Archive for Mathematical Logic manuscript No.
(will be inserted by the editor)

Kazushige Terui

Light Affine Lambda Calculus and Polynomial Time Strong Normalization

Received: date / Revised version: date –

Abstract. *Light Linear Logic* (**LLL**) and *Intuitionistic Light Affine Logic* (**ILAL**) are logics that capture polynomial time computation. It is known that every polynomial time function can be represented by a proof of these logics via the proofs-as-programs correspondence. Furthermore, there is a reduction strategy which normalizes a given proof in polynomial time. Given the latter polynomial time "weak" normalization theorem, it is natural to ask whether a "strong" form of polynomial time normalization theorem holds or not.

In this paper, we introduce an untyped term calculus, called *Light Affine Lambda Calculus* (λLA), which corresponds to **ILAL**. λLA is a bi-modal λ-calculus with certain constraints, endowed with very simple reduction rules. The main property of λLA is the polynomial time strong normalization: *any* reduction strategy normalizes a given λLA term in a polynomial number of reduction steps, and indeed in polynomial time. Since proofs of **ILAL** are structurally representable by terms of λLA, we conclude that the same holds for **ILAL**.

1. Introduction

In [9], Girard introduced *Light Linear Logic* (**LLL**) as an intrinsically polytime logical system: every polynomial time function is representable by an **LLL** proof, and every **LLL** proof (of lazy conclusions) is normalizable in polynomial time. Later on, Asperti [1] introduced a simplified system, called *Light Affine Logic*, by adding the full (unrestricted) weakening rule to **LLL**. Its intuitionistic fragment (**ILAL**) has been particularly well investigated (see [2]), because it allows a compact term notation for proofs and has clear relevance to functional programming issues.

Light Logics provide an *implicit* characterization of polynomial time; in contrast with their precursor, *Bounded Linear Logic* [10], the syntax of Light Logics do not contain any polynomials. Moreover, the characterization is *purely logical*; in contrast with those implicit characterizations of polynomial time which are mostly based on safe recursion or data ramification (see [6,

Kazushige Terui: National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, 101-8430 Tokyo, Japan. e-mail: terui@nii.ac.jp

This is a full version of the paper [21] presented at LICS 2001.

Key words or phrases: Light logics – Lambda calculus – Polynomial time

14,13,11,7]), Light Logics do not contain any built-in data types, and the characterization result is about the complexity of cut-elimination, which is of purely logical nature. It naturally extends to a consistent naive set theory, in which one can reason about polynomial time concepts [9,23]. Also notably, Light Logics are endowed with various semantics [12,3,16], which could lead to a semantic understanding of polynomial time.

An important problem remains to be settled, however. By inspecting the normalization theorem given by [9], one observes that what is actually shown is the polynomial time *weak* normalization, namely, that there is a *specific* reduction strategy which normalizes a given **LLL** proof in polynomial time. The same is true of **ILAL** [1,2]. It has been left unsettled whether the polynomial time *strong* normalization holds for these Light Logics, namely, whether *any* reduction strategy normalizes a given proof in polynomial time. The primary purpose of this paper is to give a solution to this problem.

Having such a property will be theoretically important in that it gives further credence to Light Logics as intrinsically polytime systems. It will be practically important, too. Through the Curry-Howard correspondence, the proofs of Light Logics may be considered as *feasible programs*. In this context, the property will ensure that the programs can be executed in polynomial time *independent of* evaluation strategies.

For our purpose, it is reasonable to begin with **ILAL**, because it is simpler than **LLL** as a logical system. However, the existing term calculi proposed for **ILAL** either have a complicated notion of reduction defined by more than 20 rewriting rules [1,20], or involve notational ambiguity [19, 2][1]. Therefore, it is desirable to have a simple and accurate term calculus for **ILAL**. Such a simple calculus will provide a clearer intuition on the computational aspect of Light Logics. That is our secondary purpose[2].

In this paper, we introduce a new term calculus, called *Light Affine Lambda Calculus* (λLA) (in Section 2). It amounts to a simple modification of λ-calculus with modal and let operators. The main advantage of λLA is operational simplicity; it has only five reduction rules and each rule has a clear meaning. Although λLA is untyped, all the *well-formed* terms are normalizing. We then re-introduce **ILAL** as a Curry-style type assignment system for λLA and prove the subject reduction theorem (in Section 3). There are a number of reasons for adopting a Curry-style presentation:

1. First of all, to design a truly polytime (rather than just polystep) polymorphic calculus, one has to give up a Church-style term syntax with embedded types: a universal quantifier may bind an arbitrary number

[1] See the remark in Section 9.1 of [2]. To compensate this ambiguity, the latter paper presents a proofnet syntax for **ILAL**.

[2] Another approach is to identify a well-behaved fragment of **ILAL** for which the ordinary lambda calculus works perfectly well. This line of research is pursued by [4].

of type variable occurrences, and thus iterated type instantiations (Λ reductions) may easily cause exponential growth in the size of types.[3]
2. An untyped polytime calculus deserves investigation in its own right.
3. The notion of well-formedness, rather than typability, neatly captures the syntactic condition for polynomial time normalizability.
4. Last but not least, typability in **ILAL** is presumably intractable[4], while well-formedness can be checked very easily (in quadratic time).

In this setting, we prove our main theorems (in Sections 4 and 5):

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

Since proofs of **ILAL** are structurally representable by terms of **ILAL** (where formulas are erased), we can conclude that the same theorem holds for **ILAL**.

We finally discuss some related issues, such as the relationship with the safe recursion approach, polynomial time strong normalization for **LLL**, a lowerbound for normalization and decision problems for type inference (in Section 6).

2. Light Affine Lambda Calculus

In this section, we present λLA. We begin by introducing the pseudo-terms and several basic notions in 2.1, then move on to the well-formed terms in 2.2. Finally, the reduction rules are given in 2.3.

2.1. Pseudo-Terms

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

Definition 1. *The set $\mathcal{PT}$ of* pseudo-terms *is defined by the following grammar:*

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

In addition to λ-abstraction and application, we have two modal operators (!, §) and two let operators (let-!, let-§). Pseudo-terms $!t$ and $\S t$ are called *!-box* and *§-box* respectively. It is suggestive to think of them as "boxes" in the literal sense:

[3] Proofnets (of **LLL**) contain formulas. Hence proofnets themselves are not normalizable in polynomial time. A solution suggested by [9] 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.

[4] The problem is undecidable for System F in the Curry style [24].

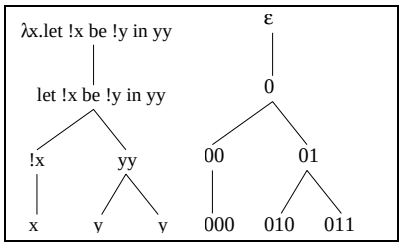


Fig. 1. Term Tree and Addresses

$!\boxed{t}$ $\S\boxed{t}$.

With boxes thus drawn, each pseudo-term is *stratified* into layers. For instance, $(\mathsf{let}\ !x =!y\ \mathsf{in}\ \S!(\lambda z.zx))z$ is stratified into three layers as follows:

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

In the sequel, symbol † stands for either ! or §. Pseudo-terms $(\lambda x.t)$ and $(\mathsf{let}\ \ \dagger x = u\ \mathsf{in}\ t)$ *bind* each occurrence of x in t. As usual, pseudo-terms are considered up to α-equivalence, and subject to the *variable convention*; namely, the bound variables are chosen to be different from the free variables, so that variable clash is never caused by substitution. The notation $t[u/x]$ denotes 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 all free variable occurrences in t.

Each pseudo-term t can be represented as a *term tree*, and each subterm occurrence u in t can be 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 $(\lambda x.\mathsf{let}\ !y =!x\ \mathsf{in}\ yy)$ and the addresses in it are illustrated in Figure 1. We write $w \sqsubseteq v$ when w is a prefix of v.

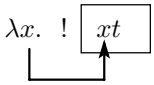
The *size* $|t|$ of a pseudo-term t is the number of nodes in its term tree. Since pseudo-terms are untyped, $|t|$ is not significantly different from the length of the string representation of t. Given a pseudo-term t and an address w, the *depth* of w in t is the number of !-boxes and §-boxes enclosing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

A *context* Φ is a pseudo-term with a 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.

