsf{plus} \vdash \langle x, \mathsf{S}(y), w\rangle \in mult$$

and

$$\forall z'(\langle x, y, z'\rangle \in \mathsf{mult} \multimap z = z'), \forall w'(\langle z, x, w'\rangle \in \mathsf{plus} \multimap w = w') \vdash \forall w'(\langle x, \mathsf{S}(y), w'\rangle \in \mathsf{mult} \multimap w = w')$$

by using Lemma 5.28 (4). From these two, we successively derive:

$$\begin{array}{c}
\langle x, y, z\rangle \in \mathsf{mult} \otimes \forall z'(\langle x, y, z'\rangle \in \mathsf{mult} \multimap z = z'), \exists^! w \in \mathsf{N}(\langle z, x, w\rangle \in \mathsf{plus}) \vdash \exists^! w \in \mathsf{N}(\langle x, \mathsf{S}(y), w\rangle \in \mathsf{mult}) \\
\hline\hline
\exists^! z \in \mathsf{N}(\langle x, y, z\rangle \in \mathsf{mult}), \forall z \in \mathsf{N}.\exists^! w \in \mathsf{N}(\langle z, x, w\rangle \in \mathsf{plus}) \vdash \exists^! w \in \mathsf{N}(\langle x, \mathsf{S}(y), w\rangle \in \mathsf{mult}) \\
\hline
\S\exists^! z \in \mathsf{N}(\langle x, y, z\rangle \in \mathsf{mult}), \S\forall z \in \mathsf{N}.\exists^! w \in \mathsf{N}(\langle z, x, w\rangle \in \mathsf{plus}) \vdash \S\exists^! w \in \mathsf{N}(\langle x, \mathsf{S}(y), w\rangle \in \mathsf{mult})
\end{array} .$$

Therefore, we obtain the desired sequent by (5.3). ■

5.4 Representing Sets and Functions

In this section, we shall demonstrate that various sets and functions required for encoding Turing machines are representable in **LST**.

5.4.1 Representation in LST

We first need to make precise what it means to represent sets and functions in **LST**.

Definition 5.31

1. A set T is *represented by* a term t of **LST** if there is a bijection $(\cdot)^*$ from T to the set of terms u such that $\vdash u \in t$ is provable in **LST**.
2. A function $F : \vec{T} \longrightarrow U$ is *represented by* a term f with domains $\vec{t}$ and codomain u if
   - $\vec{T}$ and U are represented by $\vec{t}$ and u respectively;
   - For any $\vec{m} \in \vec{T}$ and $n \in U$ such that $F(\vec{m}) = u$, $\vdash \langle \vec{m}^*, n^*\rangle \in f$ is provable in **LST**;

- $\vdash \forall\vec{x} \in t.\S^d\exists^! y \in u(\langle\vec{x}, y\rangle \in f)$ is provable in **LST** for some $d \geq 0$, where $\forall\vec{x} \in t.A$ stands for $\forall x_1 \in t \ldots \forall x_n \in t.A$.

In particular, we say that f is *flatly represented by* f in case $d = 0$. A representable function is also said to be *provably total in* **LST**, following the standard terminology.

Note that the notion of representability in the above sense is stronger than that of numeralwise representability studied in Section 5.2.5 in that the totality of a function must be *internally* provable in the formal system **LST**. We have already seen that the term N represents the set N of natural numbers, plus and mult represent addition and multiplication of natural numbers; it is also clear that the term $\{x|\exists y(x = \langle y, \mathsf{S}(y)\rangle)\}$ represents successor. Further examples of representations are provided below.

The uniqueness property incorporated in the third condition of Definition 5.31(2) implies the following:

Proposition 5.32 *Suppose that a function $F : \vec{T} \longrightarrow U$ be represented by a term f with domains $\vec{t}$ and codomain u. Let $\vec{m} \in \vec{T}$. Then $\langle\vec{m}^*, v\rangle \in f$ is provable in* **LST** *iff $v \equiv n^*$, where $n = F(\vec{m})$.*

Proof. The "if" direction holds by definition. To show the "only-if" direction, observe that

$$\vdash \S^d\exists^! y \in u(\langle\vec{m}^*, y\rangle \in f)$$

is provable in **LST** for any $\vec{m} \in \vec{T}$. By Corollary 5.6, it follows that

$$\forall y(\langle\vec{m}^*, y\rangle \in f \multimap y = v_0)) \tag{5.6}$$

is provable for some v_0. Now suppose that $\langle\vec{m}^*, v\rangle \in f$ be provable. Then we have $v = v_0$ by (5.6). Suppose also that $n = F(\vec{m})$. Then by definition, $\langle\vec{m}^*, n^*\rangle \in f$ is provable. Hence we have $n^* = v_0$ by (5.6). Hence $v = v_0 = n^*$ is provable in **LST**. Therefore $v \equiv n^*$ by Proposition 5.9. ■

5.4.2 Finite Sets

A finite set Q_n with n elements is represented by the term

$$\mathsf{Q}_n \equiv \{\mathsf{0}, \ldots, \mathsf{n-1}\}$$

with the obvious bijection. In particular, the set Q_2 of boolean values is represented by Q_2. This definition yields flat Contraction

$$t \in \mathsf{Q}_n \multimap (t \in \mathsf{Q}_n \otimes t \in \mathsf{Q}_n)$$

and Coercion

$$t \in \mathsf{Q}_n \multimap \S^d !^e t \in \mathsf{Q}_n \quad \text{for any } d \geq 1 \text{ and } e \geq 0.$$

Moreover, we have:

Proposition 5.33 *Every finite function $F : Q_{n_1} \times \cdots \times Q_{n_k} \longrightarrow Q_m$ is flatly representable with domains $\mathsf{Q}_{n_1}, \ldots, \mathsf{Q}_{n_k}$ and codomain Q_m.*

Proof. Instead of giving a detailed proof, we just describe an example, which should be sufficient for convincing ourselves. We shall show that boolean conjunction $\wedge : Q_2 \times Q_2 \longrightarrow Q_2$ is represented by

$$\mathsf{conj} \equiv \{x|(x = \langle 0,0,0\rangle) \oplus (x = \langle 0,1,0\rangle) \oplus (x = \langle 1,0,0\rangle) \oplus (x = \langle 1,1,1\rangle)\}.$$

The first and the second conditions for representability are immediate. As for the third condition, prove

$$\vdash 0 \in \mathsf{Q}_2 \otimes \langle 0,0,0\rangle \in \mathsf{conj} \otimes \forall z(\langle 0,0,z\rangle \in \mathsf{conj} \multimap z = 0),$$

from which we obtain

$$x = 0, y = 0 \vdash \exists^! z \in \mathsf{Q}_2(\langle x,y,z\rangle \in \mathsf{conj}).$$

Similarly, we can prove

$$\begin{array}{l} x = 1, y = 1 \vdash \exists^! z \in \mathsf{Q}_2(\langle x,y,z\rangle \in \mathsf{conj}), \\ x = 1, y = 0 \vdash \exists^! z \in \mathsf{Q}_2(\langle x,y,z\rangle \in \mathsf{conj}), \\ x = 1, y = 1 \vdash \exists^! z \in \mathsf{Q}_2(\langle x,y,z\rangle \in \mathsf{conj}). \end{array}$$

Therefore, we obtain

$$x \in \mathsf{Q}_2, y \in \mathsf{Q}_2 \vdash \exists^! z \in \mathsf{Q}_2(\langle x,y,z\rangle \in \mathsf{conj}),$$

by noting that $x \in \mathsf{Q}_2 \multimapboth x = 0 \oplus x = 1$. ■

As a generalization, we also have:

Proposition 5.34 *Given functions*

$$H_i \quad : \quad \vec{T} \longrightarrow U, \text{ for each } 0 \leq i \leq n-1,$$

one defines a new function $F : Q_n \times \vec{T} \longrightarrow U$ by:

$$F(i, \vec{v}) \quad = \quad H_i(\vec{v}).$$

Suppose that $\vec{T}$ and U be represented by terms $\vec{t}$ and u, and H_i's be flatly representable. Then the above F is flatly representable with domains $\mathsf{Q}_n, \vec{t}$ and codomain u.

Proof. Let H_i be represented by term h_i for $0 \leq i \leq n-1$. Define

$$f \equiv \{x|(\exists\langle\vec{y}, z\rangle \in h_0.x = \langle 0, \vec{y}, z\rangle) \oplus \cdots (\exists\langle\vec{y}, z\rangle \in h_{n-1}.x = \langle \mathsf{n}-1, \vec{y}, z\rangle).$$

Then this term f flatly represents F. ■

5.4.3 Words over Finite Alphabets 1

Let us consider the set W_n of words over a finite alphabet Q_n. First, define

$$\begin{array}{rcl} \epsilon & \equiv & \emptyset; \\ \mathsf{S}_i(t) & \equiv & \langle \mathsf{i}, t\rangle, \text{ for each } 0 \geq i < n. \end{array}$$

Then we can naturally associate a term w to each word $w \in W_n$. For example we associate to $010 \in W_2$ the term $\mathsf{S}_0(\mathsf{S}_1(\mathsf{S}_0(\epsilon)))$.

There are two ways of representing the set W_n, both of which are useful. The first is a generalization of N. Define W_2 by

$$\mathsf{W}_2 \;\equiv\; \{x|\forall\alpha.!\forall y(y \in \alpha \multimap \mathsf{S}_0(y) \in \alpha)\multimap !\forall y(y \in \alpha \multimap \mathsf{S}_1(y) \in \alpha) \multimap \S(\epsilon \in \alpha \multimap x \in \alpha)\}.$$

More generally, define for each n

$$\mathsf{W}_n \;\equiv\; \{x|\forall\alpha.!\forall y(y \in \alpha \multimap \mathsf{S}_0(y) \in \alpha) \multimap \cdots !\forall y(y \in \alpha \multimap \mathsf{S}_{n-1}(y) \in \alpha) \multimap \S(\epsilon \in \alpha \multimap x \in \alpha)\}.$$

In what follows, W_2 and W_2 are often abbreviated as W and W. The second representation of W_n will be given in the next subsection.

Proposition 5.35

1. *$\epsilon \in \mathsf{W}_n$ is provable in* **LST**.
2. *$t \in \mathsf{W}_n \multimap \mathsf{S}_i(t) \in \mathsf{W}_n$ is provable in* **LST** *for each $0 \leq i < n$.*
3. *$t \in \mathsf{W}_n$ is provable in* **LST** *if and only if t is of the form w for some $w \in W_n$.*

Proof. Similarly to the proof of Proposition 5.24. ∎

Like N, W_n admits Light Induction, hence it also admits Coercion, Contraction and Lifting:

Proposition 5.36 *The following inference rule is derivable in* **LST**:

$$\frac{\Gamma \vdash A[0/x] \quad B_0, A[y/x] \vdash A[\mathsf{S}_0(y)/x] \quad \cdots \quad B_{n-1}, A[y/x] \vdash A[\mathsf{S}_{n-1}(y)/x]}{\S\Gamma, !B_0, \ldots, !B_{n-1}, t \in \mathsf{W}_n \vdash \S A[t/x]}$$

where y does not occur in A and $B_0, \ldots, B_{n-1}$.

Proposition 5.37

1. *(Coercion) $t \in \mathsf{W}_n \multimap \S^p !^q t \in \mathsf{W}_n$ is provable for any $p \geq 1$ and $q \geq 0$.*
2. *(W_n-Contraction) The following inference rule is derivable in* **LST**:
$$\frac{t \in \mathsf{W}_n, t \in \mathsf{W}_n, \vec{u} \in \mathsf{W}_n \vdash \S^p A}{t \in \mathsf{W}_n, \vec{u} \in \mathsf{W}_n \vdash \S^{p+1} A}$$
3. *(Lifting) The following inference rule is derivable in* **LST**:
$$\frac{\vec{u} \in \mathsf{W}_n \vdash \S^p A}{\vec{u} \in \mathsf{W}_n \vdash \S^{p+q} A} \text{ for any } q \geq 0.$$

Proof. Similarly to the proofs of Propositions 5.25 and 5.26. ∎

Proposition 5.38 *The length map $|\bullet| \; : W_n \longrightarrow N$ such that*

$$|w| = \text{the length of } w$$

is representable with domain W_n and codomain N.

Proof. We consider the case $n = 2$. Define the term len by:

$$\langle x, y\rangle \in \mathsf{len} \multimapboth (x = \epsilon \otimes y = 0) \oplus \exists x' \exists y'((x = \mathsf{S}_0(x') \oplus x = \mathsf{S}_1(x')) \otimes y = \mathsf{S}(y') \otimes \langle x', y'\rangle \in \mathsf{len}).$$

Then we easily see that

(i) $\langle \epsilon, y\rangle \in \mathsf{len} \multimapboth y = 0$ and

(ii) $\langle \mathsf{S}_i(x), y\rangle \in \mathsf{len} \multimapboth \exists y'(y = \mathsf{S}(y') \otimes \langle x, y'\rangle \in \mathsf{len}$.

From these two,

(iii) $\langle w, m\rangle \in \mathsf{len}$ if $|w| = m$,

(iv) $\vdash \exists^! y \in \mathsf{N}(\langle \epsilon, y\rangle \in \mathsf{len})$,

(v) $\exists^! y \in \mathsf{N}(\langle x, y\rangle \in \mathsf{len}) \vdash \exists^! y \in \mathsf{N}(\langle \mathsf{S}_i(x), y\rangle \in \mathsf{len})$ for $i = 0, 1$.

From the last two, by Light Induction for W_2 we obtain

$$\forall x \in \mathsf{W}_2. \S \exists^! y \in \mathsf{N}(\langle x, y\rangle \in \mathsf{len})$$

as desired. ■

Proposition 5.39 *Let $n \leq m$. Then $t \in \mathsf{W}_n \vdash t \in \mathsf{W}_m$ is provable in* **LST**. *Therefore the inclusion map $In : W_n \longrightarrow W_m$ is flatly representable with domain W_n and codomain W_m.*

Proof. Proving $t \in \mathsf{W}_n \vdash t \in \mathsf{W}_m$ is easy. The term $\{x | \exists y. x = \langle y, y\rangle$ represents the inclusion map. ■

5.4.4 Words over Finite Alphabets 2

The second representation of W_n is given by fixpoint:

$$x \in \mathsf{W}'_n \multimapboth x = \epsilon \oplus \exists x' \in \mathsf{W}'_n(x = \mathsf{S}_0(x') \oplus \cdots \oplus x = \mathsf{S}_{n-1}(x')).$$

Proposition 5.40

1. *$\epsilon \in \mathsf{W}'_n$ is provable in* **LST**.
2. *$t \in \mathsf{W}'_n \multimap \mathsf{S}_i(t) \in \mathsf{W}'_n$ is provable in* **LST** *for each $0 \leq i < n$.*
3. *$t \in \mathsf{W}'_n$ is provable in* **LST** *if and only if t is of the form w for some $w \in W_n$.*

Proof. 1 and 2 are easily proved. As for 3, the "if" direction follows from 1 and 2. The "only-if" direction is proved by induction on the size (i.e., the number of symbols) of t. Assume that $t \in \mathsf{W}'_n$ is provable in **LST**. If $t \equiv \epsilon$, then there is nothing to prove. Otherwise, by Corollary 5.6, there is a term t' such that $t = \mathsf{S}_i(t')$ and $t' \in \mathsf{W}'_n$ are provable for some $i < n$. By Proposition 5.9, t and $\mathsf{S}_i(t')$ are syntactically identical. Hence t' has a smaller number of symbols than t. By the induction hypothesis, $t' \equiv \mathsf{w}$ for some $w \in W_n$. Therefore $t \equiv \mathsf{S}_i(\mathsf{w})$. ■

W'_n does not admit Induction. Instead, it allows us to flatly represent the following discriminator function:

Proposition 5.41 *Given functions*

$$\begin{aligned} H_\epsilon &: \vec{T} \longrightarrow U; \\ H_i &: W_n \times \vec{T} \longrightarrow U, \text{ for each } 0 \leq i \leq n-1, \end{aligned}$$

one defines a new function $F : W_n \times \vec{T} \longrightarrow U$, called discriminator*, by:*

$$\begin{aligned} F(\epsilon, \vec{v}) &= H_\epsilon(\vec{v}); \\ F(i \cdot w, \vec{v}) &= H_i(w, \vec{v}). \end{aligned}$$

Suppose that $\vec{T}$ and U be represented by terms $\vec{t}$ and u, and H_ϵ and H_i's be flatly represented by h and h_i's. Then the above F is flatly representable with domains $\mathsf{W}'_n, \vec{t}$ and codomain u.

