Pure lambda calculus as a term syntax for light affine logic

(Joint work with Patrick Baillot)

Kazushige Terui

terui@nii.ac.jp

National Institute of Informatics, Tokyo

Background (1)

- Intuitionistic Light Affine Logic (Girard 98, Asperti 98): a subsystem of Affine Logic (i.e. Linear Logic with unrestricted weakening) which captures Polynomial Time:

  $f : \{0,1\}^* \longrightarrow \{0,1\}$ is representable in LAL $\Longleftrightarrow$ f is polytime.

- Multiplicative Light Linear Logic (without weakening) also captures Polynomial Time (Mairson and Terui 2003).

- Nevertheless, LAL is still significant, as weakening is useful in programming.

Motivation (1)

- What is the best proof syntax for LAL?
  - Asperti's calculus with explicit substitutions: too complicated (27 rewriting rules)
  - Proof nets: simple but space-consuming.
  - Roveri's notation (Roversi 99): simple, compact but not accurate.
- Light Affine Lambda Calculus λLA (Terui 2001): Simple, compact and accurate, satisfies polynomial time strong normalization. However, not as simple as pure λ-calculus...

Motivation (2)

- It is not realistic to think of λLA (or other proof syntax) as a programming language; no one wants to use such a complicated one! (in spite of (Roversi 99))
- In reality, LAL should be used just as a sophisticated type system for λ-calculus:

1. Programmar writes a program in λ-calculus.
2. Its typability in LAL is then checked automatically ([Baillot 2002] shows typability in propositional LAL is decidable)
3. If it is typable, then polynomial time computability is guaranteed.

Motivation (3)

- But in what sense? Is it necessary to translate a program into a λ_{LA} term (or whatever) and apply LAL normalization procedure?
  - Not very economic; one would have to newly implement a λ_{LA} evaluator.
- I want to consider λ-calculus itself as a “real” term calculus for LAL with subject reduction and polytime normalization.
- Then, you can utilize some existing evaluator for λ-calculus.

λ-calculus is bad, in general

- Unfortunately, it fails in general: Even if a λ-term is typable in LAL, it may take exponential time to normalize via β-reduction:

$$y_i :!A \multimap !A \multimap !A \vdash \lambda x.y_i x x :!A \multimap !A$$

$$z :!A \vdash z :!A$$

$$y_1 :!A \multimap !A \multimap !A, \ldots, y_n :!A \multimap !A \multimap !A, z :!A \vdash (\lambda x.y_1 x x)(\cdots(\lambda x.y_n x x)z \cdots) :!A$$

$$y :!(!A \multimap !A \multimap !A), z :!!A \vdash (\lambda x.y x x)^n z : \S !A$$

Our Goal

- To define a subsystem of LAL (Dual Light Affine Logic) for which λ-calculus itself can be considered as a term syntax:
  - β-reduction satisfies subject reduction w.r.t. DLAL types.
  - β-reduction satisfies polytime strong normalization.
  - DLAL is expressive enough to represent all polytime functions.

Syntax of λLA

- Types:

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

- !-discharged types: $[A]_!$.
- $\S$-discharged type: $[A]_\S$.
- Pseudo-terms:

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

- Notation: $\dagger$ stands for either ! or $\S$.

Type Assignment Rules of λLA

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

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

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

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

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

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

Reduction Rules of λLA

$$
\begin{array}{llcl}
(\beta) & (\lambda x.t)u & \longrightarrow & t[u/x] \\
(\S) & \text{let } \S u \text{ be } \S x \text{ in } t & \longrightarrow & t[u/x] \\
(!) & \text{let } !u \text{ be } !x \text{ in } t & \longrightarrow & t[u/x] \\
(com1) & (\text{let } u \text{ be } \dagger x \text{ in } t)v & \longrightarrow & \text{let } u \text{ be } \dagger x \text{ in } (tv) \\
(com2) & \text{let } (\text{let } u \text{ be } \dagger x \text{ in } t) \text{ be } \dagger y \text{ in } v & \longrightarrow & \text{let } u \text{ be } \dagger x \text{ in } (\text{let } t \text{ be } \dagger y \text{ in } v)
\end{array}
$$

Main Properties of λLA

- Subject Reduction Theorem:

$$\Gamma \vdash t : A, \;\; t \longrightarrow u \Longrightarrow \Gamma \vdash u : A.$$

- Church-Rosser Theorem:

$$t_1 \longleftarrow^* t_0 \longrightarrow^* t_2 \quad \Longrightarrow \quad t_1 \longrightarrow^* t_3 \longleftarrow^* t_2 \text{ for some term } t_3.$$

- Polynomial Time Strong Normalization Theorem:
  Let t be of depth d.

$$t \overset{n}{\longrightarrow^*} u \quad \Longrightarrow \quad n \leq O(|t|^{2^{d+1}}).$$

(in contrast to Safe Recursion-based Systems which are only weakly (CBV) polytime.)