2.2. Well-Formed Terms

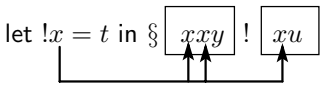
The well-formed terms are obtained by imposing certain conditions on the pseudo-terms. To begin with, let us give an informal description.

First, the base of our calculus is *affine*: each λ may bind at most one free variable. That is to say, λ is in charge of abstraction and cancellation, but not of duplication. Moreover, the λ-binding relation may not cross a box. For instance, the binding below is not allowed:



The effect is that λ- and !-components do not interleave each other.

Second, let operators must respect *stratification*: for each pair of a let operator and a variable it binds, the binding relation must cross *exactly one* box, as follows:



To see why this condition is needed, define

$$\omega_1 \equiv \lambda x.(\mathsf{let}\ !y = x\ \mathsf{in}\ y!y),$$
$$\omega_2 \equiv \lambda x.(\mathsf{let}\ !y = x\ \mathsf{in}\ !(y(!y)).$$

These pseudo-terms clearly violate the above condition; in ω_1 the binding relation indicated below does not cross a box, while in ω_2 the one indicated below crosses two boxes.

These terms causes non-termination of reductions. In fact, anticipating the reduction rules in Figure 2, we see that the following pseudo-terms are not normalizing:

$$\begin{aligned}
\omega_1!\omega_1 &\xrightarrow{(\beta)} \mathsf{let}\ !y =!\omega_1\ \mathsf{in}\ y!y \\
&\xrightarrow{(!)} \omega_1!\omega_1 \longrightarrow \cdots \\
\omega_2!\omega_2 &\xrightarrow{(\beta)} \mathsf{let}\ !y =!\omega_2\ \mathsf{in}\ !(y!y)) \\
&\xrightarrow{(!)} !(\omega_2!\omega_2) \longrightarrow \cdots
\end{aligned}$$

Note that the above condition reflects the lack of the *dereliction* and *digging* principles in Light Logics:

- Dereliction: $!A \multimap A$,
- Digging: $!A \multimap !!A$.

Third, a !-box may contain at most one free variable. This is to rule out a pseudo-term like

$$2 \equiv \lambda x.(\mathsf{let}\ !y = x\ \mathsf{in}\ !\boxed{yy}),$$

which causes an exponential explosion:

$$\begin{aligned} \mathsf{2}^n(!z) &\xrightarrow{(\beta)} \mathsf{2}^{n-1}(\mathsf{let}\ !y = !z\ \mathsf{in}\ !yy) \\ &\xrightarrow{(!)} \mathsf{2}^{n-1}(!zz) \\ &\xrightarrow{(\beta)} \mathsf{2}^{n-2}(\mathsf{let}\ !y = !zz\ \mathsf{in}\ !yy) \\ &\xrightarrow{(!)} \mathsf{2}^{n-2}(!(zz)(zz)) \\ &\longrightarrow^* !(\underbrace{zz\cdot\cdots\cdot z}_{2^n\ times}). \end{aligned}$$

This condition intuitively means that a !-box is to be duplicated, but not to duplicate another !-box. It reflects the lack of the *monoidalness* in Light Logics (which distinguishes Light Logics from *Elementary Logics* [9,8]):

– Monoidalness: $!A \otimes !B \multimap !(A \otimes B)$.

Since the last condition is too severe, we need another modal operator § as a compensation. A §-box may contain an arbitrary number of free variables. Instead, let-§ may not bind more than one free variable. In other words, a §-box is to duplicate something, but not to be duplicated. Two modalities ! and § are related by the following conditions: a let-! operator may bind a free variable in a §-box (corresponding to the *weak dereliction* principle $!A \multimap \S A$ that holds in Light Logics), whereas a let-§ operator may not bind a free variable in a !-box.

Let us now give a formal definition. Here, it is convenient to classify variables into three groups: *undischarged*, !-*discharged*, and §-*discharged* variables. These are to be bound by λ-abstraction, let-! operator and let-§ operator, respectively. In the sequel, $\uplus$ stands for disjoint union.

Definition 2. *Let X, Y, Z range over the finite sets of variables. The 4-ary relation $t \in \mathcal{T}_{X,Y,Z}$ (saying that t is a well-formed term with undischarged variables from X, !-discharged variables from Y and §-discharged variables from 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. $\lambda x.t \in \mathcal{T}_{X,Y,Z} \iff t \in \mathcal{T}_{X \uplus \{x\},Y,Z},\ FO(x,t) \leq 1$.
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. $\S t \in \mathcal{T}_{X,Y,Z} \iff t \in \mathcal{T}_{Y \uplus Z,\emptyset,\emptyset}$.
6. $\mathsf{let}\ !x = t\ \mathsf{in}\ u \in \mathcal{T}_{X,Y,Z} \iff t \in \mathcal{T}_{X,Y,Z},\ u \in \mathcal{T}_{X,Y \uplus \{x\},Z}$.
7. $\mathsf{let}\ \S x = t\ \mathsf{in}\ u \in \mathcal{T}_{X,Y,Z} \iff t \in \mathcal{T}_{X,Y,Z}, u \in \mathcal{T}_{X,Y,Z \uplus \{x\}},\ FO(x,u) \leq 1$.

We say that t is a well-formed term*, or simply a* term*, if $t \in \mathcal{T}_{X,Y,Z}$ for some X, Y and Z.*

Note that clause 2 above embodies the affinity of λ-abstraction, clause 4 embodies the no-more-than-one-variable condition on !-boxes. In the meantime, the statement 3 of the next lemma ensures that the stratification condition on !- and §- boxes is satisfied.

Lemma 1. *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 2. (Substitution)

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

Example 1. We write $\lambda^! x.t$ to denote $\lambda x.\mathsf{let}\ !y = x\ \mathsf{in}\ t[y/x]$ with y fresh.

1. While ω_1 and ω_2 in the previous subsection are not well-formed, $\omega_3 \equiv \lambda x.(\mathsf{let}\ !y = x\ \mathsf{in}\ \S yy)$ is well-formed.
2. For each natural number n, we have *Church numeral* $\overline{n} \in \mathcal{T}$ defined by
$$\overline{n} \equiv \lambda^! x.\S\lambda z.\underbrace{(x \cdots (x}_{n\ times}\, z)\cdots).$$
3. For each word $w \equiv i_0 \cdots i_n \in \{0,1\}^*$, we have $\overline{w} \in \mathcal{T}$ defined by
$$\overline{w} \equiv \lambda^! x_0.\lambda^! x_1.\S\lambda z.(x_{i_0} \cdots (x_{i_n} z)\cdots).$$
Observe that $\overline{n}$'s and $\overline{w}$'s are of depth 1.
4. $Suc \equiv \lambda n.\lambda^! f.\mathsf{let}\ \S h = n!f\ \mathsf{in}\ \S\lambda y.f(hy) \in \mathcal{T}$.

Remark 1. As discussed in Section 3 of [1], a naive use of box notation causes ambiguity, and in conjunction with naive substitution, it causes a disastrous effect on complexity.

While Asperti solved this problem by using a more sophisticated box notation $\S(t)[u_1/x_1,\ldots,u_n/x_n]$, our solution is more implicit and based on a 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 $(\mathsf{let}\ !x = y\ \mathsf{in}\ \S(txx))$ 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 **LLL**; the former contracts !-formulas, while the latter contracts discharged formulas.

Remark 2. There is a quadratic time algorithm for well-formedness checking. 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 1 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 2). Therefore the algorithm runs in time $O(n^2)$, given a term of size n.

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

Fig. 2. Reduction Rules

2.3. Reduction Rules

Definition 3. *The* reduction rules *of* λLA *are given in Figure 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 the address w or the rule (r), or both, are irrelevant, we write as $t \xrightarrow{(r)} u$, $t \xrightarrow{w} u$ and $t \longrightarrow u$, respectively. The *depth* of a reduction is the depth of its redex.

A finite sequence σ of addresses $w_0, \ldots, w_{n-1}$ 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 $t_1, \ldots, t_{n-1}$ 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 an application of rule (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 $|\sigma|$.