Proof. For simplicity, we consider the case $n = 2$ and assume $\vec{T} \equiv T$. Define the term f by

$\langle x, y, z\rangle \in f \multimapboth (x = \epsilon \otimes \langle y, z\rangle \in h_\epsilon) \oplus \exists x'((x = \mathsf{S}_0(x') \otimes \langle x', y, z\rangle \in h_0) \oplus (x = \mathsf{S}_1(x') \otimes \langle x', y, z\rangle \in h_1))$.

By assumption, $y \in t \vdash \exists^! z\langle y, z\rangle \in h_\epsilon$, from which we obtain

$$x = \epsilon, y \in t \vdash \exists! z(\langle x, y, z\rangle \in f). \tag{5.7}$$

And also,

$$x' \in \mathsf{W}'_2, y \in t \vdash \exists^! z(\langle x', y, z\rangle \in h_0)$$

by assumption, from which we obtain

$$x' \in \mathsf{W}'_2 \otimes x = \mathsf{S}_0(x'), y \in t \vdash \exists! z(\langle x, y, z\rangle \in f). \tag{5.8}$$

Similarly we have

$$x' \in \mathsf{W}'_2 \otimes x = \mathsf{S}_1(x'), y \in t \vdash \exists! z(\langle x, y, z\rangle \in f). \tag{5.9}$$

By (5.7), (5.8) and (5.9),

$$x \in \mathsf{W}'_2, y \in t \vdash \exists! z(\langle x, y, z\rangle \in f)$$

is provable as required. ■

As an instance of the above proposition, we have:

Proposition 5.42 *Predecessor for W_n, defined by*

$$\begin{aligned} P(\epsilon) &\equiv \epsilon; \\ P(i \cdot w) &\equiv w, \text{ for } 0 \leq i < n, \end{aligned}$$

is flatly representable with domain and codomain W'_n.

The following proposition shows, in some appropriate sense, that W_n is a "subset" of W'_n.

Proposition 5.43 *$t \in \mathsf{W}_n \vdash \S(t \in \mathsf{W}'_n)$ is provable in* **LST**. *Therefore, the identity map $Id : W_n \longrightarrow W_n$ is representable with domain W_n and codomain W'_n.*

Proof. We have $\vdash \epsilon \in \mathsf{W}'_n$ and $x \in \mathsf{W}'_n \vdash \mathsf{S}_i(x) \in \mathsf{W}'_n$ for each $0 \leq i < n$ by Proposition 5.40. Therefore, by Light Induction for W, we have $t \in \mathsf{W} \vdash \S(t \in \mathsf{W}_b)$. ■

The converse does not hold in general, however.

5.4.5 Cartesian Products

If two sets T and U are represented by terms t and u respectively, then their Cartesian product $T \times U$ is represented by the term

$$t \times u \equiv \{x | \exists y \exists z (x = \langle y, z \rangle \otimes y \in t \otimes z \in u\},$$

with the associated bijection

$$\langle x, y \rangle^* = \langle x^*, y^* \rangle.$$

Proposition 5.44 *Products of flatly representable functions are flatly representable with appropriate domains and codomain.*

Proof. Let $F_i : T_i \longrightarrow U_i$ be flatly represented by terms f_i respectively, with domains t_i and codomain u_i $(i = 1, 2)$. Then define:

$$f_1 \times f_2 \equiv \{x | \exists y_1 y_2 z_1 z_2 (x = \langle \langle y_1, y_2 \rangle, \langle z_1, z_2 \rangle \rangle \otimes \langle y_1, z_1 \rangle \in f_1 \otimes \langle y_2, z_2 \rangle \in f_2)\}.$$

This term flatly represents $F_1 \times F_2 : T_1 \times T_2 \longrightarrow U_1 \times U_2$, as required. ■

5.4.6 Composition and Iteration

Proposition 5.45 *Suppose that $\vec{T}$, $\vec{T'}$, U and V be represented by terms $\vec{t}$, $\vec{t'}$, u and v. Suppose that function $G : \vec{T} \times U \longrightarrow V$ be representable with domains $\vec{t}, u$ and codomain v, and that function $H : \vec{T'} \longrightarrow U$ be representable with domains $\vec{t'}$ and codomain u. Then the function $F(\vec{x}, \vec{z}) = G(\vec{x}, H(\vec{z}))$, where $\vec{x}$ and $\vec{z}$ are disjoint, is representable with domain $\vec{t}, \vec{t'}$ and codomain v.*

Proof. For simplicity, let us assume that $\vec{x} \equiv x$ and $\vec{z} \equiv z$. Suppose that G and H be represented by terms g and h. Define:

$$g \circ h \equiv \{x' | \exists xyzw (x' = \langle x, y, z \rangle \otimes \langle y, w \rangle \in h \otimes \langle x, w, z \rangle \in g)\}.$$

This term represents $G(x, H(z))$. ■

Proposition 5.46 *If $G : \vec{T} \longrightarrow U$ is representable and $H : U \longrightarrow U$ is* flatly *representable, then the function $F : N \times \vec{T} \longrightarrow U$, defined by iteration:*

$$\begin{array}{rcl} F(0, \vec{y}) & = & G(\vec{y}); \\ F(n+1, \vec{y}) & = & H(F(n, \vec{y})), \end{array}$$

is representable.

Proof. For simplicity, let us assume that $\vec{y} \equiv y$. Suppose that G and H be represented by terms g and h. Let f be the fixpoint:

$$\langle x, y, z \rangle \in f \multimap\!\!\circ\ (x = 0 \otimes \langle y, z \rangle \in g) \oplus \exists x' z' (x = \mathsf{S}(x') \otimes \langle x', y, z' \rangle \in f \otimes \langle z', z \rangle \in h).$$

By Light Induction, we can show that this term f surely represents F. We omit the proof, since it is quite similar to the representability of addition and multiplication (Proposition 5.30). ■

5.5 Encoding Turing Machines

In this section, we shall show that polynomial time computations over Turing machines can be simulated in **LST**. As a consequence, we shall show that every polytime function is provably total in **LST**. Our treatment of Turing machines below is essentially the same as in [Gir98, AR00]. We refer to [HU79] for the general background on Turing machines.

Let M be a single-tape Turing machine over the alphabet $\Sigma = \{0, 1, b\}$ (where b is for blank) and with the states $Q = \{q_0, \dots, q_{n-1}\}$ (where q_0 is the initial state). M is endowed with a transition function

$$\delta : \Sigma \times Q \longrightarrow \Sigma \times Q \times \{L, R, C\},$$

here L stands for left, R for right, and C for no-move. M has an infinite tape, which has infinitely many cells in both directions. A step of M consists of reading one symbol from the cell which the head is scanning, writing a symbol on the same cell, moving the head at most one tape cell, and entering a new state, in accordance with the transition function δ.

A *configuration* of M consists of a triple $\langle q, w_1, w_2\rangle \in Q \times \Sigma^* \times \Sigma^*$; q denotes the current state, w_1 describes the non-blank part of the tape to the left of the head, w_2 describes the non-blank part of the tape to the right of the head. By convention, w_1 is written in the reverse order, and w_2 includes the content of the cell currently scanned.

The *initial configuration* with input $w \in \{0, 1\}^*$ is of the form $\langle q_0, \epsilon, w\rangle$. Let M arrive at a configuration $\langle q, w_1, w_2\rangle$ after some steps. We extract the *output* of the computation from $\langle q, w_1, w_2\rangle$ in the following way. If w_2 does not contain blank b, then the output is w_2. Otherwise w_2 must be of the form $w \cdot b \cdot w'$ where w does not contain a blank symbol. In this case, the output is w.

Definition 5.47 A function $F : \{0, 1\}^* \longrightarrow \{0, 1\}^*$ is a *polynomial time function* if there is a Turing machine M and a monotone polynomial p such that starting from the initial configuration with input $w \in \{0, 1\}^*$ M outputs $F(w)$ after $p(|w|)$ steps, for every $w \in \{0, 1\}^*$.

Now we claim:

Theorem 5.48 *Every polynomial time function is provably total in* **LST** *with domain and codomain* W.

Proof. All the requisite materials are given in the previous sections. It just suffices to put them together. Let F be a polynomial time function. Then there is a Turing machine M and a polynomial p as in Definition 5.47.

- The set $Conf \equiv Q \times \Sigma^* \times \Sigma^*$ of configurations is represented by the term $\mathsf{Conf} \equiv \mathsf{Q}_n \times \mathsf{W}'_3 \times \mathsf{W}'_3$.
- The function $\hat{\delta} : Conf \longrightarrow Conf$ for the one-step transition is obtained by combining successors and predecessor for Σ^*, a case function for Q and a case function for Σ^*. Note that all of these are *flatly* representable in **LST** by Propositions 5.40, 5.42, 5.34 and 5.41. Hence $\hat{\delta}$ is also flatly representable with domain and codomain Conf by Proposition 5.44. (Here it is crucial to use W'_3 rather than W_3 to define Conf.)
- Using iteration, a function $F_M : N \times Conf \longrightarrow Conf$ is defined such that the value of $F_M(n, c)$ is the configuration of M after n steps starting from the configuration c. By Proposition 5.46, this F_M can be represented by a term of **LST** with domain $\mathsf{N} \times \mathsf{Conf}$ and codomain Conf.

- On the other hand, any monotone polynomial can be composed of addition and multiplication, and thus can be represented by a **LST** term with domain and codomain N. By composing it with the length function $|\bullet| : \{0,1\}^* \longrightarrow N$, we obtain a term representing the function $p(|w|)$ with domain W and codomain N.

- The initializing function which transforms an input $w \in \{0,1\}$ to an initial configuration is represented by a term with domain W and codomain Conf by using Propositions 5.39 and 5.43. The output function from $Conf$ to $\{0,1\}^*$ can be similarly represented with domain Conf and W'.

- Therefore, we get a term representing F with domain W and codomain W'; the input is used twice, once for initialization and once for the time bound $p(|w|)$, but it does not matter since we have W-Contraction.

- A problem is that the above F is represented with codomain W'. But the function $Id' : N \times \{0,1\}^* \longrightarrow \{0,1\}^*$ such that

  $$Id'(n, w) = \text{the lowermost } n \text{ bits of } w,$$

  is representable with domain $\mathsf{N} \times \mathsf{W}'$ and codomain W. Using this, we can obtain the desired term representing F with domain and codomain W, since the length of the output is obviously bounded by $p(|w|)$.

■

Chapter 6

Extracting Programs from Proofs of Light Set Theory

In Chapter 5, we have developed naive set theory based on **ILAL**. The main advantage of developing a mathematical theory based on a constructive logic is that we can extract a program from a proof of a mathematical proposition. In particular, if we have a proof of a proposition like "for every $x \in W$ there exists $y \in W$ such that $A(x, y)$," then from that proof we can extract a program which, given an input $m \in W$, returns an appropriate value $n \in W$ which satisfies the specification $A(m, n)$. In this chapter, we shall demonstrate the program extraction method for **LST**. This is carried out by extending the proofs-as-programs interpretation, which is detailed for **ILAL2** in Chapter 4, to **LST**. In our case, what we can extract is a term of $\lambda\textsc{la}$. Since any term of $\lambda\textsc{la}$ is polytime normalizable, the extracted program is a polytime one. Note that all polytime functions are provably total in **LST** as shown in Chapter 5. In conjunction with this result, the program extraction yields a complete characterization of the polytime functions: A function is polytime if and only if it is provably total in $\lambda\textsc{la}$.

In Section 6.1, we shall reformulate **LST** as a type assignment system for $\lambda\textsc{la}$. In Section 6.2, we shall prove the subject reduction theorem. The cut-elimination theorem for **LST** (in the strict sense) is proved in Section 6.3. In Section 6.4, we shall describe the program extraction method, and as a consequence obtain a complete characterization of the polytime functions.

6.1 LST as a Type Assignment System

As in the case of **ILAL2**, we shall reformulate the theory **LST** as a type assignment system for $\lambda\textsc{la}$. The formal definition is as follows:

Definition 6.1 The *terms* and *types* of **LST** are simultaneously defined by the following grammar:

$$\begin{array}{rcl} t, u & ::= & x \mid \{x|A\}; \\ A, B & ::= & t \in u \mid A \multimap B \mid \forall x.A \mid !A \mid \S A. \end{array}$$

Discharged types are defined as before. Note that terms of **LST** are distinguished from terms of $\lambda\textsc{la}$, although we use the same notation for both of them.

$$\frac{}{x\!:\!A \vdash x\!:\!A}\ Id \qquad \frac{\Gamma_1 \vdash u\!:\!A \quad x\!:\!A, \Gamma_2 \vdash t\!:\!C}{\Gamma_1, \Gamma_2 \vdash t[u/x]\!:\!C}\ Cut$$

$$\frac{\Gamma \vdash t\!:\!C}{\Delta, \Gamma \vdash t\!:\!C}\ Weak \qquad \frac{x\!:\![A]_!, y\!:\![A]_!, \Gamma \vdash t\!:\!C}{z\!:\![A]_!, \Gamma \vdash t[z/x, z/y]\!:\!C}\ Cntr$$

$$\frac{\Gamma_1 \vdash u\!:\!A_1 \quad x\!:\!A_2, \Gamma_2 \vdash t\!:\!C}{\Gamma_1, y\!:\!A_1 \multimap A_2, \Gamma_2 \vdash t[yu/x]\!:\!C}\ {\multimap}l \qquad \frac{x\!:\!A_1, \Gamma \vdash t\!:\!A_2}{\Gamma \vdash \lambda x.t\!:\!A_1 \multimap A_2}\ {\multimap}r$$

$$\frac{x\!:\!A[u/x], \Gamma \vdash t\!:\!C}{x\!:\!\forall x.A, \Gamma \vdash t\!:\!C}\ \forall l \qquad \frac{\Gamma \vdash t\!:\!A}{\Gamma \vdash t\!:\!\forall x.A}\ \forall r,\quad (x \text{ is not free in } \Gamma)$$

$$\frac{x\!:\!A[u/x], \Gamma \vdash t\!:\!C}{x\!:\!u \in \{x|A\}, \Gamma \vdash t\!:\!C}\ \in l \qquad \frac{\Gamma \vdash t\!:\!A[u/x]}{\Gamma \vdash t\!:\!u \in \{x|A\}}\ \in r$$

$$\frac{x\!:\![A]_!, \Gamma \vdash t\!:\!C}{y\,:\!!A, \Gamma \vdash \mathsf{let}\ y\ \mathsf{be}\ !x\ \mathsf{in}\ t\!:\!C}\ !l \qquad \frac{x\!:\!B \vdash t\!:\!A}{x\!:\![B]_! \vdash !t\,:\!!A}\ !r$$

$$\frac{x\!:\![A]_\S, \Gamma \vdash t\!:\!C}{y\!:\!\S A, \Gamma \vdash \mathsf{let}\ y\ \mathsf{be}\ \S x\ \mathsf{in}\ t\!:\!C}\ \S l \qquad \frac{\Gamma, \Delta \vdash t\!:\!A}{[\Gamma]_!, [\Delta]_\S \vdash \S t\!:\!\S A}\ \S r$$

In rule ($!r$), $x\!:\!B$ can be absent. In rule ($\S r$), Γ and Δ can be empty.