From λLA to λ-calculus

- In what follows, we assume that $(\S), (!), (com1), (com2)$ are automatically applied, All terms are normal w.r.t. $(\S), (!), (com1), (com2)$. There is only one reduction rule (β^*) which consists in (β) followed by $(\S), (!), (com1), (com2)$.
- Erasure $(\ )^-$: $\Lambda_{LA} \longrightarrow \Lambda$

$$
\begin{array}{rcl}
x^- & \equiv & x \\
(\lambda x.t)^- & \equiv & \lambda x.(t^-) \\
(tu)^- & \equiv & t^- u^- \\
(\dagger t)^- & \equiv & t^- \\
(\text{let } u \text{ be } \dagger x \text{ in } t)^- & \equiv & t^-[u^-/x]
\end{array}
$$

- Why we need ! and § in term syntax? What's wrong with erasures?

$\S$ is Redundant in Typed Setting

- $\S$ and ! block illegal (non-stratified) reductions:

$$
\begin{array}{rcl}
(\S\lambda x.t)v & \not\longrightarrow & \S t[v/x], \text{ whereas} \\
(\lambda x.t^-)v^- & \longrightarrow & t^-[v^-/x]. \\
\text{let } (\lambda x.t) \text{ be } \S y \text{ in } \S yv & \not\longrightarrow & \S(\lambda x.t)v \longrightarrow \S t[u/x], \text{ whereas} \\
(\lambda x.t^-)v^- & \longrightarrow & t^-[v^-/x]
\end{array}
$$

- In typed setting, however, we have
  - Lemma: uv is typable and $(\S, !, com)$-normal $\Longrightarrow u \equiv x$ or u_1u_2 or $\lambda x.u_1$.
  - Lemma: (let u be $\S y$ in v) is typable and $(\S, !, com)$-normal $\Longrightarrow u \equiv x$ or u_1u_2.
- $(\S\lambda x.t)v$ and let $(\lambda x.t)$ be $\S y$ in $\S yv$ are not typable.

! Seems Necessary...

- Assume $u :!A \multimap !A \multimap !A$ and $x_i : A$, and let

$$
\begin{aligned}
t_0 &\equiv u!x_0!x_0 \\
t_1 &\equiv \text{let } (u!x_1!x_1) \text{ be } !x_0 \text{ in } (u!x_0!x_0) \\
t_2 &\equiv \text{let } (u!x_2!x_2) \text{ be } !x_1 \text{ in } (\text{let } (u!x_1!x_1) \text{ be } !x_0 \text{ in } (u!x_0!x_0)) \\
&\ldots
\end{aligned}
$$

- $|t_i|$ is linear in i. However,

$$
\begin{aligned}
t_0^- &\equiv ux_0x_0 \\
t_1^- &\equiv u(ux_1x_1)(ux_1x_1) \\
t_2^- &\equiv u(u(ux_2x_2)(ux_2x_2))(u(ux_2x_2)(ux_2x_2))
\end{aligned}
$$

- $|t_i^-|$ is exponential in i.

! Seems Necessary... (2)

- “While every datum of type !A is eventually sharable, not all of them are actually duplicable.” (Asperti 98)
- Erasing operation wrongly duplicates all data of type !A.

Key Idea: Small-Headedness

- If the head of $\mathsf{let}_!$ is always a variable, i.e.,

$$\text{let } y \text{ be } !x \text{ in } u$$

  then for every typable t, $|t^-| \leq |t|$.

- Call such terms small headed.

- Our plan: Constrain types so that
  - All typable terms are small headed, and
  - Typable terms are closed under reduction.

Preservation of Small-Headedness (1)

- Good case:

$$(\lambda x.\text{let } x \text{ be } !y \text{ in } t)!u \longrightarrow \text{let } !u \text{ be } !y \text{ in } t \longrightarrow t[u/y].$$

- Bad case:

$$(\lambda x.\text{let } x \text{ be } !y \text{ in } t)(u_1 u_2) \text{ (SH)} \longrightarrow \text{let } (u_1 u_2) \text{ be } !y \text{ in } t \text{ (non-SH)}.$$

- Observe

$$(\lambda x^{!A}.\text{let } x^{!A} \text{ be } !y \text{ in } t)(u_1^{B\multimap !A} u_2^{B})^{!A}$$

- Types of the form $B\multimap !A$ should be excluded.