The well-formed terms are closed under reduction:

Proposition 1. *If* $t \in \mathcal{T}_{X,Y,Z}$ *and* $t \longrightarrow u$*, then* $u \in \mathcal{T}_{X,Y,Z}$*.*

Proof. We prove the following by induction on Φ: if $\Phi[t] \in \mathcal{T}_{X,Y,Z}$ and $\Phi[t] \to \Phi[u]$, then

(1) $\Phi[u] \in \mathcal{T}_{X,Y,Z}$ and
(2) $FO(x, \Phi[u]) \leq FO(x, \Phi[t])$ for every $x \in X \cup Z$.

When $\Phi \equiv \bullet$ and the reduction is of the form

$$t \equiv \mathsf{let}\ !x = !v_1\ \mathsf{in}\ v_2 \xrightarrow{(!)} v_2[v_1/x] \equiv u,$$

then $v_2 \in \mathcal{T}_{X,Y\uplus\{x\},Z}$, $v_1 \in \mathcal{T}_{Y,\emptyset,\emptyset}$ and $FO(v_1) \leq 1$. Hence $v_2[v_1/x] \in \mathcal{T}_{X,Y,Z}$ by Lemma 2. (2) is obvious since no variable in v_1 belongs to $X \cup Z$.

If $\Phi \equiv \lambda y.\Phi'$, then $\Phi'[t] \in \mathcal{T}_{X\uplus\{y\},Y,Z}$ and $FO(y, \Phi'[t]) \leq 1$. By the induction hypothesis, $\Phi'[u] \in \mathcal{T}_{X\uplus\{y\},Y,Z}$ and $FO(y, \Phi'[u]) \leq 1$. Therefore $\lambda y.\Phi'[u] \in \mathcal{T}_{X,Y,Z}$. (2) is obvious.

Other cases are similar. ■

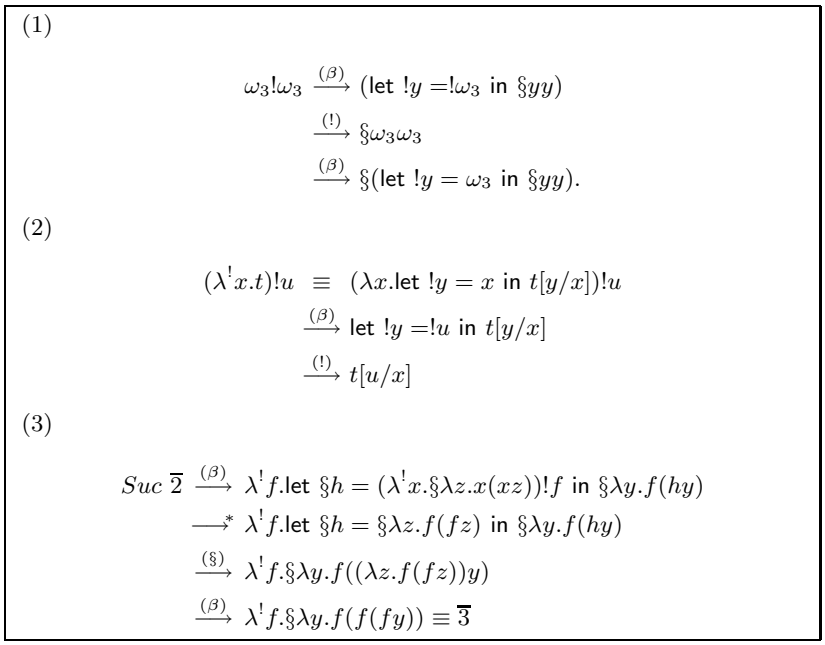


Fig. 3. Examples of normalization

Example 2.

1. We have seen in Example 1 that the term $\omega_3 \equiv \lambda x.(\mathsf{let}\ !y = x\ \mathsf{in}\ \S yy)$ is well-formed. In contrast to $\omega_1!\omega_1$ and $\omega_2!\omega_2$ described before, $\omega_3!\omega_3$ is normalizing (or better, dead-locking). See Figure 3 (1).
2. $(\lambda^! x.t)!u$ reduces to $t[u/x]$ as in Figure 3 (2).
3. The term Suc in Example 1 surely represents the successor for Church numerals. In fact, the term $Suc\ \overline{2}$ reduces to $\overline{3}$, as illustrated in Figure 3 (3).

Remark 3. The stratified structure of a term is strictly preserved by reduction. In particular, the depth of a term never increases, since in reduction rules (β), (§) 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 (§) are strictly size-decreasing, since they never involve duplication. (*com*) preserves the size. The only reduction rule that is potentially size-increasing is (!). Nevertheless, it is also strictly size-decreasing at the depth where the redex is located. The size increases only at deeper layers.

3. Type Assignment

The purpose of this section is to present *Intuitionistic Light Affine Logic* (**ILAL**) as a type assignment system for λ_{LA} and to prove the subject reduction theorem.

In 3.1, we introduce **ILAL** in a sequent calculus style. It is, however, difficult to prove the subject reduction theorem directly for it. The main difficulty is that it does not satisfy the subterm typability. To overcome this, we introduce a natural deduction system $\mathbf{ILAL}_N$ in 3.2. It is equivalent to **ILAL** as far as *closed* terms are concerned, and it does satisfy the subterm typability. Once having defined the suitable formalism $\mathbf{ILAL}_N$, it is easy to prove the subject reduction theorem. This is accomplished in 3.3.

3.1. **ILAL** *as a Type Assignment System*

Let α, β range over the type variables.

Definition 4. *The* types *(*formulas*) of* **ILAL** *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* $[A]_\S$.

In the sequel, $\dagger^n A$ abbreviates $\underbrace{\dagger \cdots \dagger}_{n\ times}\ A$.

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 $x_1 : A_1, \ldots, x_n : A_n$ in which all types are undischarged. 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 5. *The* type inference rules *of* **ILAL** *are given in Figure 4. A pseudo-term* t *is* typable *in* **ILAL** *if* $\Gamma \vdash t : A$ *is derivable for some* Γ *and* A *by those inference rules. Likewise, a pseudo-term* t *is* of type A *if* $\vdash t : A$ *is derivable.*

As expected, we have:

Proposition 2. *Every typable pseudo-term is a term. More exactly, if* $\overrightarrow{x} : \overrightarrow{A}, \overrightarrow{y} : \overrightarrow{[B]_!}, \overrightarrow{z} : \overrightarrow{[C]_\S} \vdash t : D$, *is derivable, then* $t \in \mathcal{T}_{\{\overrightarrow{x}\},\{\overrightarrow{y}\},\{\overrightarrow{z}\}}$.

Proof. By induction on the length of the typing derivation. In the case of (Cut) and $(\multimap l)$, apply Lemma 2(1). ■

Example 3. Let $\mathbf{int} \equiv \forall\alpha.!(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha)$ and $\mathbf{bint} \equiv \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha)$. Then we have $\vdash \overline{n} : \mathbf{int}$ for each $n \in N$ and $\vdash \overline{w} : \mathbf{bint}$ for each $w \in \{0,1\}^*$.

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

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

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

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

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

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

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

Fig. 4. Type Assignment System **ILAL**

An example of an untypable well-formed term is $\omega_3!\omega_3$. To see the reason, define the *erasure* of a λLA term to be a λ-term obtained by applying the following operations as much as possible:

$$\dagger u \mapsto u,$$
$$\mathsf{let}\ \dagger x = u\ \mathsf{in}\ t \mapsto t[u/x].$$

As easily seen, if a term is typable in **ILAL**, then its erasure is typable in System F (in the Curry style). Now, $\omega_3!\omega_3$ cannot be typed in **ILAL**, because its erasure is $(\lambda x.xx)(\lambda x.xx)$, a term which cannot be typed in System F.

Remark 4. Unlike the ordinaly lambda calculus, λLA does not need types to ensure normalization. Nevertheless, types are useful in several ways:

- Types are used to avoid *deadlocks*, such as $(\dagger t)u$ and $\mathsf{let}\ \dagger x = (\lambda x.t)\ \mathsf{in}\ u$.
- Some types, typically data types such as **int** and **bint**, constrain the shape of normal terms. For instance, all normal terms of type **int** are of the form $\overline{n}$ for some $n \geq 0$ (or $\lambda x.(\mathsf{let}\ !z = x\ \mathsf{in}\ \S z)$, which may be seen as an η-variant of $\overline{1}$). More generally, all normal terms of type $\S^k$**int** are of the form $\S^k\overline{n}$ (or an η-variant of $\S^k\overline{1}$).
- *Lazy* types, i.e., those which do not contain a negative occurrence of $\forall$, constrain the depth of normal terms. 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 the derivation. Thus all uses of the ! and § inference rules in the derivation are recorded within A. Hence the depth of t is necessarily bounded by the depth of A. Note that the standard data types such as **int**, **bint**, booleans and lists of booleans are all lazy.