Figure 6.1: Type Assignment System **LST**

Definition 6.2 The *type inference rules* of **LST** are those given in Figure 6.1. We say that a pseudo-term t is *typable* in **LST** if $\Gamma \vdash t\!:\!A$ is derivable for some Γ and A by those inference rules.

We only have $\multimap, \forall, !$ and $\S$ as logical connectives (type constructors). The other connectives and constants of **LST** are again defined from $\multimap$ and $\forall$: Fix an arbitrary closed term t_0 of **LST** (e.g., $\{x|x \in x\}$). Then define:

$$\begin{array}{rcl}
\exists y.A & \equiv & \forall x.(\forall y.(A \multimap t_0 \in x) \multimap t_0 \in x);\\
A \otimes B & \equiv & \forall x.((A \multimap B \multimap t_0 \in x) \multimap t_0 \in x);\\
\mathbf{1} & \equiv & \forall x.(t_0 \in x \multimap t_0 \in x);\\
A \,\&\, B & \equiv & \exists x.((t_0 \in x \multimap A) \otimes (t_0 \in x \multimap B) \otimes t_0 \in x);\\
A \oplus B & \equiv & \forall x.((A \multimap t_0 \in x) \multimap (B \multimap t_0 \in x) \multimap t_0 \in x);\\
\mathbf{0} & \equiv & \forall x.t_0 \in x,
\end{array}$$

where x is a fresh variable which does not occur in A and B.

It is easy to see that these definitions are compatible with the inference rules of **LST**.

As before, typable terms are well-formed:

$$\frac{\Gamma \vdash t{:}\forall x.A}{\Gamma \vdash t{:}A\{u/x\}}\ \forall E \qquad \frac{\Gamma \vdash t{:}A \quad x \notin FV(\Gamma)}{\Gamma \vdash t{:}\forall x.A}\ \forall I$$

$$\frac{\Gamma \vdash t{:}u \in \{x|A\}}{\Gamma \vdash t{:}A[u/x]}\ \in E \qquad \frac{\Gamma \vdash t{:}A[u/x]}{\Gamma \vdash t{:}u \in \{x|A\}}\ \in I$$

Figure 6.2: Inference Rules for $\mathbf{LST}_N$

Theorem 6.3 *Every pseudo-term which is typable in* **LST** *is a term. More exactly, if* $\vec{x}{:}\vec{A}, \vec{y}{:}[\vec{B}]_!, \vec{z}{:}[\vec{C}]_\S \vdash t{:}D$, *then* $t \in \mathcal{T}_{\{\vec{x}\},\{\vec{y}\},\{\vec{z}\}}$.

Example 6.4 In the proof of Proposition 5.24 (2), we described a (formal) proof of $t \in \mathsf{N} \vdash \mathsf{S}(t) \in \mathsf{N}$. That proof can be viewed as a typing derivation for successor:

$$Suc \equiv \lambda yx.\mathsf{let}\ x\ \mathsf{be}\ !x'\ \mathsf{in}\ (\mathsf{let}\ y!x'\ \mathsf{be}\ \S y'\ \mathsf{in}\ \S(\lambda z.x'(y'z)))$$

which was introduced in Example 3.4 of Chapter 3. In fact, we have

$$\vdash Suc{:}t \in \mathsf{N} \multimap \mathsf{S}(t) \in \mathsf{N}$$

in **LST**.

6.2 Subject Reduction Theorem for LST

The subject reduction theorem can be proved for **LST** just in the same way as for **ILAL2**. To show this, we introduce a natural deduction system $\mathbf{LST}_N$, and prove the equivalence of **LST** and $\mathbf{LST}_N$ up to the closed terms. Then we show the generation lemma for $\mathbf{LST}_N$, from which the subject reduction theorem follows immediately. Instead of repeating the same argument as before, we shall just point out the modifications to be needed.

The inference rules of $\mathbf{LST}_N$ are the same as the inference rules of $\mathbf{ILAL2}_N$(see Figure 4.2) except that we have the rules in Figure 6.2 instead of $\forall E$ and $\forall I$ in Figure 4.2.

It is routine to show that $\mathbf{LST}_N$ is equivalent to **LST** as far as the closed terms are concerned.

Define a binary relation $>$ by

$$\begin{array}{rcl} A & > & \forall x.A \\ \forall x.A & > & A[u/x] \\ A[u/x] & > & u \in \{x|A\} \\ u \in \{x|A\} & > & A[u/x] \end{array}$$

With this definition of $>$, we can show an analogue of the generation lemma for $\mathbf{ILAL2}_N$ (Lemma 4.17). Therefore we have:

Theorem 6.5 (Subject Reduction for $\mathbf{LST}_N$) *If* $\Gamma \vdash t{:}A$ *is derivable in* $\mathbf{LST}_N$ *and* $t \longrightarrow u$, *then* $\Gamma \vdash u{:}A$ *is derivable in* $\mathbf{LST}_N$.

Corollary 6.6 (Subject Reduction for LST) *If* $\Gamma \vdash t{:}A$ *is derivable in* **LST** *and* $t \longrightarrow u$, *then* $\Gamma \vdash u{:}A$ *is derivable in* **LST**.

6.3 Cut-Elimination Theorem for LST

The subject reduction theorem in conjunction with normalizability of λLA terms almost implies the cut-elimination theorem for **LST**. However, a typing derivation for a normal term may still contain several *implicit cuts*, which do not appear as redices in λLA. In this section, we shall describe how to eliminate those implicit cuts and thus complete the proof of the cut-elimination theorem for **LST**.

Proposition 6.7 *If* $\Gamma \vdash t{:}C$ *is derivable in* **LST**, *then so is* $\Gamma[\vec{u}/\vec{x}] \vdash t{:}C[\vec{u}/\vec{x}]$, *for every sequence* $\vec{u}$ *of terms of* **LST**.

Proof. By induction on the length of the derivation. ■

Lemma 6.8 *If* $\Gamma \vdash t:C$ *is derivable in* **LST** *and* t *is normal, then there is a cut-free derivation of the same conclusion.*

Proof. Let π be a derivation of $\Gamma \vdash t:C$. For each (Cut) inference occurring in π we define its *rank* as the size (i.e., the number of inference rules) of the subderivation above it.

Since t is normal, π cannot contain a principal cut for $\multimap$, ! and §. In what follows, we shall show that the other types of cuts can be removed or replaced with a cut of smaller rank.

(Case 1) Principal cuts for $\forall$ and $\in$. Apply the following reduction:

$$\dfrac{\dfrac{\overset{\vdots\ \pi_1}{\Gamma_1 \vdash u:A}}{\Gamma_1 \vdash u:\forall y.A} \quad \dfrac{\overset{\vdots\ \pi_2}{x:A[v/y],\Gamma_2 \vdash t:C}}{x:\forall y.A,\Gamma_2 \vdash t:C}}{\Gamma_1,\Gamma_2 \vdash t[u/x]:C}\,Cut \quad\longrightarrow\quad \dfrac{\overset{\vdots\ \pi_1[v/y]}{\Gamma_1 \vdash u:A[v/y]} \quad \overset{\vdots\ \pi_2}{x:A[v/y],\Gamma_2 \vdash t:C}}{\Gamma_1,\Gamma_2 \vdash t[u/x]:C}\,Cut$$

$$\dfrac{\dfrac{\overset{\vdots\ \pi_1}{\Gamma_1 \vdash u:A[v/y]}}{\Gamma_1 \vdash u:v \in \{y|A\}} \quad \dfrac{\overset{\vdots\ \pi_2}{x:A[v/y],\Gamma_2 \vdash t:C}}{x:v \in \{y|A\},\Gamma_2 \vdash t:C}}{\Gamma_1,\Gamma_2 \vdash t[u/x]:C}\,Cut \quad\longrightarrow\quad \dfrac{\overset{\vdots\ \pi_1}{\Gamma_1 \vdash u:A[v/y]} \quad \overset{\vdots\ \pi_2}{x:A[v/y],\Gamma_2 \vdash t:C}}{\Gamma_1,\Gamma_2 \vdash t[u/x]:C}\,Cut$$

(Case 2) Weakening cuts. Apply the following reduction:

$$\dfrac{\overset{\vdots\ \pi_1}{\Gamma_1 \vdash u:A} \quad \dfrac{\overset{\vdots\ \pi_2}{\Gamma_2 \vdash t:C}}{x:A,\Gamma_2 \vdash t:C}\,Weak}{\Gamma_1,\Gamma_2 \vdash t:C}\,Cut \quad\longrightarrow\quad \dfrac{\overset{\vdots\ \pi_2}{\Gamma_2 \vdash t:C}}{\overline{\Gamma_1,\Gamma_2 \vdash t:C}}\,Weak.$$

(Case 3) Axiom cuts. Apply the following reduction:

$$\dfrac{x:A \vdash x:A \quad \overset{\vdots\ \pi_2}{x:A,\Gamma_2 \vdash t:C}}{x:A,\Gamma_2 \vdash t:C}\,Cut \quad\longrightarrow\quad \overset{\vdots\ \pi_2}{x:A,\Gamma_2 \vdash t:C.}$$

(Case 4) Commutative cuts, i.e., when one of the cut formulas is not principal. We shall consider only some typical cases. If the cut is of the form

$$\dfrac{\dfrac{\vdots\ \pi_1}{\Gamma_1 \vdash u : A \multimap B} \quad \dfrac{\dfrac{\vdots\ \pi_2}{x : A \multimap B, \Gamma_2' \vdash t' : C'}}{x : A \multimap B, \Gamma_2 \vdash t : C}\ J}{\Gamma_1, \Gamma_2 \vdash t[u/x] : C}\ Cut,$$

namely J is not $\multimap l$, then replace it with

$$\dfrac{\dfrac{\dfrac{\vdots\ \pi_1}{\Gamma_1 \vdash u : A \multimap B} \quad \dfrac{\vdots\ \pi_2}{x : A \multimap B, \Gamma_2' \vdash t' : C'}}{\Gamma_1, \Gamma_2' \vdash t'[u/x] : C'}\ Cut}{\Gamma_1, \Gamma_2 \vdash t[u/x] : C}\ J \quad .$$

If the cut is of the form

$$\dfrac{\dfrac{\ldots}{\Gamma_1 \vdash u : A \multimap B}\ J \quad \dfrac{\dfrac{\vdots}{\Delta_1 \vdash v : A} \quad \dfrac{\vdots}{y : B, \Delta_2 \vdash t : C}}{x : A \multimap B, \Delta_1, \Delta_2 \vdash t[xv/y] : C}}{\Gamma_1, \Delta_1, \Delta_2 \vdash t[uv/y] : C}\ Cut,$$

then the only possibilities of J are $(Cntr)$, $(\multimap l)$, $(\forall l)$ and $(\in\ l)$. In each case, the cut can be successfully replaced with another one of smaller rank. For example, when (J) is $(\forall l)$, apply the following reduction:

$$\dfrac{\dfrac{\dfrac{\vdots}{z : D[s/w], \Gamma_1 \vdash u : A \multimap B}}{z : \forall w.D, \Gamma_1 \vdash u : A \multimap B}\ \forall l \quad \dfrac{\dfrac{\vdots}{\Delta_1 \vdash v : A} \quad \dfrac{\vdots}{y : B, \Delta_2 \vdash t : C}}{x : A \multimap B, \Delta_1, \Delta_2 \vdash t[xv/y] : C}}{z : \forall w.D, \Gamma_1, \Delta_1, \Delta_2 \vdash t[uv/y] : C}\ Cut$$

$$\longrightarrow \qquad \dfrac{\dfrac{\dfrac{\vdots}{z : D[s/w], \Gamma_1 \vdash u : A \multimap B} \quad \dfrac{\dfrac{\vdots}{\Delta_1 \vdash v : A} \quad \dfrac{\vdots}{y : B, \Delta_2 \vdash t : C}}{x : A \multimap B, \Delta_1, \Delta_2 \vdash t[xv/y] : C}}{z : D[s/w], \Gamma_1, \Delta_1, \Delta_2 \vdash t[uv/y] : C}\ Cut}{z : \forall w.D, \Gamma_1, \Delta_1, \Delta_2 \vdash t[uv/y] : C}\ \forall l \quad .$$

The point is that J cannot be neither $(!l)$ nor $(\S l)$, since these two would introduce a commutative redex.

Other cases are similar. ■

Theorem 6.9 (Cut-Elimination Theorem for LST) *If* $\Gamma \vdash t : C$ *is derivable in* **LST***, then* $\Gamma \vdash u : C$ *is cut-free derivable, where* u *is the normal form of* t*.*

Proof. By normalizability of λLA terms, the subject reduction theorem and Lemma 6.8. ■

Remark 6.10 It is straightforward to adapt the above argument for **ILAL2**, since it is carried out by induction on the size of proofs, not by induction on the complexity of formulas. Hence analogues of Lemma 6.8 and Theorem 6.9 hold for **ILAL2**, too.

The cut-elimination theorem has the following corollary, which will play an important role in the program extraction.

Corollary 6.11

1. *If $\vdash t\!:\!\S A$ is derivable and t is normal, then t is of the form $\S t'$ and $\vdash t'\!:\!A$ is derivable.*
2. *If $\vdash t\!:\!\exists y.A$ is derivable and t is normal, then t is of the form $\lambda z.zt'$ and $\vdash t'\!:\!A[u/y]$ is derivable for some term u of* **LST**.

Proof.

1. Immediate.
2. Recall that $\exists y.A$ is defined as $\forall x.(\forall y.(A \multimap t_0 \in x) \multimap t_0 \in x)$. The last part of the cut-free derivation of $\vdash t\!:\!\exists y.A$ must be of the following form:

$$\cfrac{\cfrac{\cfrac{\cfrac{\begin{matrix}\vdots\\ \vdash t'\!:\!A[u/y]\end{matrix}\qquad w\!:\!t_0 \in x \vdash w\!:\!t_0 \in x}{z\!:\!A[u/y] \multimap t_0 \in x \vdash zt'\!:\!t_0 \in x}}{z\!:\!\forall y.(A \multimap t_0 \in x) \vdash zt'\!:\!t_0 \in x}}{\vdash \lambda z.zt'\!:\!\forall y.(A \multimap t_0 \in x) \multimap t_0 \in x}}{\vdash \lambda z.zt'\!:\!\exists y.A}\ ,$$

   and $t \equiv \lambda z.zt'$.

■

6.4 Extraction of λLA terms from LST proofs

Now we shall describe how to extract a term of λLA from a proof of **LST**. Instead of dealing with the general case, we shall confine ourselves to proofs of formulas of the form $\forall x \in \mathsf{W}.\S^d \exists y \in \mathsf{W}.A$, which should be sufficient for illustrating the general pattern.

First, through a careful analysis of cut-free derivations, we can show the following:

Proposition 6.12 *Suppose that t be a normal term of λLA.*

1. *For every natural number n, $\vdash t\!:\!\mathsf{n} \in \mathsf{N}$ is derivable if and only if $t \equiv \overline{n}$ (or t is an η-variant of $\overline{1}$).*
2. *For every $w \in \{0,1\}^*$, $\vdash t\!:\!\mathsf{w} \in \mathsf{W}$ is derivable if and only if $t \equiv \overline{w}$ (or t is an η-variant of $\overline{0}$ or $\overline{1}$).*

The following terms are used in the program extraction.