Preservation of Small-Headedness (2)

- Another bad case:

  let $(\S u)$ be $\S x$ in $\S($let x be $!y$ in $t)$ (SH) $\longrightarrow$ $\S($let u be $!y$ in $t)$ (non-SH).

- Observe

$$\text{let } (\S u)^{\S !A} \text{ be } \S x \text{ in } \S(\text{let } x^{!A} \text{ be } !y \text{ in } t)$$

- Types of the form $\S !A$ or $!!A$ should be excluded.
- We exclude $\forall\alpha.!A$, too.
- The only possible use of !: $!A \multimap B \equiv A \Rightarrow B$
- Consider the subsystem of Light Affine Logic where the use of ! is restricted to $!A \multimap B$.

Syntax of Dual Light Affine Logic

- Somewhat similar to Dual Intuitionistic Linear Logic of (Barber and Plotkin 1997).
- Types:

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

- Judgement: $\Gamma; \Delta \vdash t : C$
  (Γ "!-discharged" or "intuitionistic," while Δ "non-discharged" or "linear.")
- Pseudo-terms: Pure λ-terms

Type Assignment Rules of DLAL

$$\frac{}{;x:A \vdash x:A}\ (Id)$$

$$\frac{\Gamma_1;\Delta_1 \vdash u:A \quad \Gamma_2;x:A,\Delta_2 \vdash t:C}{\Gamma_1,\Gamma_2;\Delta_1,\Delta_2 \vdash t[u/x]:C}\ (Cut1) \qquad \frac{;y:B \vdash u:A \quad x:A,\Gamma;\Delta \vdash t:C}{y:B,\Gamma;\Delta \vdash t[u/x]:C}\ (Cut2)$$

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

$$\frac{\Gamma_1;\Delta_1 \vdash u:A \quad \Gamma_2;x:B,\Delta_2 \vdash t:C}{\Gamma_1,\Gamma_2;y:A \multimap B,\Delta_1,\Delta_2 \vdash t[yu/x]:C}\ (\multimap l) \qquad \frac{\Gamma;x:A,\Delta \vdash t:B}{\Gamma;\Delta \vdash \lambda x.t:A \multimap B}\ (\multimap r)$$

$$\frac{;z:D \vdash u:A \quad \Gamma;x:B,\Delta \vdash t:C}{z:D,\Gamma;y:A \Rightarrow B,\Delta \vdash t[yu/x]:C}\ (\Rightarrow l) \qquad \frac{x:A,\Gamma;\Delta \vdash t:B}{\Gamma;\Delta \vdash \lambda x.t:A \Rightarrow B}\ (\Rightarrow r)$$

$$\frac{;\Gamma,x_1:B_1,\ldots,x_n:B_n \vdash t:A}{\Gamma;x_1:\S B_1,\ldots,x_n:\S B_n \vdash t:\S A}\ (\S)$$

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

Comments on DLAL

- Rules for Intuitionistic implication are abbreviations of

$$\dfrac{\dfrac{[A]_!, \Gamma \vdash B}{!A, \Gamma \vdash B}}{\Gamma \vdash !A \multimap B} \qquad \dfrac{\dfrac{D \vdash A}{[D]_! \vdash !A} \quad B, \Gamma \vdash C}{[D]_!, !A \multimap B, \Gamma \vdash C}$$

- Intuitionistic Modus Ponens has limitation on contexts:

$$\dfrac{; D \vdash A \quad \Gamma; \Delta \vdash A \Rightarrow B}{D, \Gamma; \Delta \vdash B}$$

- Dereliction (from linear to intuitionistic context) is synchronous and accompanied by $\S$-promotion:

$$\dfrac{; \Gamma, \Delta \vdash A}{\Gamma; \S\Delta \vdash \S A}$$

DLAL is Expressive Enough

- Data types are all representable in DLAL:

$$\begin{aligned} N &\equiv \forall\alpha.!(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha) \\ &\equiv \forall\alpha.(\alpha \multimap \alpha) \Rightarrow \S(\alpha \multimap \alpha). \end{aligned}$$