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

The expressive power of **ILAL**, hence of λ_{LA}, is witnessed by:

Theorem 1 (Girard[9], Roversi[19]). *Every function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ that is computable in polynomial time is represented by a term of type* $\mathbf{bint} \multimap \S^d\mathbf{bint}$ *for some d.*

See [2] for a detailed exposition. Neergaard and Mairson [18] improve the above theorem by showing that a function in $DTIME[n^k]$ can be represented by a proof-net of depth $O(\log k)$. The converse will be taken up in Section 5.

3.2. Natural Deduction System $\mathbf{ILAL}_N$

ILAL does not satisfy the subterm typability; for example, $\S(xx)$ can be typed as $x:[A]_! \vdash \S(xx) : \S A$ with $A \equiv \forall\alpha.\alpha$, but it is clear that its subterm xx cannot be typed. This makes difficult to prove the subject reduction theorem directly for **ILAL**. For this reason, we introduce a type system $\mathbf{ILAL}_N$ in natural deduction style, which does satisfy the subterm typability. Then we show that **ILAL** and $\mathbf{ILAL}_N$ are equivalent as far as closed terms are concerned.

The inference rules of $\mathbf{ILAL}_N$ are given in Figure 5.

Lemma 3. *If $x:\mu, \Gamma \vdash t:A$ is derivable in $\mathbf{ILAL}_N$ and $x \notin FV(t)$, where μ is either a nondischarged or a discharged type, then $\Gamma \vdash t:A$ is derivable in $\mathbf{ILAL}_N$. The same property holds for* **ILAL***, too.*

Proof. By induction on the derivation. ■

Lemma 4. *The following rules are derivable in $\mathbf{ILAL}_N$:*

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

$$\frac{\Gamma \vdash u:A \quad x:A, \Gamma \vdash t:C}{\Gamma \vdash t[u/x]:C}\ (Cut)$$

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

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

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

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

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

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

Fig. 5. Natural Deduction System $\mathbf{ILAL}_N$

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

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.
(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 show the critical case. Suppose that $t \equiv !t'$ and the last inference rule in the derivation is of the form:

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

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

Lemma 5. *If $\Gamma \vdash t : A$ is derivable in* **ILAL***, then it is also derivable in* $\mathbf{ILAL}_N$.

Proof. By induction on the derivation, using derived rules $(Weak)$, $(Cntr)$ and (Cut) in Lemma 4. ■

A *variable substitution* is a function θ from the set of term variables to itself. $t\theta$ denotes the term obtained by replacing each free variable x in t with $\theta(x)$. $\Gamma\theta$ denotes the set of declarations obtained from Γ by replacing each variable x with $\theta(x)$. 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 6. *Suppose that $\Gamma \vdash t:A$ is derivable in* $\mathbf{ILAL}_N$*. Then there are Γ', t' and a variable substitution θ such that*

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

Proof. By induction on the derivation. We only deal with 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 **ILAL**,
- $\Gamma_1'\theta_1 \equiv \Gamma_2'\theta_2 \equiv \Gamma$, $t'\theta_1 \equiv t$ 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, $\Gamma_1', \Gamma_2' \vdash t'u' : B$ is derivable in **ILAL**. A suitable variable substitution θ is defined by

$$\begin{aligned} \theta(x) &= \theta_1(x), \quad \text{if } x \in FV(\Gamma_1'); \\ &= \theta_2(x), \quad \text{otherwise.} \end{aligned}$$

(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 **ILAL**,
- $\theta(x_1) = \cdots = \theta(x_n) = x$, $\Gamma'\theta \equiv \Gamma$ 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 3, $x_i:A, \Gamma' \vdash t':B$ is derivable in **ILAL**. Rename x_i as x. Then we can apply $(\multimap r)$ to derive $\Gamma' \vdash \lambda x.(t'[x/x_i]):A \multimap B$ in **ILAL**.
(Case 3) The last inference is $(!E)$ of the form:

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

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 **ILAL**,
- $\theta(x_1) = \cdots = \theta(x_n) = x$, $\Gamma_1'\theta_1 \equiv \Gamma_2'\theta_2 \equiv \Gamma$, $u'\theta_1 \equiv u$ 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), $\Gamma_1', \Gamma_2' \vdash \mathsf{let}\ !x = u'\ \mathsf{in}\ (t'[x/x_1, \ldots, x/x_n])$ is derivable in **ILAL**. A suitable variable substitution θ can be defined as in (Case 1). ■

As a consequence, we obtain:

Theorem 2 (Equivalence between ILAL and ILAL$_N$). *$\vdash t : A$ is derivable in* **ILAL** *if and only if it is derivable in* **ILAL**$_N$.

Proof. By Lemma 5 and Lemma 6. ■

3.3. Subject Reduction Theorem

We prove the subject reduction theorem for **ILAL**$_N$ by employing the technique described in [5]. The following definition is a refinement of Definition 4.2.1 in [5].

For any set Γ of declarations, define a binary relation $>_\Gamma$ on types by

$$\begin{aligned} &\forall\alpha.A >_\Gamma A[B/\alpha], \\ &\quad A >_\Gamma \forall\alpha.A, \quad \text{if } \alpha \notin FV(\Gamma). \end{aligned}$$

Denote the reflexive transitive closure of $>_\Gamma$ by $\geq_\Gamma$. It is easy to see that whenever $\Gamma \vdash t : A$ and $A \geq_\Gamma B$, $\Gamma \vdash t : B$ is derivable.

Lemma 7 (Generation lemma). *The following hold for* **ILAL**$_N$*:*

1. $\Gamma \vdash x : A \Longrightarrow x : B \in \Gamma$ *for some* $B \geq_\Gamma A$.
2. $\Gamma \vdash \lambda x.t : A \Longrightarrow x : B, \Gamma \vdash t : C$ *for some* B *and* C *such that* $B \multimap C \geq_\Gamma A$, *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* $C \geq_\Gamma A$.
4. $\Gamma \vdash \mathsf{let}\ !x = u\ \mathsf{in}\ t : A \Longrightarrow \Gamma \vdash u :!B$ *and* $x : [B]_!, \Gamma \vdash t : C$ *for some* B *and* $C \geq_\Gamma A$.
5. $\Gamma \vdash !t : A \Longrightarrow \Delta \vdash t : C$ *for some* C *and* Δ *such that* $!C \geq_\Gamma A$ *and* $[\Delta]_! \subseteq \Gamma$, *and* $FO(t) \leq 1$.
6. $\Gamma \vdash \mathsf{let}\ \S x = u\ \mathsf{in}\ t : A \Longrightarrow \Gamma \vdash u : \S B$ *and* $x : [B]_\S, \Gamma \vdash t : C$ *for some* B *and* $C \geq_\Gamma A$, *and* $FO(x, t) \leq 1$.
7. $\Gamma \vdash \S t : A \Longrightarrow \Delta, \Sigma \vdash t : C$ *for some* C, Δ *and* Σ *such that* $\S C \geq_\Gamma A$ *and* $[\Delta]_!, [\Sigma]_\S \subseteq \Gamma$.

Proof. By induction on derivations. Note that $(\forall I)$ and $(\forall E)$ are the only inference rules that do not change the subject t. These rules are handled by means of $\geq_\Gamma$. ■

Theorem 3 (Subject Reduction for ILAL$_N$). *If $\Gamma \vdash t : A$ is derivable in* **ILAL**$_N$ *and $t \longrightarrow u$, then $\Gamma \vdash u : A$ is derivable in* **ILAL**$_N$.

Proof. The proof is parallel to that of Proposition 1. We prove the following by induction on Φ: if $\Gamma \vdash \Phi[t] : A$ is derivable in **ILAL**$_N$ and $\Phi[t] \to \Phi[u]$, then

(1) $\Gamma \vdash \Phi[u] : A$ and
(2) $FO(x, \Phi[u]) \leq FO(x, \Phi[t])$ for each x such that either $x : A \in \Gamma$ or $x : [A]_{\S} \in \Gamma$ for some A.