Definition 6.13 For each $d \geq 0$, define λLA terms $Fst^d(z)$ and $Ext^d(z)$, both having a free variable z, as follows:

$$\begin{array}{rcl} Fst^0(z) & \equiv & z(\lambda x_1 x_2.x_1) \\ Fst^{d+1}(z) & \equiv & \mathsf{let}\ z\ \mathsf{be}\ \S z'\ \mathsf{in}\ \S Fst^d(z') \\ Ext^0(z) & \equiv & z(\lambda x.x) \\ Ext^{d+1}(z) & \equiv & \mathsf{let}\ z\ \mathsf{be}\ \S z'\ \mathsf{in}\ \S Ext^d(z') \end{array}$$

Lemma 6.14

1. *$z : \S^d A \otimes B \vdash Fst^d(z) : \S^d A$ is derivable for any pair of types A and B.*
2. *$Fst^d(\S^d t) \longrightarrow^* \S^d(Fst^0(t))$.*
3. *$Ext^d(\S^d \lambda z.zt) \longrightarrow^* \S^d t$.*

Proof.
1. Let y be a variable which does not occur in A and B.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x_1 : A \vdash x_1 : A}{x_1 : A \vdash x_1 : t_0 \in \{y|A\}}}{x_1 : A, x_2 : B \vdash x_1 : t_0 \in \{y|A\}}}{\vdash \lambda x_1 x_2.x_1 : A \multimap B \multimap t_0 \in \{y|A\}} \quad \dfrac{z' : A \vdash z' : A}{z' : t_0 \in \{y|A\} \vdash z' : A}}{z : (A \multimap B \multimap t_0 \in \{y|A\}) \multimap t_0 \in \{y|A\} \vdash z(\lambda x_1 x_2.x_1) : A}}{z : A \otimes B \vdash z(\lambda x_1 x_2.x_1) : A}$$

2 and 3 are easily checked. ∎

Our main theorem in this chapter is the following:

Theorem 6.15 (Program Extraction) *Let $\vdash \forall x \in \mathsf{W}.\S^d \exists y \in \mathsf{W}.A$ be derivable in* **LST**. *Then there is a term u of λLA such that for every $w \in \{0,1\}^*$, $u\overline{w}$ reduces to $\S^d \overline{w'}$ and the formula $A[\mathsf{w}/x, \mathsf{w}'/y]$ is provable in* **LST**.

Proof. Let t be a term of λLA such that

$$\vdash t : \forall x \in \mathsf{W}.\S^d \exists y \in \mathsf{W}.A$$

is derivable in **LST**. We claim that the desired term is $\lambda z.Fst^d(Ext^d(tz))$.

Let $w \in \{0,1\}^*$. By Proposition 6.12, $\vdash \overline{w} : \mathsf{w} \in \mathsf{W}$ is derivable. On the other hand, it is easy to see that

$$\vdash t : \mathsf{w} \in \mathsf{W} \multimap \S^d \exists y \in \mathsf{W}.A[\mathsf{w}/x]$$

is derivable, so that we have

$$\vdash t\overline{w} : \S^d \exists y \in \mathsf{W}.A[\mathsf{w}/x].$$

Let $nf(t\overline{w})$ be the normal form of $t\overline{w}$. Then, by the subject reduction theorem and Corollary 6.11, $nf(t\overline{w})$ must be of the form $\S^d \lambda z.zv$ and

$$\vdash v : s \in \mathsf{W} \otimes A[\mathsf{w}/x, s/y]$$

is derivable for some term s of **LST**. Therefore, $s \in \mathsf{W}$ and $A[\mathsf{w}/x, s/y]$ are provable in **LST**. Hence by Proposition 5.35, $s \equiv \mathsf{w}'$ for some $w' \in \{0,1\}^*$.

By Lemma 6.14, we have $\vdash Fst^0(v) : \mathsf{w}' \in \mathsf{W}$, hence the normal form of $Fst^0(v)$ is $\overline{w'}$ by the subject reduction theorem and Proposition 6.12.

To put things together, we obtain:

$$(\lambda z.Fst^d(Ext^d(tz)))\overline{w} \longrightarrow Fst^d(Ext^d(t\overline{w})) \longrightarrow^* Fst^d(Ext^d(\S^d \lambda z.zv))$$

$$\longrightarrow^* Fst^d(\S^d v) \longrightarrow^* \S^d(Fst^0(v)) \longrightarrow^* \S^d\overline{w'},$$

as required. ■

As a corollary, we have the following characterization theorem:

Corollary 6.16 (Characterization of the Polytime Functions) *Let $F : \{0,1\}^* \longrightarrow \{0,1\}^*$ be a function. The following are equivalent:*

1. *F is polytime computable.*
2. *F is provably total with domain and codomain W in **LST**.*
3. *F is λLA-representable.*

Proof.
(1 ⇒ 2) By Theorem 5.48.
(2 ⇒ 3) By definition, there is a term f of **LST**, a natural number $d \geq 0$ and a term t of λLA such that

$$\vdash t : \forall x \in \mathsf{W}.\S^d \exists^! y \in \mathsf{W}(\langle x, y\rangle \in f).$$

From this, we can easily obtain

$$\vdash t' : \forall x \in \mathsf{W}.\S^d \exists y \in \mathsf{W}(\langle x, y\rangle \in f).$$

Hence the program extraction theorem applies, and we obtain a term u of λLA such that for any $w \in \{0,1\}^*$, $u\overline{w}$ reduces to $\S^d\overline{w'}$ and $\langle \mathsf{w}, \mathsf{w}'\rangle \in f$ is provable in **LST**. The latter is equivalent to $F(w) = w'$ by Proposition 5.32.
(3 ⇒ 1) By the polytime strong normalization theorem. ■

Chapter 7

Phase Semantics for Light Logic

In this chapter, we shall introduce *light affine phase semantics*, which is meant to be a sound and complete semantics for **ILAL**, and show the finite model property for **ILAL**. As a consequence, we obtain decidability of **ILAL**, and decidability of the type inhabitation problem for **ILAL** as a type assignment system. Light affine phase semantics has its origin in phase semantics for **ILL** [Abr90, Tro92, Ono94, Oka96, Oka99], phase semantics for Affine Logic [Laf97, Oka01] and phase semantics for **LLL** [KOSar]. In particular, our interpretation of the light exponentials is a straightforward simplification of the interpretation in the last one.

Although phase semantics is somewhat abstract in its nature, there is a good way to grasp its intuition, that is to think of it as a generalization of Kripke semantics for Intuitionistic Logic. Recall that Kripke semantics is based on partially ordered sets. Given a partially ordered set $(P, \leq)$, each formula A is interpreted by an upward closed subset of P. Conjunction is interpreted by set-theoretic intersection, while disjunction is by union. The interpretation of implication is peculiar, and is based on the structure of partially ordered sets. Phase semantics modifies this construction in several ways. First, we consider commutative monoids $(M, \cdot, \varepsilon)$ rather than partially ordered sets. Second, instead of considering $\leq$-upward closure, we explicitly introduce a closure operator Cl over $\mathcal{P}(M)$, and interpret each formula A by a subset $X \subseteq M$ which is closed in the sense of Cl, namely $X = Cl(X)$. Additive connectives are interpreted based on set-theoretic operations $\cap$ and $\cup$ as before, while multiplicative connectives are interpreted by monoid operations.

The structure of this chapter is almost parallel to the structure of Chapter 2, except that phase semantics for **ILLL** is mentioned at last in Section 7.4. In Section 7.1, we shall recall phase semantics for **ILL**. We shall also introduce an important technique of the quotient model construction via logical congruence, which was originally devised for classical systems of Linear and Affine Logics by Lafont [Laf97] and later modified for intuitionistic systems by Okada and Terui [OT99]. In Section 7.2, we shall describe phase semantics for **IAL**, and apply the quotienting technique to obtain the finite model property for **IAL** (a result proved in [OT99]). This serves as a preliminary to the later application of the quotienting technique to **ILAL**. In Section 7.3, we shall introduce light affine phase semantics for **ILAL**. Then we shall show the finite model property for **ILAL** with respect to the generalized version of light affine phase semantics by applying the quotienting technique developed before; this is a novel result of this thesis. We shall conclude this chapter with some remarks and open problems in Section 7.4.

7.1 Phase Semantics for ILL

In this section, we shall recall the fundamentals of phase semantics for **ILL** [Abr90, Tro92, Ono94, Oka96, Oka99]. The basic definition of phase semantics (in 7.1.2) essentially comes from [Abr90, Tro92, Ono94]. The notion of co-basis and Theorem 7.10 is due to our unpublished manuscript [KOT99]. The definition of the canonical model and the phase-semantic proof to the cut-elimination theorem (7.1.3) are due to [Oka96, Oka99]. The quotienting technique (in 7.1.4) is due to [Laf97, OT99].

7.1.1 Preliminary on Monoids

To fix notation and terminology, we shall recall the definition of monoids and some related concepts.

Definition 7.1 A *monoid* is a triple $(M, \cdot, \varepsilon)$ such that M is a set, $\cdot$ is a function $M \times M \to M$, ε is an element of M, and the following properties are satisfied for any $x, y, z \in M$:

- $(x \cdot y) \cdot z = x \cdot (y \cdot z)$;
- $\varepsilon \cdot x = x \cdot \varepsilon = x$.

A monoid is *commutative* if $x \cdot y = y \cdot x$ for any $x, y \in M$.

For $X, Y \subseteq M$, we define

$$X \cdot Y \;=\; \{x \cdot y | x \in X, y \in Y\}.$$

$X \cdot Y$ is associative and has unit $\{\varepsilon\}$.

Definition 7.2 A monoid $M_1 = (M_1, \cdot_1, \varepsilon_1)$ is said to be a *submonoid* of another monoid $M_2 = (M_2, \cdot_2, \varepsilon_2)$ if $M_1 \subseteq M_2$, $\varepsilon_1 = \varepsilon_2$ and $\cdot_1$ agrees with $\cdot_2$ on M_1.

Definition 7.3 Given two monoids M_1, M_2 A *monoid homomorphism* from M_1 to M_2 is a function $h : M_1 \longrightarrow M_2$ such that $h(\varepsilon) = \varepsilon$ and $h(x \cdot y) = h(x) \cdot h(y)$ for any $x, y \in M_1$.

7.1.2 Phase Structures

Definition 7.4 (Cf. [Tro92]) A *phase structure* (M, Cl) consists of a commutative monoid $M = (M, \cdot, \varepsilon)$ with a *closure operator* Cl, that is a mapping from the subsets of M to the subsets of M such that for all $X, Y \subseteq M$:

(Cl1) $X \subseteq Cl(X)$,

(Cl2) $Cl(Cl(X)) \subseteq Cl(X)$,

(Cl3) $X \subseteq Y \Longrightarrow Cl(X) \subseteq Cl(Y)$,

(Cl4) $Cl(X) \cdot Cl(Y) \subseteq Cl(X \cdot Y)$.

A set $X \subseteq M$ is said to be *closed* if $X = Cl(X)$.

Define

$$X \multimap Y \;=\; \{z \in M | \forall x \in X \; x \cdot z \in Y\}.$$

We have:

(*) $X \cdot Y \subseteq Z \iff X \subseteq Y \multimap Z.$

In particular, $Y \subseteq Z \Longrightarrow \varepsilon \in Y \multimap Z$.

Lemma 7.5

1. *$X \multimap Y$ is closed whenever Y is.*
2. *An arbitrary intersection $\bigcap_{\lambda\in\Lambda} X_\lambda$ is closed whenever each X_λ is closed.*

Proof.
(1) Assume that $Y = Cl(Y)$. We have $X \multimap Y \subseteq Cl(X \multimap Y)$ by (Cl1). On the other hand,

$$\begin{array}{rcll} X \multimap Y \subseteq X \multimap Y & \Rightarrow & X \cdot X \multimap Y \subseteq Y & \text{by (*)} \\ & \Rightarrow & Cl(X \cdot X \multimap Y) \subseteq Cl(Y) & \text{by (Cl3)} \\ & \Rightarrow & Cl(X) \cdot Cl(X \multimap Y) \subseteq Cl(Y) & \text{by (Cl4)} \\ & \Rightarrow & X \cdot Cl(X \multimap Y) \subseteq Cl(Y) & \text{by (Cl1)} \\ & \Rightarrow & X \cdot Cl(X \multimap Y) \subseteq Y & \text{by (Cl2)} \\ & \Rightarrow & Cl(X \multimap Y) \subseteq X \multimap Y & \text{by (*)} \end{array}$$

(2) We have $\bigcap_{\lambda\in\Lambda} X_\lambda \subseteq Cl(\bigcap_{\lambda\in\Lambda} X_\lambda)$. On the other hand, for each $\lambda \in \Lambda$, $\bigcap_{\lambda\in\Lambda} X_\lambda \subseteq X_\lambda$. Hence

$$Cl(\bigcap_{\lambda\in\Lambda} X_\lambda) \subseteq Cl(X_\lambda) = X_\lambda$$

by (Cl3) and (Cl2). Therefore, $Cl(\bigcap_{\lambda\in\Lambda} X_\lambda) \subseteq \bigcap_{\lambda\in\Lambda} X_\lambda$. ■

Definition 7.6 A preorder $\preceq$ on M is defined by:

$$x \preceq y \iff Cl(\{x\}) \subseteq Cl(\{y\}).$$

Remark 7.7 Observe that if $X = Cl(X)$ then X is downward closed with respect to $\preceq$. This is reminiscent of the $\leq$-upward closure of Kripke semantics. The closure operator Cl is, however, more abstract than the latter, since the converse direction does not hold in general, i.e., being downward closed with respect to $\preceq$ is not sufficient for being closed in the sense of Cl.

Definition 7.8 Given a phase structure (M, Cl), one defines the following sets and operations over $X, Y \subseteq M$:

- $\mathbf{1} = Cl(\{\varepsilon\})$; $\top = M$; $\mathbf{0} = Cl(\emptyset)$.
- $X \otimes Y = Cl(X \cdot Y)$; $X \mathbin{\&} Y = X \cap Y$; $X \oplus Y = Cl(X \cup Y)$.

All these operations yield a closed set whenever its components X and Y are.

Definition 7.9 ([KOT99]) Given a phase structure (M, Cl), a family $\mathcal{X} = \{X_\lambda\}_{\lambda\in\Lambda}$ of closed subsets of M is called a *co-basis* if for any $Y \subseteq M$:

$$Cl(Y) = \bigcap\{X_\lambda | Y \subseteq X_\lambda, \lambda \in \Lambda\}.$$

Indeed, the (uncountable) family of *all* closed sets forms a trivial co-basis. Conversely, a closure operator Cl can be *induced* by a specific family of subsets of M satisfying a certain property:

Theorem 7.10 *Let M be a monoid and $\mathcal{X} = \{X_\lambda\}_{\lambda\in\Lambda}$ be a family of subsets of M such that $X \in \mathcal{X}$ and $x \in M$ implies that $\{x\} \multimap X \in \mathcal{X}$. Then operator Cl defined by:*

$$Cl(Y) = \bigcap\{X_\lambda | Y \subseteq X_\lambda, \lambda \in \Lambda\}$$

satisfies all axioms (Cl1)—(Cl4).

Proof. (Cl1) — (Cl3) are all immediate. We claim that Cl above satisfies the following:

(Cl4') $X \cdot Cl(Y) \subseteq Cl(X \cdot Y)$ for any $X, Y \subseteq M$.