- Theorem: All polynomial time functions are representable with type $W \multimap \S^d W$ in DLAL (where W is a type for binary words).
- Proof: Read (Asperti and Roversi 2001) and observe that the only place where non-intuitionistic ! appears is in Coercion:

$$coerc : N \multimap \S!N.$$

Modified Coercion

- Lemma: There is a term $coerc' : (N \Rightarrow A) \Rightarrow (N \multimap \S A)$ such that t and $coerc'(t)$ are extentionally equivalent for every $t : N \Rightarrow A$.
- Ex) In DLAL, we have

$$
\begin{array}{rcl}
mult & : & N \Rightarrow (N \multimap \S N) \\
coerc'(mult) & : & N \multimap \S(N \multimap \S N) \\
mult' & : & N \multimap N \multimap \S\S N \ \text{(by coercion for §).}
\end{array}
$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\ Suc \\ ;N \vdash N\end{array} \qquad \dfrac{}{;A \vdash A}}{N; N \Rightarrow A \vdash A}}{;N \Rightarrow A \vdash N \Rightarrow A}}{;\vdash (N \Rightarrow A) \multimap (N \Rightarrow A)} \qquad \dfrac{\dfrac{\dfrac{}{;N \Rightarrow A \vdash N \Rightarrow A} \qquad \dfrac{\begin{array}{c}\vdots\ Zero \\ ;\vdash N\end{array} \qquad \dfrac{}{;A \vdash A}}{;N \Rightarrow A \vdash A}}{;N \Rightarrow A, (N \Rightarrow A) \multimap (N \Rightarrow A) \vdash A}}{N \Rightarrow A; \S((N \Rightarrow A) \multimap (N \Rightarrow A)) \vdash \S A}}{N \Rightarrow A; ((N \Rightarrow A) \multimap (N \Rightarrow A)) \Rightarrow \S((N \Rightarrow A) \multimap (N \Rightarrow A)) \vdash \S A}}{N \Rightarrow A; N \vdash \S A}
$$

Proving Main Properties of DLAL (1)

- DLAL may be considered as type assignment for λLA, too.
- Lemma (Subject Reduction for λLA w.r.t. DLAL): Let $t : \lambda$LA. $\Gamma; \Delta \vdash t : A$ in DLAL, $t \longrightarrow u \Longrightarrow \Gamma; \Delta \vdash u : A$ in DLAL.
- Lemma (Small-Headedness): $($let u be $!x$ in $v)$ is typable in DLAL and $(\S, !, com)$-normal $\Longrightarrow u \equiv x$.
- Lemma (Erasing is linear and preserves Types):
  1. Let $t : \lambda$LA. $\Gamma; \Delta \vdash t : A$ in DLAL $\Longrightarrow \Gamma; \Delta \vdash t^- : A$ and $|t^-| \leq |t|$.
  2. Let $t : \lambda$-term, $\Gamma; \Delta \vdash t : A$ in DLAL $\Longrightarrow \Gamma; \Delta \vdash \tilde{t} : A$ for some $\tilde{t} : \lambda$LA such that $(\tilde{t})^- \equiv t$ and $|t| \leq |\tilde{t}|$.

Proving Main Properties of DLAL (2)

- Lemma (λLA simulates λ): Let t be a $(\S, !, com)$-normal λLA-term.

$$
\begin{tikzcd}
t_0 \arrow[r, "(\beta)"] & u_0 \\
t \arrow[u, "-"'] \arrow[r, dotted, "(\beta^*)"] & u \arrow[u, dotted, "-"']
\end{tikzcd}
$$

- Proof: By induction on t.

Main Properties of DLAL (1)

- Subject Reduction Theorem: Let t : λ-term.

$$\Gamma; \Delta \vdash t : A, \;\; t \longrightarrow u \Longrightarrow \Gamma; \Delta \vdash u : A.$$

- Proof: There is a λLA-term $\tilde{t}$ s.t. $\Gamma; \Delta \vdash \tilde{t} : A$ and

$$\begin{tikzcd} t \arrow[r, "(\beta)"] & u \\ \tilde{t} \arrow[u, "-"'] \arrow[r, dotted, "(\beta^*)"] & \tilde{u} \arrow[u, dotted, "-"'] \end{tikzcd}$$

By Subject reduction for λLA, $\Gamma; \Delta \vdash \tilde{u} : A$. So $\Gamma; \Delta \vdash u : A$.

Main Properties of DLAL (2)

- Polynomial Time Strong Normalization Theorem:
  Let $\vdash t : A$ and A be Π_1 and of depth d.

$$t \xrightarrow{n}{}^{*} u \Longrightarrow n \leq O(|t|^{2^{d+1}}).$$

- Proof:

$$\begin{tikzcd}
t \arrow[r, "(\beta)"] & \cdot \arrow[r, dotted] & \cdot \arrow[r, "(\beta)"] & u \\
\tilde{t} \arrow[u, "-"'] \arrow[r, dotted, "(\beta^*)"] & \cdot \arrow[r, dotted] & \cdot \arrow[r, dotted, "(\beta^*)"] & \tilde{u} \arrow[u, dotted, "-"']
\end{tikzcd}$$