Suppose that $\Phi \equiv \bullet$ and the reduction is of the form

$$t \equiv \mathsf{let}\ !x = !v_1\ \mathsf{in}\ v_2 \xrightarrow{(!)} v_2[v_1/x] \equiv u.$$

By Generation Lemma (4) and (5), we have:

$x : [B]_!, \Gamma \vdash v_2 : C$ for some B and $C \geq_\Gamma A$,
$\Delta \vdash v_1 : D$ for some Δ and D such that $[\Delta]_! \subseteq \Gamma$ and $D \geq_\Gamma B$.

From the latter, it follows that $\Delta \vdash v_1 : B$. By $(Cut_!)$ of Lemma 4, we derive $\Gamma \vdash v_2[v_1/x] : C$, and thus $\Gamma \vdash v_2[v_1/x] : A$. (2) is obvious, because any free variable in v_1 is !-discharged in Γ.

When t is a (β) redex, argue as in the proof of Theorem 4.2.5 in [5]. Other cases are similar. ■

Corollary 1 (Subject Reduction for ILAL). *If $\Gamma \vdash t : A$ is derivable in* **ILAL** *and $t \longrightarrow u$, then $\Gamma \vdash u : A$ is derivable in* **ILAL**.

Proof. The general case is reducible to the case of closed terms, by using the following facts:

$$x : A, \Gamma \vdash t : B \Longleftrightarrow \Gamma \vdash \lambda x.t : A \multimap B$$
$$x : [A]_\dagger, \Gamma \vdash t : B \Longleftrightarrow y : \dagger A, \Gamma \vdash \mathsf{let}\ \dagger x = y\ \mathsf{in}\ t : B.$$

Hence the corollary follows from Theorem 2 and Theorem 3. ■

4. Proving the Polystep Strong Normalization Theorem

In this section, we prove the polystep strong normalization theorem. The basic idea is very simple: to reduce polystep strong normalization to polystep weak normalization, by showing that any reduction sequence can be transformed (*standardized*) into a *longer* one which is subject to the outer-layer-first strategy. Since the latter sequence is polynomially bounded by Girard-Asperti's result [9, 1], we can conclude that any sequence is polynomially bounded as well. Unfortunately, this argument does not work for λLA itself, since standardization in λLA may shorten the reduction sequence. To avoid this, we first introduce an auxiliary term calculus, and show any reduction sequence in λLA can be translated into another in the auxiliary calculus in 4.1. Then we show the standardization theorem in 4.2. Finally, we accommodate Girard-Asperti's polynomial time weak normalization theorem to our setting in 4.3.

4.1. An Extended Calculus with Explicit Weakening

Suppose that $t \xrightarrow{(r)} u$ and consider the following reduction sequence of length two:

$$(\lambda x.y)!t \xrightarrow{(r)} (\lambda x.y)!u \xrightarrow{(\beta)} y.$$

It is not outer-layer-first. If one *standardizes* it, i.e., transforms it into an outer-layer-first one, then the result is

$$(\lambda x.y)!t \xrightarrow{(\beta)} y,$$

a shorter reduction sequence of length one, and the standardization argument breaks down. The failure may be ascribed to *implicit* weakening involved by λ-abstraction. We avoid this by introducing an extended calculus where weakening is *explicit*.

The set $\mathcal{PT}^w$ of *extended pseudo-terms* is defined to be the set $\mathcal{PT}$ augmented with subexpressions of the form $\mathsf{let}\ \_ = t\ \mathsf{in}\ u$ (explicit weakening). To define a new well-formedness relation $t \in \mathcal{T}^w_{X,Y,Z}$, we modify Definition 2 as follows.

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

2' $\lambda x.t \in \mathcal{T}^w_{X,Y,Z} \iff t \in \mathcal{T}^w_{X \uplus \{x\},Y,Z},\ FO(x,t) = 1.$

6' $\mathsf{let}\ !x = t\ \mathsf{in}\ u \in \mathcal{T}^w_{X,Y,Z} \iff t \in \mathcal{T}^w_{X,Y,Z}, u \in \mathcal{T}^w_{X,Y \uplus \{x\},Z}, FO(x,u) \geq 1.$

7' $\mathsf{let}\ \S x = t\ \mathsf{in}\ u \in \mathcal{T}^w_{X,Y,Z} \iff t \in \mathcal{T}^w_{X,Y,Z},\ u \in \mathcal{T}^w_{X,Y,Z \uplus \{x\}}, FO(x,u) = 1.$

Namely, we require that λ and let-§ bind exactly one variable and let-! binds at least one variable.

(2) Add the following clause:

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

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

The reduction rules in Figure 2 are extended to $\mathcal{PT}^w$ with the following modifications:

- Extend (*com*) to the new let operator.
- Add a new reduction rule (_): $\mathsf{let}\ \_ = u\ \mathsf{in}\ t \longrightarrow t$.

Reduction rules other than (_) are called *proper*. A reduction sequence is *proper* if it consists of proper reductions.

Lemmas 1 and 2 hold for $\mathcal{T}^w$, too. In addition, we have

Proposition 3. *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 λ_{LA} terms into extended terms.

Lemma 8. *For every 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. When $t \equiv \lambda x.u$ and $FO(x,u) = 0$, define t^w to be $\lambda x.(\mathsf{let}\ \_ = x\ \mathsf{in}\ u^w)$. When $t \equiv (\mathsf{let}\ \dagger x = v\ \mathsf{in}\ u)$ and $FO(x,u) = 0$, define t^w to be $\mathsf{let}\ \dagger x = v^w\ \mathsf{in}\ (\mathsf{let}\ \_ = \S x\ \mathsf{in}\ u^w)$. Other cases are straightforward. ■

Theorem 4 (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 $|\sigma| \leq |\tau|$, $|t_0'| = O(|t_0|)$ and*

$$\begin{array}{ccc} t_0 & \xrightarrow{\sigma}{}^* & t_1 \\ {\scriptstyle *}\uparrow{\scriptstyle (\_)} & & \uparrow{\scriptstyle *}{\scriptstyle (\_)} \\ t_0' & \xrightarrow{\tau}{}^* & t_1'. \end{array}$$

The proof is based on permutation of reduction sequences. Each step of permutation is supported by the following two lemmas.

Lemma 9. *Let t_0 be an extended term. If $t_0 \xrightarrow{(\_)} t_1 \xrightarrow{(com)} t_2$, then*

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

for some t_1' and t_1''.

Proof. Let u be the contractum of the first reduction (at address w_0) and v be the redex of the second reduction (at address w_1) in t_1. One can distinguish four cases: (i) u and v are separated ($w_0 \not\sqsubseteq w_1$ and $w_0 \not\sqsupseteq w_1$), (ii) u contains v ($w_0 \sqsubseteq w_1$), (iii) v properly contains u ($w_0 \sqsupseteq w_1 00$ or $w_0 \sqsupseteq w_1 1$), (iv) u and v overlap ($w_0 = w_1 0$). In the first three cases, permutation is straightforward and does not change the length of a reduction sequence. In the case (iv), we apply the *permutation rule* in Figure 6(a), which says that the reduction sequence in solid line may be replaced with the other one in broken line. ■

Lemma 10. *Let t_0 be an extended term. 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. As before, one can distinguish four cases, and the critical case is (iv): two reductions overlap. In this case, use the permutation rules in Figure 6 (b) and (c). ■

Proof of Theorem 4. We argue step by step as follows:

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

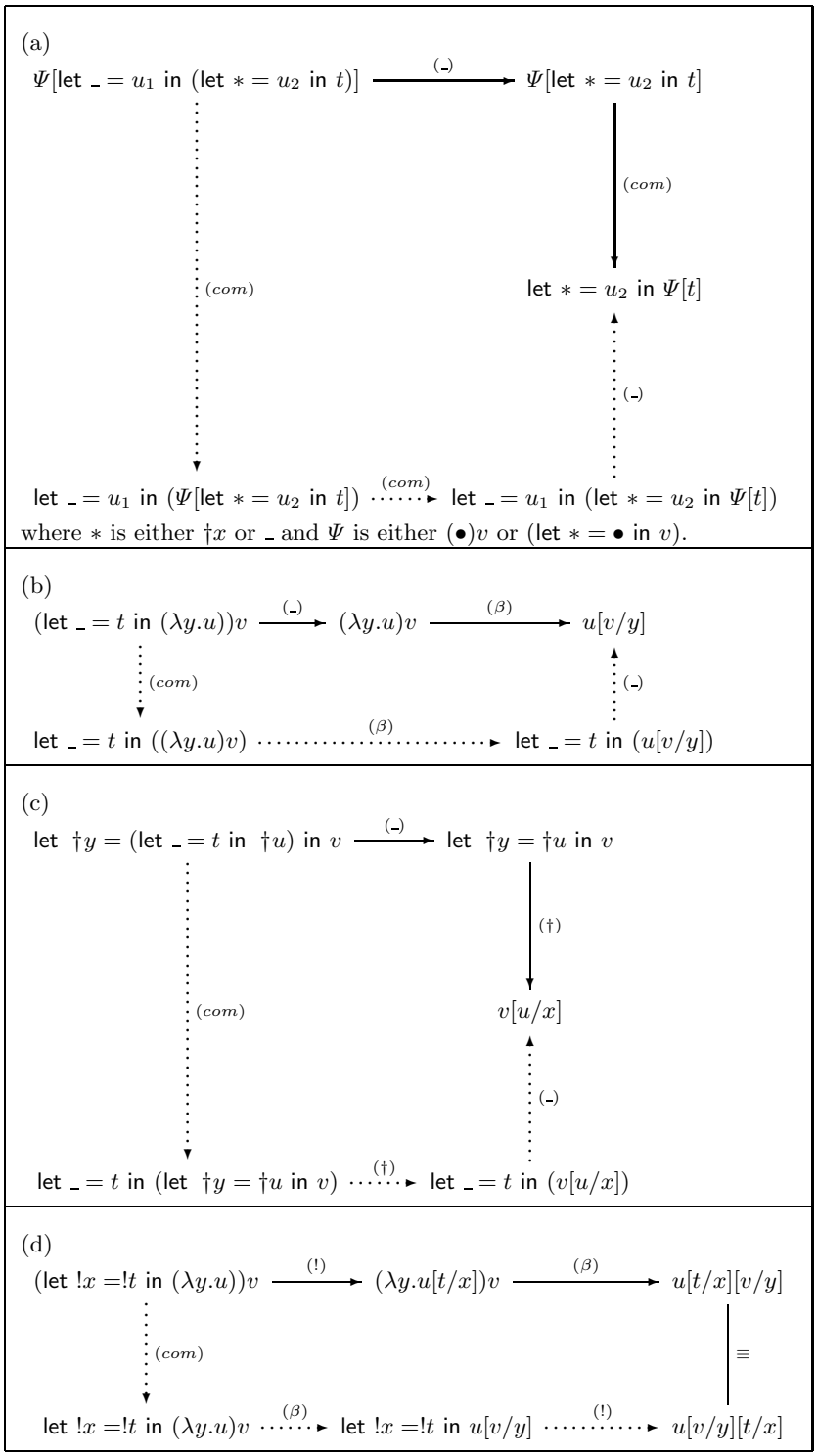


Fig. 6. Permutation Rules

where $|\nu| \leq |\nu'|$. This is proved by induction on $|\nu|$, using Lemma 9.

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

where $|\tau| = |\tau'|$ and $|\nu| \leq |\nu'|$. This is proved by induction on $|\tau|$, using (1).

(3) If $t_0 \xrightarrow{\tau,(\_)*} t_1 \xrightarrow{(r)} t_2$ with (r) proper, then $t_0 \xrightarrow{\nu'*} t_1' \xrightarrow{(\_)*} t_2$,

where ν' is proper and $|\nu'| \geq 1$. This is proved by induction on $|\tau|$, using Lemma 10 and (2).

(4) If $t_0 \xrightarrow{(\_)*} t_1 \xrightarrow{\nu*} t_2$ with ν proper, then $t_0 \xrightarrow{\nu'*} t_1' \xrightarrow{(\_)*} t_2$,

where ν' is proper and $|\nu| \leq |\nu'|$. This is proved by induction on $|\nu|$, using (3).

Now given a term t_0 and a reduction sequence $t_0 \xrightarrow{\sigma}{}^* t_1$, we apply Lemma 8 to obtain a suitable extended term t_0' such that $t_0' \xrightarrow{(\_)*} t_0$. Then the theorem immediately follows from (4). ■

4.2. Standardization Theorem

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

Theorem 5 (Standardization). *Let t_0 be an extended term and $t_0 \xrightarrow{\sigma*} t_1$ be a proper reduction sequence. Then there is a standard proper reduction sequence*

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

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

The proof is again based on permutation of reduction sequences. Each step of permutation is supported by the following two lemmas.

Lemma 11. *Let t_0 be an extended term and $t_0 \xrightarrow{(r_0)} t_1 \xrightarrow{(r_1)} t_2$ be a proper reduction sequence. Suppose that the first reduction is at depth d_0, the second reduction at depth d_1 and $d_0 > d_1$. Then there is a reduction sequence*

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

such that the first reduction is at depth d_1 and all reductions in σ are at depth d_0. Moreover, $|\sigma| \geq 1$.

Proof. As in the proof of Lemma 9, we can distinguish four cases, and the critical case is (iii): the redex of (r_1) properly contains the contractum of (r_0) (note that (ii) and (iv) are impossible). For example, let us consider the following reduction sequence, where the second reduction is (!):

$$\mathsf{let}\ !x = !\Phi[u]\ \mathsf{in}\ v \xrightarrow{(r_0)} \mathsf{let}\ !x = !\Phi[u']\ \mathsf{in}\ v \xrightarrow{(!)} v[\Phi[u']/x].$$

It can be transformed into

$$\mathsf{let}\ !x = !\Phi[u]\ \mathsf{in}\ v \xrightarrow{(!)} v[\Phi[u]/x] \xrightarrow{\sigma,(r_0)}{}^{*}\ v[\Phi[u']/x].$$

Note that v contains at least one x by definition of the extended terms, therefore we always have $|\sigma| \geq 1$. Other cases are straightforward. ■

Lemma 12. *Let t_0 be an extended term and let $t_0 \xrightarrow{(!)} t_1 \xrightarrow{(r)} t_2$ be a proper reduction sequence, where (r) is not (!) and two reductions are at the same depth d. Then there is a reduction sequence*

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

such that every reduction in it is at depth d.

Proof. The most critical is the case where (r) is (β) and two reductions overlap. In this case, use the permutation rule in Figure 6(d). The equivalence between $u[t/x][v/y]$ and $u[v/y][t/x]$ is easily checked (recall that we have adopted the variable convention, so that x does not occur in v and y does not occur in t). ■

Proof of Theorem 5.
(1) Every proper reduction sequence $t_0 \longrightarrow^{*} u$ can be transformed into the following one 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 4, using Lemma 11.
(2) For every $i \leq n-1$, the reduction sequence $t_i \xrightarrow{\tau_i}{}^{*} t_{i+1}$ can be transformed into

$$t_i \xrightarrow{\tau_i'}{}^{*}\ t_i' \xrightarrow{\tau_i''}{}^{*}\ t_{i+1},$$

where τ_i'' consists of (!) reductions and τ_i' consists of other reductions. This is proved by induction on the number of (!) reductions in τ_i, using Lemma 12. ■

4.3. Polystep Weak Normalization Theorem

Here we accommodate the polystep weak normalization theorem of [9,1] to our setting. In our case, the length of a reduction sequence may slightly exceed the size of the final term, because we have the commuting reduction rule (*com*).