Then (Cl4) is derived by using (Cl4') twice:

$$\begin{array}{rcl} Cl(X) \cdot Cl(Y) & \subseteq & Cl(Cl(X) \cdot Y) \\ & = & Cl(Y \cdot Cl(X)) \\ & \subseteq & Cl(Cl(Y \cdot X)) \\ & = & Cl(X \cdot Y). \end{array}$$

To show (Cl4"), let $X \cdot Y \subseteq X_\lambda$ with $\lambda \in \Lambda$. Then for any $x \in X$, $\{x\} \cdot Y \subseteq X_\lambda$, i.e.,

$$Y \subseteq \{x\} \multimap X_\lambda.$$

Since $\{x\} \multimap X_\lambda \in \mathcal{X}$ by assumption,

$$\bigcap\{X_\lambda | Y \subseteq X_\lambda, \lambda \in \Lambda\} \subseteq \{x\} \multimap X_\lambda.$$

Therefore

$$\{x\} \cdot \bigcap\{X_\lambda | Y \subseteq X_\lambda, \lambda \in \Lambda\} \subseteq X_\lambda,$$

for any $x \in X$. Namely,

$$X \cdot \bigcap\{X_\lambda | Y \subseteq X_\lambda, \lambda \in \Lambda\} \subseteq \bigcap\{X_\lambda | X \cdot Y \subseteq X_\lambda, \lambda \in \Lambda\},$$

as required. ■

Several interpretations have been proposed for the exponential modality !. Among those, we adopt Lafont's interpretation [Laf97] here.

Given a phase structure (M, Cl), the set

$$J(M) = \{x \in \mathbf{1} \mid x \preceq x \cdot x\}$$

is easily shown to be a submonoid of M.

Definition 7.11 ([Laf97]) An *enriched phase structure* (M, Cl, K) consists of a phase structure (M, Cl) with a submonoid K of $J(M)$. In an enriched phase structure, one defines

- $!X = Cl(X \cap K)$.

For example, K may be $J(M)$ itself, or the singleton $\{\varepsilon\}$, or the set of idempotents $I = \{x \in \mathbf{1} \mid x \cdot x = x\}$ in $\mathbf{1}$.

Definition 7.12 A *valuation* v over an enriched phase structure (M, Cl, K) is an assignment of a closed set $v(\alpha)$ to each atomic formula α. A *phase model* is a phase structure endowed with a valuation.

The valuation v is extended to arbitrary **ILAL** formulas in a natural way:

- $v(\mathbf{1}) = \mathbf{1}$, $v(\top) = \top$, $v(\mathbf{0}) = \mathbf{0}$;
- $v(A \otimes B) = v(A) \otimes v(B)$, $v(A \multimap B) = v(A) \multimap v(B)$;
- $v(A \,\&\, B) = v(A) \,\&\, v(B)$, $v(A \oplus B) = v(A) \oplus v(B)$;
- $v(!A) = !v(A)$.

A formula A is *satisfied* in (M, Cl, K, v) if $\varepsilon \in v(A)$.

The phase models are sound and complete for **ILL**; indeed, they are complete for *cut-free* **ILL**. Hence the cut-elimination theorem follows [Oka96, Oka99]:

Theorem 7.13 *For any formula A, the following are equivalent:*

1. *A is provable in* **ILL**;
2. *A is cut-free provable in* **ILL**;
3. *A is satisfied in every phase model.*

The implication $(1 \Rightarrow 3)$ can be shown by the standard soundness argument. $(2 \Rightarrow 1)$ is trivial. To show $(3 \Rightarrow 2)$, we construct the canonical model, which is described below.

7.1.3 The Canonical Model

Here we shall describe Okada's construction of the canonical model. The construction will be later modified so that a finitely generated canonical model is obtained (in 7.2.2).

Definition 7.14 ([Oka96, Oka99]) The *canonical model* $\mathbf{ILL}^\bullet = (M_0, Cl_0, K_0, v_0)$ for **ILL** is defined as follows:

- M_0 is the free commutative monoid generated by the formulas of **ILAL**; namely elements of M are multisets of formulas, the binary operation $\cdot$ is multiset union, and the unit is the empty multiset $\emptyset$.

- For each formula C of **ILL**, define its *outer value* $[\![C]\!]$ by

$$[\![C]\!] = \{\Sigma \mid \Sigma \vdash C \text{ is } \textit{cut-free} \text{ provable in } \mathbf{ILL}\}.$$

- Let $\mathcal{X}$ be the set of outer values of all **ILL** formulas. Define Cl_0 by

$$Cl_0(X) = \bigcap\{Y \mid X \subseteq Y, Y \in \mathcal{X}\}.$$

- $K_0 = \{!\Gamma \mid !\Gamma \in M_0\}$.
- $v(\alpha) = [\![\alpha]\!]$ for each atomic formula α.

ILL$^\bullet$ is indeed a phase model, and the following *Main Lemma* establishes Theorem 7.13:

Lemma 7.15 (Main Lemma [Oka96, Oka99]) *For any formula A of* **ILL***, $A \in v_0(A) \subseteq [\![A]\!]$. In particular, if $\varepsilon \in v_0(A)$, A is cut-free provable in* **ILL***.*

This lemma is proved by induction on A.

7.1.4 Quotient Model Construction

Here we shall describe the technique of quotient model construction [Laf97, OT99], which is the key to prove the finite model property for various systems. In what follows, we shall often use notation $f \circ g(X)$ for function composition in place of $f(g(X))$.

Let (M, Cl, K, v) be a phase model. For any $x, y \in M$, define $x \sim y$ by

$$x \sim y \iff x \preceq y \text{ and } y \preceq x.$$

Then $\sim$ is reflexive, symmetric, transitive, and satisfies:

- $x \sim y$ implies $x \cdot z \sim y \cdot z$.
- $x \sim y$ and $y \in X$ imply $x \in Cl(X)$ for any $X \subseteq M$.

A binary relation satisfying these properties are called a *logical congruence.*

Let $\pi(x)$ denote the equivalence class to which x belongs. Then, the *quotient model* $(M_\sim, Cl_\sim, K_\sim, v_\sim)$ of (M, Cl, K, v) is defined as follows:

$$\begin{array}{rcl}
M_\sim & = & \{\pi(x) \mid x \in M\} \\
\pi(x) \cdot \pi(y) & = & \pi(x \cdot y) \\
\varepsilon & = & \pi(\varepsilon) \\
Cl_\sim(X) & = & \pi(Cl(\pi^{-1}(X))) \\
K_\sim & = & \pi(K) \\
v_\sim(\alpha) & = & \pi(v(\alpha)).
\end{array}$$

π is a monoid homomorphism from M to $M_\sim$. Moreover, $\pi^- \circ \pi(X) = Cl(X)$ for any $X \subseteq M$.

Lemma 7.16 *For any $X \subseteq M$, $\pi(Cl(X)) = Cl_\sim(\pi(X))$. Therefore, $\pi(X)$ is closed whenever X is.*

Proof.

$$\begin{aligned} Cl_\sim \circ \pi(X) &= \pi \circ Cl \circ \pi^{-1} \circ \pi(X) \\ &= \pi \circ Cl \circ Cl(X) \\ &= \pi \circ Cl(X). \end{aligned}$$

■

Lemma 7.17 $(M_\sim, Cl_\sim, K_\sim, v_\sim)$ *is a phase model.*

Proof. First we prove that $Cl_\sim$ is a closure operator.
(Cl1) Suppose that $X \subseteq M_\sim$. Then we have:

$$\begin{aligned} \pi^{-1}(X) &\subseteq Cl(\pi^{-1}(X)) \\ X = \pi(\pi^{-1}(X)) &\subseteq \pi(Cl(\pi^{-1}(X))). \end{aligned}$$

(Cl2) Let $X \subseteq Y \subseteq M_\sim$. Then we have:

$$\begin{aligned} \pi^{-1}(X) &\subseteq \pi^{-1}(Y) \\ Cl \circ \pi^{-1}(X) &\subseteq Cl \circ \pi^{-1}(Y) \\ \pi \circ Cl \circ \pi^{-1}(X) &\subseteq \pi \circ Cl \circ \pi^{-1}(Y). \end{aligned}$$

(Cl3)

$$\begin{aligned} \pi \circ Cl \circ \pi^{-1} \circ \pi \circ Cl \circ \pi^{-1}(X) &= \pi \circ Cl \circ Cl \circ \pi^{-1}(X) \\ &= \pi \circ Cl \circ Cl \circ \pi^{-1}(X) \\ &= \pi \circ Cl \circ \pi^{-1}(X). \end{aligned}$$

(Cl4) Observe that

$$\begin{aligned} \pi(Z_1) \cdot \pi(Z_2) &= \pi(Z_1 \cdot Z_2) \ \text{ for any } Z_1, Z_2 \subseteq M; \\ \pi^{-1}(W_1) \cdot \pi^{-1}(W_2) &= \pi^{-1}(W_1 \cdot W_2) \ \text{ for any } W_1, W_2 \subseteq M_\sim. \end{aligned}$$

Therefore,

$$\begin{aligned} \pi \circ Cl \circ \pi^{-1}(X) \cdot \pi \circ Cl \circ \pi^{-1}(Y) &= \pi(Cl \circ \pi^{-1}(X) \cdot Cl \circ \pi^{-1}(Y)) \\ &\subseteq \pi \circ Cl(\pi^{-1}(X) \cdot \pi^{-1}(Y)) \\ &\subseteq \pi \circ Cl \pi^{-1}(X \cdot Y). \end{aligned}$$

It remains to show that $K_\sim$ is a submonoid of $J(M_\sim)$. For any $x \in K_\sim$, there is $x' \in K$ such that $x = \pi(x')$. Since $x' \preceq x' \cdot x'$, we have

$$\pi(x') \preceq \pi(x' \cdot x') = \pi(x') \cdot \pi(x').$$

Furthermore, $x' \in \mathbf{1} = Cl(\{\varepsilon\})$. Therefore,

$$\begin{aligned} \pi(x') &\in \pi \circ Cl(\{\varepsilon\})) \\ &\subseteq \pi \circ Cl \circ \pi^{-1}(\{\varepsilon\}) \\ &= Cl_\sim(\{\varepsilon\}). \end{aligned}$$

Therefore, $K_\sim \subseteq J(M_\sim)$. ■

Lemma 7.18 *All operations* $\mathbf{1}, \mathbf{0}, \top, \otimes, \multimap, \&, \oplus, !$ *commute with* π*. Namely,*

$$\begin{array}{rcl} \pi(*) & = & *, \textit{for } * \in \{\mathbf{1}, \mathbf{0}, \top\}; \\ \pi(X) \star \pi(Y) & = & \pi(X \star Y), \textit{for } * \in \{\otimes, \multimap, \&, \oplus\}, \\ !\pi(X) & = & \pi(!X), \end{array}$$

for any closed sets $X, Y \subseteq M$*.*

Proof.
(i)

$$\begin{array}{rcl} \pi(X \otimes Y) & = & \pi \circ Cl(X \cdot Y) \\ & = & Cl_{\sim}(\pi(X \cdot Y)) \\ & = & Cl_{\sim}(\pi(X) \cdot \pi(Y)) \\ & = & \pi(X) \otimes \pi(Y). \end{array}$$

(ii)

$$\pi(X \,\&\, Y) \;=\; \pi(X) \,\&\, \pi(X).$$

(iii)

$$\begin{array}{rcl} \pi(X \oplus Y) & = & \pi \circ Cl(X \cup Y) \\ & = & Cl_{\sim}(\pi(X \cup Y)) \\ & = & Cl_{\sim}(\pi(X) \cup \pi(Y)) \\ & = & \pi(X) \oplus \pi(Y). \end{array}$$

(iv)

$$\begin{array}{rcl} x \in \pi(X) \multimap \pi(Y) & \iff & \{x\} \cdot \pi(X) \subseteq \pi(Y) \\ & \iff & \pi^{-1}(\{x\}) \cdot \pi^{-1} \circ \pi(X) \subseteq \pi^{-1} \circ \pi(Y) \\ & \iff & \pi^{-1}(\{x\}) \cdot X \subseteq Y \\ & \iff & \pi^{-1}(\{x\}) \subseteq X \multimap Y \\ & \iff & \pi \circ \pi^{-1}(\{x\}) \subseteq \pi(X \multimap Y) \\ & \iff & x \in \pi(X \multimap Y). \end{array}$$

(v)

$$\begin{array}{rcl} \pi(!X) & = & \pi(Cl(X \cap K)) \\ & = & Cl_{\sim}(\pi(X \cap K)) \\ & = & Cl_{\sim}(\pi(X) \cap \pi(K)) \\ & = & !\pi(X). \end{array}$$

The cases of constants are similar. ■

Theorem 7.19 (Lafont [Laf97], Okada-Terui [OT99]) *Every formula* A *of* **ILL** *is satisfied in* (M, Cl, K, v) *iff it is satisfied in the quotient model* $(M_{\sim}, Cl_{\sim}, K_{\sim}, v_{\sim})$*.*

Proof. By induction on A, using Lemma 7.18. ■

Proposition 7.20 *If (M, Cl, K) has a finite co-basis, then $M_\sim$ is finite.*

Proof. Let a finite co-basis $\mathcal{X}$ over (M, Cl, K) be given. It is easy to show that $x \sim y$ iff $x \in X \iff y \in X$ for any $X \in \mathcal{X}$. Hence it follows that there are exactly $2^{|\mathcal{X}|}$ equivalence classes over M. Hence $M_\sim$ is finite. ■

From the results above, we obtain:

Corollary 7.21 *For any phase structure (M, Cl, K) having a finite co-basis, there is a finite phase structure which satisfies the same set of formulas as (M, Cl, K).*

7.2 Phase Semantics for IAL

Now we move on to phase semantics for **IAL** [Ono94, Laf97, OT99, Oka01]. We shall give the definition in 7.2.1, and then prove the finite model property for **IAL** in 7.2.2, using the canonical model construction and the quotienting technique detailed before. A by-product is the cut-elimination theorem for **IAL**. The content of this section is largely based on [OT99].

7.2.1 Affine Phase Structures

Phase semantics for **IAL** is obtained from phase semantics for **ILL** by imposing a further constraint on the closed sets.

Definition 7.22 A set $X \subseteq M$ is an *ideal* if $X \cdot M \subseteq X$. An *affine phase structure* is an enriched phase structure (M, Cl, K) with every closed set ideal.

In an affine phase structure, we have:

- $\top = \mathbf{1}$; $X \otimes Y \subseteq X \,\&\, Y$.

Unrestricted Weakening is characterized by the principle $A \multimap \mathbf{1}$, which is always satisfied in an affine phase structure. Hence the soundness of affine phase structure for **IAL** is obvious. Note that the set $J(M)$ can be simply defined as $\{x | x \preceq x \cdot x\}$.

7.2.2 The Finite Model Property for IAL

First of all, we need some general facts on monoids. Let M be a commutative monoid. For $x, y \in M$, let $x \leq y$ if $y = xz$ for some $z \in M$.

Given $Y \subseteq M$, $Y \cdot M$ is always an ideal, which is denoted by $Id_M(Y)$. We drop the subscript M when they are obvious from the context.

If M is a free commutative monoid generated by $\{p_1, \dots, p_k\}$, then each $x \in M$ can be represented as $p_1^{x_1} \cdots p_k^{x_k}$ for some nonnegative integers $x_1, \dots, x_k$. In this case, $x \dot{-} y$ iff $p_1^{x_1 \dot{-} y_1} \cdots p_k^{x_k \dot{-} y_k}$ where $x_i \dot{-} y_i$ is $x_i - y_i$ if $x_i \geq y_i$ and is 0 otherwise. For $X \subseteq M$ and $y \in M$, $X \dot{-} y$ denotes the set $\{x \dot{-} y | x \in X\}$.

The following lemma is easily proved.

Lemma 7.23 *Let M be a finitely generated free commutative monoid. Then, for any $X \subseteq M$, $\{y\} \multimap Id(X) = Id(X \dot{-} y)$.*

We need, as in [Laf97], the following lemma which was also crucial in the first proof of decidability of Affine Logic by Kopylov [Kop95].