Theorem 6 (Polystep Weak Normalization). *Let t be an extended term of depth d and σ be a standard proper reduction sequence $t \xrightarrow{\sigma}{}^{*} u$. Then $|u| \leq |t|^{2^d}$ and $|\sigma| \leq |t|^{2^{d+1}}$.*

Lemma 13. *Let t be an extended term with the size $|t| \geq 2$. Let σ be a proper reduction sequence $t \xrightarrow{\sigma} {}^{*} u$ that consists of (!) reductions at some fixed depth i. Then $|u| \leq |t|(|t|-1)$.*

Proof. We only consider the case $i = 0$, as other cases are similar. We assume that there is no (com) redex of the form

$$(*) \quad \mathsf{let}\ !y = (\mathsf{let}\ !x = !u_1\ \mathsf{in}\ !u_2)\ \mathsf{in}\ u_3$$

at depth 0. It does not cause loss of generality, because we can always find extended terms t' and u' which does not contain a redex of the form (*) and

$$\begin{array}{ccc} t & \xrightarrow{\sigma}{}^{*} & u \\ {\scriptstyle *}\downarrow{\scriptstyle (com)} & & \downarrow{\scriptstyle *}{\scriptstyle (com)} \\ t' & \xrightarrow{\tau}{}^{*} & u'. \end{array}$$

The following notion is useful to estimate the potential size growth caused by (!) reductions. To each extended term t, we associate its *unfolding* $\sharp t \in \mathcal{PT}^w$ as follows:

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

For every extended term v, we claim:

(1) $FO(\sharp v) \leq |v|$.
(2) $|v| \leq |\sharp v| \leq |v|(|v|-1)$.
(3) if $v \xrightarrow{(!)} v'$ at depth 0, then $|\sharp v'| \leq |\sharp v|$.

The lemma follows from (2) and (3):

$$|u| \leq |\sharp u| \leq |\sharp t| \leq |t|(|t|-1).$$

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

$$\begin{aligned} FO(\sharp v) &= FO(!u_1) \cdot FO(x, \sharp u_2) + FO(\sharp u_2) - FO(x, \sharp u_2) \\ &\leq FO(\sharp u_2) \leq |u_2| \leq |v|, \end{aligned}$$

because $FO(!u_1) \leq 1$ 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 $|\sharp v| = |v| \leq |v|(|v|-1)$. If $v \equiv \mathsf{let}\ !x = !u_1\ \mathsf{in}\ u_2$, then

$$\begin{aligned}|\sharp v| &\leq |!u_1| \cdot FO(x, \sharp u_2) + |\sharp u_2| \\ &\leq |!u_1| \cdot |u_2| + |u_2|(|u_2|-1) \\ &\leq |u_2|(|!u_1| + |u_2| - 1) \\ &\leq |v|(|v|-1).\end{aligned}$$

Claim (3) is intuitively clear, as all !-boxes which are to be duplicated are already duplicated by unfolding. It can be formally proved by induction on v. (Note that Claim (3) would not be true if v contained a subterm of the form (*).) ■

Lemma 14. *Let σ be a proper reduction sequence $t \xrightarrow{\sigma} {}^{*} u$ which consists of reductions at some fixed depth i. Then $|\sigma| \leq |t|^2$.*

Proof. Given an extended term v, denote by $|v|_i$ the size of v at depth i. Given v and its subterm u of the form $u \equiv (\mathsf{let} * = u_1\ \mathsf{in}\ u_2)$ at depth i, where $*$ is either $\_$ or $\dagger x$, we define

$$com_i(u, v) := |v|_i - |u_2|_i.$$

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

(1) $|v|_i + com_i(v) \leq |v|^2$.

(2) If $v \xrightarrow{(r)} v'$ by a reduction at depth i, then $|v'|_i + com_i(v') < |v|_i + com_i(v)$.

The lemma follows from these two.

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

$$|v|_i + com_i(v) \leq |v|_i + (|v| - 1) \cdot |v| \leq |v|^2.$$

Claim (2) can be established with the following observations. In case that (r) is (β), (§) or (!), $|v|_i$ strictly decreases and $com_i(v)$ never increases (see Remark 3). In case that (r) is (com), $|v|_i$ does not change and $com_i(v)$ strictly decreases. ■

Proof of Theorem 6. Suppose that σ is partitioned into

$$t \equiv t_0 \xrightarrow{\sigma_0 *} \xrightarrow{\sigma_1 *} t_1 \xrightarrow{\sigma_2 *} \xrightarrow{\sigma_3 *} t_2 \cdots t_d \xrightarrow{\sigma_{2d} *} u,$$

where σ_{2i+1} consists of (!)-reductions at depth i and σ_{2i} consists of other reductions at depth i (for $i \leq d$). By applying Lemma 13 repeatedly, we immediately obtain $|t_d| \leq |t|^{2^d}$. (Without loss of generality, we may assume

that $|t_i| \geq 2$ for $0 \leq i \leq d-1$.) Therefore $|u| \leq |t|^{2^d}$. To give a bound on the length of σ, we further show

$$|t_0| + |t_1| + \cdots + |t_d| \leq |t_0|^{2^d}$$

by induction on d. When $d = 0$, it is trivial. When $d > 0$,

$$\begin{aligned} |t_0| + \cdots + |t_{d-1}| + |t_d| &\leq |t_0|^{2^{d-1}} + |t_d| \\ &\leq |t_0|^{2^{d-1}} + |t_{d-1}|(|t_{d-1}| - 1) \\ &\leq |t_0|^{2^{d-1}} + |t_0|^{2^{d-1}}(|t_0|^{2^{d-1}} - 1) = |t_0|^{2^d}, \end{aligned}$$

by using the induction hypothesis twice as well as Lemma 13. From this and Lemma 14,

$$\begin{aligned} |\sigma| &\leq |t_0|^2 + |t_1|^2 + \cdots + |t_d|^2 \\ &\leq (|t_0| + |t_1| + \cdots + |t_d|)^2 \\ &\leq (|t_0|^{2^d})^2 = |t_0|^{2^{d+1}}. \end{aligned}$$

■

5. Main Results

Now we are in a position to state the main results of this paper.

Theorem 7 (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 $O(s^{2^d})$.*

Proof. Let $t_0 \xrightarrow{\sigma} {}^* t_1$ be a reduction sequence in λLA. By Theorem 4, there are extended terms u_0, u_1 and a proper reduction sequence $u_0 \xrightarrow{\tau} {}^* u_1$ such that $|u_0| = O(|t_0|)$, $|t_1| \leq |u_1|$ and $|\sigma| \leq |\tau|$. By Theorem 5, there is a standard reduction sequence $u_0 \xrightarrow{\nu} {}^* u_1$ longer than τ. By Theorem 6,

$$|\sigma| \leq |\tau| \leq |\nu| \leq |u_0|^{2^{d+1}} = O(s^{2^{d+1}}),$$

and

$$|t_1| \leq |u_1| \leq |u_0|^{2^d} = O(s^{2^d}).$$

■

Corollary 2 (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.*

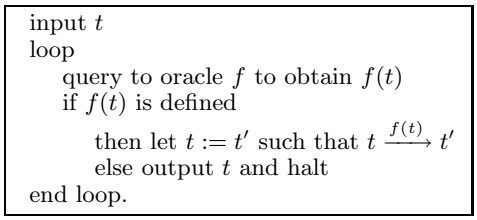


Fig. 7. Algorithm $\mathsf{normalize}_f$

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

To make precise what we mean by *polynomial time* 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 in t whenever t is reducible, and is undefined otherwise. We can think of a Turing machine $\mathsf{normalize}_f$ with function oracle f whose behavior is described in Figure 7.

Corollary 3 (Polynomial time 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. In fact, the worst case, namely the case of (!)-reduction, consists in substituting a subterm of size $\leq |t|$ for at most $|t|$ variable occurrences. As each term in the reduction sequence has size $\leq s^{2^d}$, the total runtime can be estimated by $O(s^{2^d \cdot 2} \cdot s^{2^{d+1}}) = O(s^{2^{d+2}})$. ■

The following is a restatement of Girard-Asperti's result [9,1].

Theorem 8. *Every term t of type* $\mathbf{bint} \multimap \S^d\mathbf{bint}$ *represents a function $g : \{0,1\}^* \longrightarrow \{0,1\}^*$ which is computable in time $O(n^{2^{d+3}})$.*

Proof. Recall that all $\overline{w}$'s are of depth 1, so that the depth of $t\overline{w}$ is constant when w ranges over $\{0,1\}^*$. 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 (4)). By Corollary 3, the normal form of $t\overline{w}$ is computed in time $O(|t\overline{w}|^{2^{d+3}})$, thus in time $O(|w|^{2^{d+3}})$. The normal form should be (a variant of) $\S^d\overline{w'}$ (see Remark 4 (2)), and such w' is uniquely determined by the Church-Rosser property. ■

Corollary 4 (Characterization of the Polynomial Time Functions). *A function $g : \{0,1\}^* \longrightarrow \{0,1\}^*$ is computable in polynomial time if and only if it is represented by a* λLA *term of type* $\mathbf{bint} \multimap \S^d\mathbf{bint}$ *for some d.*

6. Concluding Discussion

We have introduced an untyped term calculus λLA. An advantage of λLA over other existing term calculi for **ILAL** [1,20,19,2] lies in its simplicity; in particular, λLA is free from explicit substitutions, which are essential in [1]. Although explicit substitutions are often useful in giving fine control over the substitution operation, they complicate syntax too much and make the operational intuition unclear. By contrast, our syntax adopts the standard notion of substitution, and henceforth succeeds in reducing 27 rewriting rules of [1] to just 5, all of which have a clear operational meaning. The simplicity of λLA allows us to prove the basic properties such as Church-Rosser and subject reduction in a highly convincing way.

We have reformulated **ILAL** as a type assignment system for λLA, and proved the subject reduction theorem in Section 3. It basically means that proofs of **ILAL** are structurally representable by terms of λLA, and cut-elimination in **ILAL** is in full accordance with normalization in λLA. We have also proved the polynomial time strong normalization theorem for λLA. It says that terms of λLA are normalizable in polynomial time independently of which reduction strategy we take. These two results together imply the polynomial time strong normalization for **ILAL**.

In what follows, we discuss several related issues.

Light Logics and the safe recursion approach. Our main theorem suggests a sharp distinction between λLA and the polytime functional systems based on safe recursion [13,11,7], because normalization in the latter systems is at best *weakly* polytime; although every term of type $\mathit{bint} \to \mathit{bint}$ *denotes* a polynomial time function, it admits of an exponentially long reduction sequence. An interesting consequence is that there cannot be a reduction-preserving embedding of the safe recursion systems into λLA. It should be contrasted with the result of [17], which shows that safe recursion with *non-contractible safe variables* is indeed interepretable in **ILAL**.

Polynomial time strong normalization for **LLL**. It is a delicate question whether **LLL** satisfies the polynomial time strong normalizability, because it depends on what are counted as redices. It seems that **LLL** does satisfy the property with the proviso that only the *ready* cuts (in the sense of [9]) are counted as redices. But it already limits the range of admissible reduction strategies considerably.

In the meantime, there is no doubt that **LLL** without additives is strongly polytime. It is shown in [15] that **LLL** without additives is expressive enough to represent all polynomial time functions.

Lowerbound for normalization. We have obtained an upperbound $O(s^{2^d+2})$ for normalization. On the other hand, Neergaard and Mairson [18] have observed that there is also a considerably tight lowerbound. To see this, observe that the squaring function n^2 can be represented by a term of type

	Type Checking	Typability	Inhabitation
$\mathbf{ILAL}_0$	yes	yes	yes
ILAL	no	?	no

Table 1. Decision Problems for **ILAL**

$\mathbf{int} \multimap \S^2\mathbf{int}$, and thus the function n^{2^d} is represented with depth $2d+1$; whatever algorithm one uses for normalization, it takes at least n^{2^d} steps on Turing machines to write down the output. Therefore, we have a lowerbound $O(s^{2^d})$ for normalization of terms of depth $2d+1$.

It seems possible to sharpen both of these two bounds.

Decision Problems for Type Inference. As for decidability of type inference, one can ask the following questions both for **ILAL** and $\mathbf{ILAL}_0$, where the latter is **ILAL** without second order quantifiers:

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 are summarized in Table 1.

The decidability of type checking and typability for $\mathbf{ILAL}_0$ is shown by Roversi [20]; although his term calculus is different from ours, his result can be accommodated to ours. The undecidability of type checking for **ILAL** follows from the undecidability of the same problem for System F. It is shown in [24] that the semi-unification problem, which is known to be undecidable, is reducible to type checking of assersions of the following form:

$$\vdash \lambda bc.b(\lambda x.cxx) : (\forall\gamma.(\gamma \to \gamma) \to \beta) \to \forall\overrightarrow{\alpha}.A \to \beta,$$

where A is quantifier-free and $\forall\overrightarrow{\alpha}.A$ is its universal closure. We can show that the above assersion holds in System F if and only if the following one holds in **ILAL**:

$$\vdash \lambda bc.b(\lambda^! x.\mathsf{let}\ \S d = c\ \mathsf{in}\ \S dxx) : (\forall\gamma.(!\gamma \multimap \S\gamma) \multimap \beta) \multimap \forall\overrightarrow{\alpha}.\S A' \multimap \beta,$$

where A' is A with $\to$ replaced by $\multimap$. Therefore, type checking for **ILAL** is undecidable.

It is open whether typability for **ILAL** is decidable or not. As for inhabitation problems, see [22].

Acknowledgements. We are indebted to Professor Harry Mairson and Professor Mitsuhiro Okada for helpful suggestions and stimulating discussions. Our thanks are also due to Daniel de Carvalho for his careful reading of the earlier version.

References

1. A. Asperti. Light affine logic. In *Proceedings of LICS'98*, 1998.
2. A. Asperti and L. Roversi. Intuitionistic light affine logic (proof-nets, normalization complexity, expressive power, programming notation). *ACM Transactions on Computational Logic*, 3(1):137 – 175, 2002.
3. P. Baillot. Stratified coherence spaces: a denotational semantics for light linear logic. *Theoretical Computer Science*, 2004. to appear.
4. P. Baillot and K. Terui. Light types for polynomial time computation in lambda-calculus. In *Proceedings of LICS 2004*, to appear.
5. 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.
6. S. Bellantoni and S. Cook. New recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992.
7. S. Bellantoni, K.-H. Niggl, and H. Schwichtenberg. Ramification, modality and linearity in higher type recursion. *Annals for Pure and Applied Logic*, 104:17–30, 2000.
8. V. Danos and J.-B. Joinet. Linear logic & elementary time. *Information and Computation*, 183(1):123–137, 2003.
9. J.-Y. Girard. Light linear logic. *Information and Computation*, 14(3):175–204, 1998.
10. 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.
11. M. Hofmann. Safe recursion with higher types and BCK algebra. *Annals of Pure and Applied Logic*, 104:113–166, 2000.
12. M. Kanovich, M. Okada, and A. Scedrov. Phase semantics for light linear logic. *Theoretical Computer Science*, 294(3):525–549, 2003.
13. D. Leivant. A foundational delineation of poly-time. *Information and Computation*, 110(2):390–420, 1994.
14. D. Leivant and J.-Y. Marion. Ramified recurrence and computational complexity I: Word recurrence and poly-time. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pages 320 – 343. Birkhauser, 1994.
15. H. Mairson and K. Terui. On the computational complexity of cut-elimination in linear logic. In *Proceedings of ICTCS 2003*, pages 23–36. LNCS 2841, 2003.
16. A. S. Murawski and C.-H. L. Ong. Discreet games, light affine logic and ptime computation. In *Proceedings of CSL 2000*, pages 427–441. Springer-Verlag, LNCS 1862, 2000.
17. A. S. Murawski and C.-H. L. Ong. On an interpretation of safe recursion in light affine logic. *Theoretical Computer Science*, 318:197–223, 2004.
18. P. Neergaard and H. Mairson. LAL is square: Representation and expressiveness in light affine logic. Presented at the Fourth International Workshop on Implicit Computational Complexity, 2002.
19. L. Roversi. A P-time completeness proof for light logics. In *Proceedings of CSL'99*, pages 469–483. Springer-Verlag, LNCS 1683, 1999.
20. 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.
21. K. Terui. Light affine lambda calculus and polytime strong normalization. In *Proceedings of LICS2001*, pages 209–220, 2001.
22. K. Terui. *Light Logic and Polynomial Time Computation.* PhD thesis, Keio University, March 2002. Available at http://research.nii.ac.jp/∼terui.
23. K. Terui. Light affine set theory: a naive set theory of polynomial time. *Studia Logica*, 77:9–40, 2004.
24. J. B. Wells. Typability and type checking in system F are equivalent and undecidable. *Annals of Pure and Applied Logic*, 98:111–156, 1999.