Lemma 7.24 *Let M be a finitely generated free commutative monoid, and X be an ideal of it. Then $X = Id(\{x_1, \dots, x_n\})$ for some $\{x_1, \dots, x_n\} \subseteq M$.*

For the proof, see [Laf97].

The construction of the canonical model (in 7.1.3) is slightly modified.

Definition 7.25 Let A be a formula of **IAL**. The *canonical model* $\mathbf{IAL}^\bullet[A]$ is defined analogously to $\mathbf{ILL}^\bullet$, but with some modifications:

- M_0 is the free commutative monoid generated by the *subformulas* of A.
- The outer values are defined for *sequents*: given $\Gamma, C \in M_0$, its *outer value* $[\![\Gamma \vdash C]\!]$ is defined by:
$$[\![\Gamma \vdash C]\!] = \{\Sigma \in M_0 \mid \Sigma, \Gamma \vdash C \text{ is } \textit{cut-free} \text{ provable in } \mathbf{IAL}\}.$$
Write $[\![C]\!]$ for $[\![\vdash C]\!]$.
- The co-basis $\mathcal{X}$ and Cl_0 are defined by:
$$\begin{array}{rcl} \mathcal{X} & = & \{[\![\Gamma \vdash C]\!] \| \ \Gamma, C \text{ consists of subformulas of } A.\} \\ Cl_0(X) & = & \bigcap\{Y \mid X \subseteq Y, Y \in \mathcal{X}\}. \end{array}$$
- $K_0 = \{!\Gamma \mid !\Gamma \in M_0\}$.
- $v(\alpha) = [\![\alpha]\!]$ for each atomic formula α.

Lemma 7.26 $\mathbf{IAL}^\bullet[A]$ *is a phase model.*

Proof. We have to check that (1) Cl_0 is a closure operator, and (2) K_0 is a submonoid of $J(M)$.

(1) For any $\Sigma \in M_0$ and $[\![\Gamma \vdash C]\!] \in \mathcal{X}$,
$$\{\Sigma\} \multimap [\![\Gamma \vdash C]\!] = [\![\Sigma, \Gamma \vdash C]\!] \in \mathcal{X}.$$
Hence the set $\mathcal{X}$ meets the condition of Theorem 7.10. Therefore Cl_0 is a closure operator.

(2) We have to show that $!\Gamma \preceq !\Gamma \cdot !\Gamma$ for any $!\Gamma \in K$. For this, it suffices to show that whenever $!\Gamma \cdot !\Gamma \in [\![\Delta \vdash C]\!]$, it also holds that $!\Gamma \in [\![\Delta \vdash C]\!]$. But this is immediate by Contraction. ■

Remark 7.27 It is crucial to consider the outer values for *sequents*; the outer values for formulas are not enough to form a co-basis.

The main lemma is restricted to the subformulas of A:

Lemma 7.28 *For any subformula B of A, $B \in v_0(B) \subseteq [\![B]\!]$. In particular, if $\varepsilon \in v_0(A)$, A is cut-free provable in* **ILL**.

The proof is just as for Lemma 7.15. With the help of Lemma 7.24, we can show the following:

Lemma 7.29 *The co-basis $\mathcal{X}$ is finite.*

Proof. There are only finitely many outer values $[\![B]\!]$ for formulas. By Lemma 7.24, $[\![B]\!] = Id(\{\Delta_1, \dots, \Delta_n\})$ for some $\{\Delta_1, \dots, \Delta_n\}$. For a given $\Sigma \in M_0$,

$$[\![\Sigma \vdash B]\!] = \{\Sigma\} \multimap [\![B]\!] = Id(\{\Delta_1 \dot{-} \Sigma, \dots, \Delta_n \dot{-} \Sigma\})$$

by Lemma 7.23. But there are only finitely many Π's such that $\Pi \leq \Delta_i$, i.e., the number of sets of the form $Id(\{\Delta_1 \dot{-} \Sigma, \dots, \Delta_n \dot{-} \Sigma\})$ is finite. Therefore, $\mathcal{X}$ is finite. ■

The following theorem summarizes the soundness, the completeness, the cut-elimination and the finite model property for **IAL** once for all:

Theorem 7.30 *For any formula A, the following statements are equivalent:*

1. *A is provable in* **IAL**.
2. *A is satisfied in all affine phase models.*
3. *A is satisfied in all finite affine phase models.*
4. *A is satisfied in the quotient of* $\mathbf{IAL}^\bullet[A]$.
5. *A is satisfied in* $\mathbf{IAL}^\bullet[A]$.
6. *A is cut-free provable in* **IAL**.

Proof. The implication $(3 \Rightarrow 4)$ is due to Lemma 7.29 and Proposition 7.20. $(4 \Rightarrow 5)$ holds by Theorem 7.19. $(5 \Rightarrow 6)$ holds by Lemma 7.28. Other implications are trivial. ■

In particular, we have:

Corollary 7.31 (Okada-Terui[OT99]) **IAL** *has the finite model property. Hence* **IAL** *is decidable.*

7.3 Phase Semantics for ILAL

We now introduce light affine phase semantics for **ILAL** in 7.3.1, which is an immediate modification of fibred phase semantics for **LLL** and **ILLL** [KOSar]. The definition of the latter is given in Section 7.4, where it is also explained what our modification consists in. We prove the finite model property for **ILAL** based on a slightly generalized version of light affine phase semantics in 7.3.2.

7.3.1 Light Affine Phase Structures

Definition 7.32 A *light affine phase structure* (M, Cl, f, h) is an affine phase structure (M, Cl) endowed with a pair (f, h) of functions which satisfies the following:

1. f is a function from M to $J(M)$;
2. $f(\varepsilon) = \varepsilon$;
3. h is a monoid homomorphism from M to itself;
4. f is bounded by h: for any $x \in M$, $f(x) \preceq h(x)$.

We define:

- $!X = Cl(f(X))$;
- $\S X = Cl(h(X))$.

In the above definition, condition 1 corresponds to Contraction, condition 2 corresponds to Monoidalness 2, condition 3 to Monoidalness§ and condition 4 to Weak Dereliction; Functricity is automatically satisfied.

It is possible to relax the definition above, so that f may be a *partial* function. With this relaxation, an affine phase structure (M, Cl, K) becomes a special case: let g be the identity function on M and f be defined by

$$\begin{array}{rcl} f(x) & = & x, \quad \text{if } x \in K; \\ & = & \text{undefined, otherwise.} \end{array}$$

Then (M, Cl, f, g) is a light affine phase structure with f partial, and we have $Cl(X \cap K) = Cl(f(X))$.

Example 7.33 As an example of light affine phase structures, we give a countermodel to Dereliction, Digging and Monoidalness 1, which are syntactically rejected in **ILAL**. Let $(M, \leq)$ be the following partially ordered set: $M = \{1, a, b, 0\}$ and the partial order $\leq$ is defined by

$$\begin{tikzcd} & 1 \arrow[dl] \arrow[dr] & \\ a \arrow[dr] & & b, \arrow[dl] \\ & 0 & \end{tikzcd}$$

where $x \longrightarrow y$ indicates $x \geq y$. Define a commutative binary operation $\cdot$ over M by:

$$\begin{array}{rcl} 1 \cdot x & = & x \\ 0 \cdot x & = & 0 \\ a \cdot a & = & 0 \\ a \cdot b & = & 0 \\ b \cdot b & = & b, \end{array}$$

where $x \in M$. Then $M = (M, \cdot, 1)$ is a commutative monoid. For $X \subseteq M$, define $Cl(X) = \{x | \exists y \in X(x \leq y)\}$. Then (M, Cl) is an affine phase structure with $J(M) = \{1, b, 0\}$. Finally define $f, h : M \longrightarrow M$ by:

$$\begin{array}{rcl} f(1) & = & 1 \\ f(a) & = & b \\ f(b) & = & 0 \\ f(0) & = & 0 \\ h(x) & = & 1 \end{array}$$

Then it is easy to check that h is a monoid homomorphism and f is bound by h. Therefore, (M, Cl, f, h) is a light affine phase structure. Let Y be a closed set $\{a, 0\}$. Then we have:

- $!Y = \{b, 0\}$.
- $!!Y = \{0\}$.
- $!(Y \otimes Y) = \{0\}$.
- $!Y \otimes !Y = \{b, \perp\}$.

From these, we see:

- $1 \not\in !Y \multimap Y$.
- $1 \not\in !Y \multimap !!Y$.
- $1 \not\in !Y \otimes !Y \multimap !(Y \otimes Y)$.

Therefore, Dereliction, Digging and Monoidalness 1 are not satisfied in (M, Cl, f, h).

The following generalized version of light affine phase semantics is needed in the next subsection.

Definition 7.34 A *generalized light affine phase structure* (M, Cl, F, H) is an affine phase structure (M, Cl) with two functions $F : \mathcal{P}(M) \longrightarrow \mathcal{P}(J(M))$ and $H : \mathcal{P}(M) \longrightarrow \mathcal{P}(M)$ such that

- $X \subseteq Y$ implies $F(X) \subseteq F(Y)$ and $H(X) \subseteq H(Y)$;
- $\varepsilon \in X$ implies $\varepsilon \in F(X)$ and $\varepsilon \in H(X)$;
- $H(X) \cdot H(Y) \subseteq H(X \cdot Y)$ for any $X, Y \subseteq M$;
- $F(X) \subseteq Cl(H(X))$

Given such (M, Cl, F, H), one defines

- $!X = Cl(F(X))$;
- $\S X = Cl(H(X))$.

It is clear that a light affine phase structure (M, Cl, f, h) is a special case; just define $F(X) = \{f(x) | x \in X\}$ and $H(X) = \{h(x) | x \in X\}$.

Lemma 7.35 *For every generalized light phase structure (M, Cl, F, H) and $X, Y \subseteq M$, we have:*

1. $X \subseteq Y$ *implies* $!X \subseteq !Y$.
2. $\mathbf{1} \subseteq !\mathbf{1}$.
3. $!X \subseteq !X \otimes !X$.
4. $X \subseteq Y$ *implies* $\S X \subseteq \S Y$.
5. $\S X \otimes \S Y \subseteq \S(X \otimes Y)$.
6. $!X \subseteq \S X$.

Proof. All are more or less immediate from the definition. ■

Definition 7.36 A *light affine phase model* is a light affine phase structure with a valuation. In a light affine phase model, the valuation v is extended by:

- $v(!A) = !(v(A))$; $v(\S A) = \S(v(A))$.

The soundness for **ILAL** follows from Lemma 7.35. In the next subsection, we shall show the completeness, the cut-elimination and the finite model property for **ILAL** once for all.

7.3.2 The Finite Model Property for ILAL

Let us first define the canonical model for **ILAL**.

Definition 7.37 Given a formula A, the *canonical model* $\mathbf{ILAL}^\bullet[A] = (M_0, Cl_0, v_0, f_0, h_0)$ is defined analogously to $\mathbf{IAL}^\bullet[A]$, with some modifications:

- M_0 is the free commutative monoid generated by the subformulas of A and $\mathbf{0}$.
- For a sequent $\Gamma \vdash C$ which consists of subformulas of A and $\mathbf{0}$,
$$[\![\Gamma \vdash C]\!] = \{\Sigma \in M_0 | \ \Sigma, \Gamma \vdash C \text{ is } \textit{cut-free} \text{ provable in } \mathbf{ILAL}\}.$$
- Let $\mathcal{X}$ be the set of outer values of all sequents which consist of subformulas of A and $\mathbf{0}$, and define Cl_0 by
$$Cl_0(X) = \bigcap\{Y \mid X \subseteq Y, Y \in \mathcal{X}\}.$$

- Define f_0 and h_0 by:

$$
\begin{array}{rcll}
f_0(\emptyset) & = & \emptyset; & \\
f_0(B) & = & !B, & \text{if } !B \text{ is a subformula of } A; \\
 & = & \mathbf{0} & \text{otherwise;} \\
f_0(\Gamma) & = & \mathbf{0} & \text{if } |\Gamma| \geq 2. \\
h_0(\emptyset) & = & \emptyset; & \\
h_0(B) & = & \S B, & \text{if } \S B \text{ is a subformula of } A; \\
 & = & !B, & \text{if } \S B \text{ is not a subformula of } A\text{, but } !B \text{ is;} \\
 & = & \mathbf{0}, & \text{otherwise;} \\
h_0(B_1, \dots, B_n) & = & h_0(B_1), \dots, h_0(B_n). &
\end{array}
$$

- $v(\alpha) = [\![\alpha]\!]$ for each atomic formula α.

Lemma 7.38 **ILAL$^\bullet$**$[A]$ *is a light affine phase model.*

Proof. It is sufficient to check that (f_0, h_0) meets the conditions of Definition 7.34. As for condition 1, $!B \preceq !B \cdot !B$ holds as before, and it also holds that $\mathbf{0} \preceq \mathbf{0} \cdot \mathbf{0}$, since $\mathbf{0}$ is the least element with respect to $\preceq$. Conditions 2 and 3 are evident. As for condition 4 prove the following by induction on the length of the cut-free proof:

- If $\S B, \Gamma \vdash C$ is cut-free provable in **ILAL**, then so is $!B, \Gamma \vdash C$.

When $f_0(\Gamma) = \mathbf{0}$, then clearly we have $f_0(\Gamma) \preceq h_0(\Gamma)$. Otherwise, we have either $f_0(\Gamma) = h_0(\Gamma) = !B$ or $f_0(\Gamma) = !B \preceq \S B = h_0(\Gamma)$. ■

The main lemma of [Oka96, Oka99] can be accommodated for **ILAL**.

Lemma 7.39 *For any subformula B of A, $B \in v_0(B) \subseteq [\![B]\!]$. In particular, if $\varepsilon \in v_0(A)$, A is cut-free provable in* **ILAL**.

Proof. By induction on B. We just check the cases for light exponentials.

(Case 1) $B \equiv !C$. By the induction hypothesis, $C \in v_0(C)$. Hence $!C \in f(v_0(C)) \subseteq v_0(!C)$. On the other hand, assume $\Gamma \in f_0(v_0(C))$. Then either $\Gamma \equiv \mathbf{0}$ or $\Gamma \equiv !D$ and $D \in v_0(C)$ for some subformula $!D$ of A. In the former case, it is clear that $\mathbf{0} \in [\![!C]\!]$. In the latter case, $D \vdash C$ is cut-free provable by the induction hypothesis. Hence $!D \vdash !C$ is cut-free provable, too.

(Case 2) $B \equiv \S C$. $\S C \in v_0(\S C)$ can be shown similarly to (Case 1). On the other hand, assume $\Gamma \in h_0(v_0(C))$. If Γ contains $\mathbf{0}$, then $\Gamma \vdash \S C$ is an axiom. Otherwise, $\Gamma \equiv !\Delta_1, \S\Delta_2$ and $\Delta_1, \Delta_2 \in v_0(C)$. By the induction hypothesis, $\Delta_1, \Delta_2 \vdash C$ is cut-free provable. Hence $!\Delta_1, \S\Delta_2 \vdash \S C$ is cut-free provable, too. ■

Lemma 7.40 **ILAL$^\bullet$**$[A]$ *has a finite co-basis.*

Proof. The same as Lemma 7.29. ■

The next task is to construct a finite model from $\mathbf{ILAL}^\bullet[A]$. At this point, however, we face an obstacle, since the light affine phase models are not closed under the quotient model construction. The main reason is that $x \sim y$ does not imply $h(x) \sim h(y)$. Typically,

$$B, C \sim B \otimes C$$

holds in $\mathbf{ILAL}^\bullet[A]$, but never

$$\S B, \S C \sim \ \S(B \otimes C).$$

Fortunately, the *generalized* light affine phase models are closed under the quotient model construction, and we should content ourselves with that.

Definition 7.41 Given a generalized light affine phase model (M, Cl, F, H, v), its *quotient* $(M_\sim, Cl_\sim, F_\sim, H_\sim, v_\sim)$ is defined as in Subsection 7.1.4, with $F_\sim$ and $H_\sim$ defined as follows:

- $F_\sim(X) = \pi \circ F \circ \pi^{-1}(X)$;
- $H_\sim(X) = \pi \circ H \circ \pi^{-1}(X)$.

With this definition, we have:

Lemma 7.42

1. *$(M_\sim, Cl_\sim, F_\sim, H_\sim, v_\sim)$ is a generalized light affine phase model.*
2. *When X is a closed subset of M, $\pi(!X) = !\pi(X)$ and $\pi(\S X) = \S\pi(X)$.*

Proof. As for 1, let us just check that (1a) $F_\sim(X) \subseteq J(M_\sim)$, and (1b) $F_\sim(X) \subseteq Cl_\sim(H_\sim(X))$ for any $X \subseteq M_\sim$. Others are straightforward.

(1a) Since $F \circ \pi^{-1}(X) \subseteq J(M)$, we have $\pi \circ F \circ \pi^{-1}(X) \subseteq \pi \circ J(M)$. It is easily checked that $\pi \circ J(M) \subseteq J(M_\sim)$.

(1b) We have $F \circ \pi^{-1}(X) \subseteq Cl \circ H \circ \pi^{-1}(X)$. Hence,

$$\begin{array}{rcl} \pi \circ F \circ \pi^{-1}(X) & \subseteq & \pi \circ Cl \circ H \circ \pi^{-1}(X) \\ & = & Cl_\sim \circ \pi \circ H \circ \pi^{-1}(X) \\ & = & Cl_\sim \circ H_\sim(X). \end{array}$$

As for 2, we have:

$$\begin{array}{rcl} \S(\pi(X)) & = & Cl_\sim \circ \pi \circ H \circ \pi^{-1} \circ \pi(X) \\ & = & Cl_\sim \circ \pi \circ H(X) \\ & = & \pi \circ Cl \circ H(X) \\ & = & \pi(\S X). \end{array}$$

Similarly for !. ■

Theorem 7.43 *Let (M, Cl, f, h, v) be a light affine phase model. Then, every formula A of* **ILAL** *is satisfied in (M, Cl, f, h, v) iff it is satisfied in its quotient $(M_\sim, Cl_\sim, f_\sim, h_\sim, v_\sim)$.*

Proof. Similar to Theorem 7.19, using the previous lemma. ∎

To put things together, we obtain:

Theorem 7.44 *For any formula A, the following statements are equivalent:*

1. *A is provable in* **ILAL**.
2. *A is satisfied in all light affine phase models.*
3. *A is satisfied in all generalized light affine phase models.*
4. *A is satisfied in all finite generalized affine phase models.*
5. *A is satisfied in the quotient of* $\mathbf{ILAL}^\bullet[A]$.
6. *A is satisfied in* $\mathbf{ILAL}^\bullet[A]$.
7. *A is cut-free provable in* **IAL**.

Proof. The implication $(1 \Rightarrow 3)$ is by the soundness argument. $(3 \Rightarrow 4)$ holds trivially, and $(4 \Rightarrow 5)$ holds by Lemma 7.40 and Proposition 7.20. $(5 \Rightarrow 6)$ holds by Theorem 7.43, and $(6 \Rightarrow 7)$ holds by Lemma 7.39. $(7 \Rightarrow 1)$ is trivial. Finally, $(3 \Rightarrow 2)$ holds because 2 is a special case of 3, and $(2 \Rightarrow 6)$ holds because $\mathbf{ILAL}^\bullet[A]$ is a (non-generalized) light affine phase model. ∎

In particular, we have:

Corollary 7.45 **ILAL** *has the finite model property. Hence* **ILAL** *is decidable.*

7.4 Remarks

In this chapter, we have investigated phase semantics for Light Logic, with a special emphasis on the finite model property. Here are some remarks.

Comparison with fibred phase semantics. As was said before, light affine phase semantics is a simplification of fibred phase semantics for **ILLL** [KOSar]. Here we shall recall the definition of the latter and compare it with the former.

Definition 7.46 A *fibred phase structure* is a family $\{(M_n, Cl_n, f_n, h_n)\}_{n\geq 0}$, where for each integer $n \geq 0$,

1. (M_n, Cl_n) is a phase structure;
2. $f_{n+1} : M_{n+1} \longrightarrow M_n$ is a function;
3. $h_{n+1} : M_{n+1} \longrightarrow M_n$ is a monoid homomorphism;

4. f_n is bounded by h_n: For every $x \in M_n$ there exists $y \in M_n$ such that $y \preceq x$ and $f_n(x) \preceq h_n(y)$;
5. f_n satisfies the intermediate value property: For every $x, y \in M_n$ such that $f_n(x), f_n(y)$ are defined, there exists $z \in M_n$ such that $z \preceq x, z \preceq y$ and $f_n(x) \cdot f_n(y) = f_n(z)$.

Given a family $X = \{X_n\}_{n \geq 0}$ of closed subsets of $\{M_n\}_{n \geq 0}$, define families $!X = \{(!X)_n\}_{n \geq 0}$ and $\S X = \{(\S X)_n\}_{n \geq 0}$ as follows:

- $(!X)_n = Cl_n(f_{n+1}(X_{n+1}) \cap J_n)$;
- $(\S X)_n = Cl_n(h_{n+1}(X_{n+1}))$.

It is not necessary to consider a family of phase structures if one is only interested in completeness; just a single phase structure would do. In [KOSar], the fibred structure is rather used to give a concrete mathematical example of models which invalidate Dereliction, Digging and Monoidalness. The same applies to our light affine phase semantics. Although we have considered a non-fibred version of light affine phase semantics in this chapter, we could easily extend it to a fibred version.

Beside fibredness, light affine phase semantics differs from fibred phase semantics in the following points:

- We do not need the intermediate value property for f; this is because we do not have Exponential Isomorphism in syntax of **ILAL**.
- Instead, we need the condition $f(\varepsilon) = \varepsilon$, which corresponds to Monoidalness 2.
- The boundedness condition and the interpretation of ! are slightly simplified. This small improvement also applies to fibred phase semantics.

The simplicity of light affine phase semantics is particularly advantageous when it comes to constructing a concrete model. In Example 7.33 we have described a simple countermodel to Dereliction, Digging and Monoidalness 1 which consists of just 4 elements. It does not seem to be so easy to have such a simple construction in fibred phase semantics, with the main trouble being the intermediate value property.

Finite model property with respect to non-generalized light affine phase models. The finite model property for **ILAL** which we showed was with respect to the generalized light affine phase models. It is open whether the same holds with respect to the (non-generalized) light affine phase models.

Finite model property for other systems of Linear and Affine Logics. The quotienting method used here has a much wider range of applications. Using this method, Lafont proved the finite model property for the multiplicative additive fragment of Linear Logic, Classical Affine Logic and their noncommutative versions [Laf97]. On the other hand, we proved the finite model property for the multiplicative additive fragment of Intuitionistic Linear Logic and for Classical/Intuitionistic Linear Logic with Unrestricted Contraction in [OT99]. In the same paper, we also showed the finite model property for various systems of *Substructural Logics* ([Ono90, Ono94, Ono98]).

Extensions of light affine phase semantics. The extension of our phase semantics to the higher order versions is easily achieved along the line of [Oka96, Oka99]. It is also easy to define classical light affine phase semantics and obtain the soundness/completeness, the cut-elimination and the finite model property for **LAL**. An interesting open problem is whether light affine phase semantics can be extended to Light Set Theory with a suitable semantic proof to the cut-elimination theorem. There is no principled reason which forbids us to give such a semantic proof to cut-elimination, since it can be proved anyway by a syntactic means. It would be, however, extremely difficult, because it requires us to express the naive comprehension principle of **LST** in terms of phase semantics which rests upon the standard set theory.

Chapter 8

Conclusion

In this thesis, we have investigated various aspects of Light Logic, aiming at a better understanding of the polytime computation in the proofs-as-programs paradigm. Let us summarize our main contributions:

1. In Chapter 3, we introduced an untyped term calculus λLA which embodies the essential mechanisms of the proof system of Light Logic. It roughly amounts to an untyped bi-modal lambda calculus with certain linearity and stratification conditions. Two modal operators are introduced in such a way that the complexity of normalization becomes exactly polytime. The calculus has an advantage of simplicity over other existing term calculi for **ILAL** [Asp98, Rov00, Rov99, AR00]; in particular, λLA is free from explicit substitutions which are used in [Asp98]. Although explicit substitutions are often useful in giving fine control over the substitution operation, they complicate syntax too much and make the operational intuition of a calculus unclear. In contrast, our syntax adopts the standard notion of substitution, and henceforth succeeds in reducing 27 rewriting rules of [Asp98] to just 5, all of which have a clear operational meaning. The simplicity of λLA allows us to concentrate on critical issues in the Light Logic computation, and to prove the basic properties such as the Church-Rosser property and the subject reduction property in a highly convincing way.

   We then proved the polytime strong normalization theorem for λLA, which states that terms of λLA are normalizable in polynomial time regardless of which reduction strategy we take. Theoretically, this property suggests a sharp distinction between λLA and other polytime functional systems such as [LM93, Hof97, BNS99, Hof98, Hofar, Lei99], since in the latter, normalization is at best *weakly* polytime. Practically, the theorem guarantees us a free choice of a reduction strategy when it comes to designing realistic polytime programming languages. The choice of a reduction strategy is often crucial in language design, and the theorem shows that λLA is rather flexible in this respect.

2. In Chapter 4, we reformulated **ILAL2** as a type assignment system for λLA, and proved the subject reduction theorem. It basically means that proofs of **ILAL2** are structurally representable by terms of λLA, and cut-elimination in **ILAL2** is in full accordance with normalization in λLA. From this and the previous result, it follows that the polytime strong normalizability also holds for the proof system of **ILAL2**. Note that before our work only the polytime *weak* normalizability has been known for Light Logic, and it has been left open whether the polytime *strong* normalizability also holds or not. Our result gives a positive answer to this problem. Meanwhile, it has been known that all polytime functions are representable by proofs of **ILAL2** [Rov99].

Therefore, the above result also yields a complete characterization of the polytime functions via λLA terms: A function is polytime if and only if it is representable by a term of λLA.

3. In Chapter 5, we elaborated the detail of Light Set Theory, which had been left by [Gir98]. We introduced naive set theory **LST** based on **ILAL**. **LST** is endowed with a powerful mechanism of naive comprehension, and as a result, all recursive functions are numeralwise representable in (the modality-free fragment of) **LST**. This means that in **LST** we may speak of functions not just in terms of proofs, but also in terms of formulas. The main theorem of Chapter 5 then states that each polytime function is representable by a formula of **LST** and moreover its totality is provable in **LST**. This result, on the one hand, extends preceding works on contraction-free naive set theory [Gri81, Whi93, Shi96, Shi99] and shows that enriching contraction-free naive set theory with light exponentials leads to a richer and still consistent naive set theory in which functions are not just numeralwise representable, but also provably total as far as polytime functions are concerned. On the other hand, it extends the representability result of [Gir98, Rov99], which is concerned with *proofs*, to another form of representability which is concerned with *formulas* of **LST**.

   Our main theorem, provable totality of all polytime functions, is not a matter of surprise at all, as it is naturally expected from the representability result for **ILAL2**. Nevertheless, it is not so trivial and requires of a thorough checking, because representation-via-formulas in **LST** is not fully parallel to representation-via-proofs in **ILAL2**. For instance, addition of natural numbers is represented by a proof of
$$\mathbf{int} \multimap \mathbf{int} \multimap \mathbf{int}$$
   in **ILAL2**, but the totality statement for addition in **LST** is roughly of the form
$$x \in \mathsf{N} \multimap \S y \in \mathsf{N} \multimap \S\exists^! z \in \mathsf{N}(\langle x, y, z\rangle \in \mathsf{plus}),$$
   that is, it requires of an extra §. Although there is no problem in deriving totality of all polynomials from this formula, it should be noted that the algorithm extracted from it is surely different from the algorithm corresponding to the proof of $\mathbf{int} \multimap \mathbf{int} \multimap \mathbf{int}$ in **ILAL2**. Another difference between **LST** and **ILAL2** is that we can define sets by using fixpoints in **LST**. It allows a flat (i.e., depth-0) definition W' of the set $\{0, 1\}^*$, which admits of a flat representation of the discriminator function for it. Note that encoding of Turing machines in **ILAL2** is terribly complicated precisely because the discriminator function is not flatly representable in **ILAL2** (see [Rov99]). We can, on the other hand, take advantage of the flat discriminator in **LST** and obtain a well-structured, simple encoding of Turing machines.

4. In Chapter 6, we reformulated **LST** as a type assignment system for λLA and demonstrated the program extraction method for **LST**: Given a proof of **LST**, we can automatically extract its algorithmic content as a term of λLA. In particular, from a proof of a totality statement for a function f we can extract a term of λLA which represents f.

   On the practical side, this method suggests a possibility of designing an integrated formal system for feasible programming and verification, where a certified program can be extracted from a proof of its formal specification, and moreover an exact polytime upperbound for execution can be obtained from that proof at the same time.

   On the theoretical side, the program extraction method implies that every function provably total in **LST** is representable as a term of λLA, and therefore polytime computable. In conjunction with the main result of Chapter 5, we can conclude that the provably total functions in **LST** are exactly polytime. This property confirms the truly polytime nature of **LST**, although much has to be done to further claim that **LST** is really a suitable framework for feasible mathematics.

5. In Chapter 7, we studied phase semantics for **ILAL**. We introduced light affine phase semantics, which is a simplification of fibred phase semantics of [KOSar], and then showed the finite model property for **ILAL** (with respect to the generalized version of light affine phase semantics), by means of the quotienting method which was developed in [Laf97, OT99]. It implies decidability of **ILAL**, and thus decidability of the type inhabitation problem for λLA and **ILAL**.

Through these investigations, we have obtained a complete logical/type-theoretical characterization of polytime functions, i.e., the equivalence among

- functions computable by Turing machines in polynomial time,
- functions representable by proofs of **ILAL2**,
- functions provably total in **LST**, and
- functions representable by terms of λLA.

Moreover, this equivalence is constructive in the sense that we can *effectively* obtain one of the above representations from another. If we have a polytime Turing machine which computes the function f, then we can effectively encode f as a proof of **ILAL2** or as a formula of **LST**. If we have a proof of **ILAL2** which represents f, then we can effectively obtain a term of λLA which represents f via the proofs-as-programs interpretation. If we have a proof of the totality of f in **LST**, we can effectively obtain a term of λLA which represents f via the program extraction method. Finally, if we have a term of λLA which represents f, we can concretely define a polytime Turing machine which computes f (see Figure 3.6 in Chapter 3).

In future work, we shall investigate the following:

1. Incorporation of inductive data types as primitives in λLA, while keeping the polytime upperbound for normalization; such an extension will make programming in λLA easier, and thus make λLA more appropriate as a basis for realistic polytime functional programming languages.
2. Further development of Light Set Theory; in this thesis, we have mainly dealt with the provable totality of polytime functions, but we think that the use of **LST** is far more extensive. It would be interesting if we could formalize a proof of a nontrivial mathematical theorem in **LST**, and extract a polytime algorithm which is implicit in that proof.
3. Extension of the Light Logic approach to other complexity classes such as the polynomial space functions; an attempt has been already made in [Ter00], but it is too complicated and perhaps a better formalization could be obtained.

Bibliography

[Abr90] V. Michele Abrusci. Sequent calculus for intuitionistic linear propositional logic. In P. P. Petkov, editor, *Mathematical Logic*, pages 223–242, New York, London, 1990. Plenum Press. Proceedings of the Summer School and Conference on Mathematical Logic, honorably dedicated to the 90th Anniversary of Arend Heyting (1898–1980), Chaika, Bulgaria, 1988.

[Abr93] Samson Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111:3–57, 1993.

[And92] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[AR00] A. Asperti and L. Roversi. Intuitionistic light affine logic (proof-nets, normalization complexity, expressive power, programming notation). Ftp available at http://www.di.unito.it/ http://www.di.unito.it/r̃over, 2000.

[Asp98] A. Asperti. Light affine logic. In *Proceedings of LICS'98*, 1998.

[Bai00] P. Baillot. Stratified coherent spaces: a denotational semantics for light linear logic. Presented at the Second International Workshop on Implicit Computational Complexity, 2000.

[Bar81] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* Elsevier North-Holland, 1981.

[Bar92] H. P. Barendregt. Lambda calculi with types. In S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science, Volume 2*, pages 117–309. Oxford University Press, 1992.

[BC92] S. Bellantoni and S. Cook. New recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992.

[BNS99] S. Bellantoni, K.-H. Niggl, and H. Schwichtenberg. Ramification, modality and linearity in higher type recursion. Presented at the First International Workshop on Implicit Computational Complexity, 1999.

[Bus86] S. R. Buss. *Bounded Arithmetic.* Bibliopolis, 1986.

[Bus93] S. R. Buss. A note on bootstrapping intuitionistic bounded arithmetic. In Aczel, Simmons, and Wainer, editors, *Proof Theory*, pages 151–169. Cambridge University Press, 1993.

[CF58] H. B. Curry and R. Feys. *Combinatory Logic.* North Holland, 1958.

[Cob65] A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland, Amsterdam, 1965.

[Coo75] S. A. Cook. Feasibly constructive proofs and the propositional calculus. In *Proceedings of the 7th Annual ACM Symposium on Theory of Computing*, pages 83 – 97, 1975.

[CU93] S. Cook and A. Urquhart. Functional interpretations of feasibly constructive arithmetic. *Annals of Pure and Applied Logic*, 63:103 – 200, 1993.

[DJ99] V. Danos and J.-B. Joinet. Linear logic & elementary time. presented at ICC'99, 1999.

[DJS95] Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. On the linear decoration of intuitionistic derivations. In *Archive for Mathematical Logic*, volume 33, pages 387–412, 1995.

[DJS97] V. Danos, J.-B. Joinet, and H. Schellinx. A new deconstructive logic: linear logic. *The Journal of Symbolic Logic*, 62:755–807, 1997.

[EF99] H-D. Ebbinghaus and J. Flum. *Finite Model Theory, 2nd Edition.* Springer, 1999.

[EucBC] Euclid. *The Elements.* 3rd cent. B.C.

[Göd58] K. Gödel. Über eine bisher noch nicht benützte erweiterung des finiten standpunktes. *Dialectica*, 12:280–287, 1958. English translation in: Gödel's Works, Vol. II (Oxford Univ. Press, Oxford, 1990).

[Gen35] G. Gentzen. Untersuchungen über das logische schliessen. *Math. Zeitschrift*, 39:176 –210, 405–431, 1935. English translation in *The Collected Papers of Gerhart Gentzen*, translated by M. Szabo, North-Holland.

[Gir72] Jean-Yves Girard. *Une extension de l'interprétation de Godel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types.* PhD thesis, Univ. Paris VII, 1972. Also in the Proceedings of the Second Scandinavian Logic Symposium, North Holland, 1973.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gir91] Jean-Yves Girard. A new constructive logic: Classical logic. *Mathematical Structures in Computer Science*, 1:255–296, 1991.

[Gir95] Jean-Yves Girard. Linear logic: Its syntax and semantics. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 1–42. Cambridge University Press, 1995. Proceedings of the Workshop on Linear Logic, Ithaca, New York, June 1993.

[Gir98] J.-Y. Girard. Light linear logic. *Information and Computation*, 14(3):175–204, 1998.

[Gir99a] J.-Y. Girard. On denotational completeness. *Theoretical Computer Science*, 227:249 – 273, 1999.

[Gir99b] J.-Y. Girard. On the meaning of logical rules I: syntax vs. semantics. In U. Berger and H. Schwichtenberg, editors, *Computational Logic*, pages 215 – 272. Heidelberg Springer-Verlag, 1999.

[GJ78] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* Freeman, San Francisco, 1978.

[GLR95] Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors. *Advances in Linear Logic*, number 222 in London Mathematical Society Lecture Notes Series. Cambridge University Press, 1995.

[GLT88] Jean-Yves Girard, Yves Lafont, and P. Taylor. *Proofs and Types.* Cambridge Tracts in Theoretical Computer Science 7. Cambridge University Press, 1988.

[Gri74] V. N. Grishin. A nonstandard logic and its application to set theory (russian). In *Studies in Formalized Languages and Nonclassical Logics (Russian)*, pages 135 – 171. Izdat, Nauka, Moskow, 1974.

[Gri81] V. N. Grishin. Predicate and set-theoretic calculi based on logic without contractions. *Math. USSR. Izvestija*, 18:41–59, 1981.

[GS86] Y. Gurevich and S. Shelah. Fixed-point extensions of first-order logic. *Annals of Pure and Applied Logic*, 32:265 – 280, 1986.

[GSS92] Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. Bounded linear logic: A modular approach to polynomial time computability. *Theoretical Computer Science*, 97:1–66, 1992. Extended abstract in *Feasible Mathematics*, S. R. Buss and P. J. Scott editors, Proceedings of the MCI Workshop, Ithaca, NY, June 1989, Birkhauser, Boston, pp. 195–209.

[Gur83] Y. Gurevich. Algebras of feasible functions. In *Proc. of the 24th Symposium on Foundations of Computer Science*, pages 210–214. IEEE Computer Society Press, 1983.

[HBar] M. Hofmann and S. Bellantoni. A new "feasible" arithmetic. *Journal of Symbolic Logic*, to appear.

[Hin69] J. R. Hindley. The principal type scheme of an object in combinatory logic. *Trans. Amer. Math. Soc.*, 146:29 – 60, 1969.

[Hof] M. Hofmann. Programming languages capturing complexity classes. SIGACT News Logic Column 9.

[Hof97] M. Hofmann. An application of category-theoretic semantics to the characterisation of complexity classes using higher-order function algebras. *Bulletin of Symbolic Logic*, 3(4), 1997.

[Hof98] M. Hofmann. *Type Systems for Polynomial-Time Computation.* Habilitationsschrift, Technical University of Darmstadt, 1998.

[Hofar] M. Hofmann. Safe recursion with higher types and BCK algebra. *Annals of Pure and Applied Logic*, to appear.

[How80] W. A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–490. Academic Press, 1980.

[HP93] P. Hajék and P. Pudlák. *Metamathematics of First-Order Arithmetic.* Springer-Verlag, 1993.

[HS00] M. Hofmann and P. Scott. Realizability models for BLL-like languages. Presented at the Second International Workshop on Implicit Computational Complexity, 2000.

[HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Mass, 1979.

[Imm86] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86 – 104, 1986.

[Imm87] N. Immerman. Languages which capture complexity classes. *SIAM Journal of Computing*, 16:760 – 778, 1987.

[Jon97] N. Jones. *Computability and Complexity from a Programming Perspective.* MIT Press, 1997.

[Kop95] Alexei P. Kopylov. Decidability of linear affine logic. In D. Kozen, editor, *Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 496–504, San Diego, California, June 1995.

[KOSar] M. Kanovitch, M. Okada, and A. Scedrov. Phase semantics for light linear logic. *Theoretical Computer Science*, to appear. An extended abstract appeared in Proceedings of MFPS'97.

[KOT99] Max Kanovitch, Mitsuhiro Okada, and Kazushige Terui. Intuitionistic phase semantics is almost classical. Unpublished Manuscript, 1999.

[Kra95] J. Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory.* Cambridge University Press, 1995.

[Löb76] M. H. Löb. Embedding first-order predicate logic in fragments of intuitionistic logic. *Journal of Symbolic Logic*, 41:705 – 718, 1976.

[Laf96] Yves Lafont. The undecidability of second order linear logic without exponentials. *Journal of Symbolic Logic*, 61(2):541 – 548, 1996.

[Laf97] Yves Lafont. The finite model property for various fragments of linear logic. *Journal of Symbolic Logic*, 62(4):1202 – 1208, 1997.

[Laf01] Y. Lafont. Soft linear logic and polynomial time. Manuscript, 2001.

[Lei90] D. Leivant. Inductive definitions over finite structures. *Information and Computation*, 89:95 – 108, 1990.

[Lei93] D. Leivant. Stratified functional programs and computational complexity. In *Conference Record of the 13th Annual ACM Symposium on Principles of Programming Languages*, pages 325 – 333, 1993.

[Lei94] D. Leivant. A foundational delineation of poly-time. *Information and Computation*, 1994.

[Lei95] D. Leivant. Intrinsic theories and computational complexity. In D. Leivant, editor, *Logic and Computational Complexity*, pages 177–194. Springer-Verlag LNCS 960, 1995.

[Lei99] D. Leivant. Applicative control and computational complexity. In *Proceedings of CSL'99*, pages 82–95. Springer-Verlag, LNCS 1683, 1999.

[LM93] D. Leivant and J.-Y. Marion. Lambda calculus characterizations of poly-time. *Fundamenta Informaticae*, 19:167–184, 1993.

[LM94] D. Leivant and J.-Y. Marion. Ramified recurrence and computational complexity II: substitution and polyspace. In J. Tiuryn and L. Pacholsky, editors, *Computer Science Logic*. Springer, 1994.

[LMSS92] Patrick Lincoln, John Mitchell, Andre Scedrov, and Natarajan Shankar. Decision problems for propositional linear logic. *Annals of Pure and Applied Logic*, 56:239–311, April 1992. Also in the Proceedings of the 31th Annual Symposium on Foundations of Computer Science, St Louis, Missouri, October 1990, IEEE Computer Society Press. Also available as Technical Report SRI-CSL-90-08 from SRI International, Computer Science Laboratory.

[LSS95] Patrick Lincoln, Andre Scedrov, and Natarajan Shankar. Decision problems for second order linear logic. In D. Kozen, editor, *Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 476–485, San Diego, California, June 1995.

[Mil78] R. Milner. A theory of type polymorphism in programming. *J. Computer and Systems Sciences*, 17:348–375, 1978.

[MO00a] A. S. Murawski and C.-H. L. Ong. Can safe recursion be interpreted in light logic? Presented at the Second International Workshop on Implicit Computational Complexity, 2000.

[MO00b] A. S. Murawski and C.-H. L. Ong. Discreet games, light affine logic and ptime computation. In *Proceedings of CSL2000*, pages 427–441. Springer-Verlag, LNCS 1862, 2000.

[Oka96] Mitsuhiro Okada. Phase semantics for higher order completeness, cut-elimination and normalization proofs (extended abstract). In J.-Y. Girard, M. Okada, and A. Scedrov, editors, *ENTCS (Electronic Notes in Theoretical Computer Science) Vol.3: A Special Issue on the Linear Logic 96, Tokyo Meeting*. Elsevier-ENTCS, 1996. An earlier version is available by ftp anonymous on iml.univ-mrs.fr, in pub/okada.

[Oka99] Mitsuhiro Okada. Phase semantic cut-elimination and normalization proofs of first- and higher-order linear logic. *Theoretical Computer Science*, 227:333– 396, 1999.

[Oka01] Mitsuhiro Okada. A uniform proof for higher order cut-elimination and normalization theorem. *Theoretical Computer Science*, to appear, 2001.

[Ono90] Hiroakira Ono. Structural rules and a logical hierarchy. In P. P. Petkov, editor, *Mathematical Logic*, pages 95–104. Plenum Press, 1990. Proceedings of the Summer School and Conference on Mathematical Logic, honorably dedicated to the 90th Anniversary of Arend Heyting (1898–1980), Chaika, Bulgaria, 1988.

[Ono94] Hiroakira Ono. Semantics for substructural logics. In K. Došen and P. Schröder-Heister, editors, *Substructural logics*, pages 259–291. Oxford University Press, 1994.

[Ono98] Hiroakira Ono. Decidability and finite model property of substructural logics. In *The Tbilisi Symposium on Language, Logic and Computation 1995*, pages 263–274. CSLI Publications, 1998.

[OT99] Mitsuhiro Okada and Kazushige Terui. The finite model property for various fragments of intuitionistic linear logic. *Journal of Symbolic Logic*, 64(2):790–802, 1999.

[Pap85] C. Papadimitriou. A note on the expressive power of PROLOG. *Bulletin of EATCS*, 26:21 – 23, 1985.

[Pra65] D. Prawitz. *Natural Deduction.* Almqvist & Wiksell, Stockholm, 1965.

[Rov99] L. Roversi. A P-time completeness proof for light logics. In *Proceedings of CSL'99*, pages 469–483. Springer-Verlag, LNCS 1683, 1999.

[Rov00] L. Roversi. Light affine logic as a programming language: a first contribution. *Internatinal Journal of Foundations of Computer Science*, 11(1):113 – 152, March 2000.

[Saz80] V. Sazonov. Polynomial computability and recursivity in finite domains. *Electronische Informationsverarbeitung und Kybernetik*, 7:319 – 323, 1980.

[Sce93] Andre Scedrov. A brief guide to linear logic. In G. Rozenberg and A. Salomaa, editors, *Current Trends in Theoretical Computer Science*, pages 377–394. World Scientific Publishing Company, 1993. Also in Bulletin of the European Association for Theoretical Computer Science, volume 41, pages 154–165.

[Sch94] Harold Schellinx. *The Noble Art of Linear Decorating.* PhD thesis, Institute for Logic, Language and Computation, University of Amsterdam, 1994.

[Shi96] M. Shirahata. A linear conservative extension of zermelo-fraenkel set theory. *Studia Logica*, 56:361 – 392, 1996.

[Shi99] M. Shirahata. Fixpoint theorem in linear set theory. Unpublished Manuscript, 1999.

[Sho67] R. Shoenfield. *Mathematical Logic.* Addison-Wesley, 1967.

[ST96] H. Schwichtenberg and A. S. Troelstra. *Basic Proof Theory.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1996.

[Sta79] R. Statman. The typed λ-calculus is not elementary recursive. *Theoretical Computer Science*, 9:73 – 81, 1979.

[Tak87] G. Takeuti. *Proof Theory.* North Holland, the second edition, 1987.

[Ter00] K. Terui. Linear logical characterization of polyspace functions. Presented at the Second International Workshop on Implicit Computational Complexity, 2000.

[Ter01] K. Terui. Light affine lambda calculus and polytime strong normalization. In *Proceedings of LICS2001*, pages 209–220, 2001.

[Tro92] Anne S. Troelstra. *Lectures on Linear Logic.* CSLI Lecture Notes 29, Center for the Study of Language and Information, Stanford, California, 1992.

[TvD88] Anne S. Troelstra and Dirk van Dalen. *Constructivism in Mathematics, An Introduction, Vol. 1.* Studies in Logic and the Foundations of Mathematics 121. North Holland, 1988.

[Var82] M. Vardi. Complexity and relational query languages. In *Proc. of the 14th ACM Symposium on Theory of Computing*, pages 137 – 146, 1982.

[Wad93] P. Wadler. A syntax for linear logic. In *Proceedings of MFPS'93.* Springer Verlag, LNCS 802, 1993.

[Wel94] J. B. Wells. Typability and type-checking in the second-order calculus are equivalent and undecidable. In *Proceedings of LICS'94*, pages 176–185. IEEE, 1994.

[Whi93] R. B. White. A consistent theory of attributes in a logic without contraction. *Studia Logica*, 52:113 – 142, 1993.