

An Elementary Affine λ -calculus with Multithreading and Side Effects

Antoine Madet Roberto Amadio

Laboratoire PPS
Université Paris Diderot

CHOCO party
06/04/2011, ENS Lyon

Motivation

- ▶ Control the time complexity of a higher-order functional language featuring **multithreading** and **side effects** (references, channels)
- ▶ Focus is on **elementary time** (tower of exponentials)

Iteration with side effects

list_it : $\forall u. \forall t. (u \rightarrow t \rightarrow t) \rightarrow \text{List } u \rightarrow t \rightarrow t$

mult2 : $\text{Ref } \mathbf{N} \rightarrow \mathbf{1} \rightarrow \mathbf{1}$

run : $\mathbf{1}$

run = list_it mult2 [x, y, z] *

	run		$(x \leftarrow \overline{m})$		$(y \leftarrow \overline{n})$		$(z \leftarrow \overline{p})$
$\xrightarrow{*}$	*		$(x \leftarrow \overline{2m})$		$(y \leftarrow \overline{2n})$		$(z \leftarrow \overline{2p})$

Iteration with side effects

list_it : $\forall u. \forall t. (u \rightarrow t \rightarrow t) \rightarrow \text{List } u \rightarrow t \rightarrow t$

mult2 : $\text{Ref } \mathbf{N} \rightarrow \mathbf{1} \rightarrow \mathbf{1}$

run : $\mathbf{1}$

run = list_it mult2 [x, y, z] *

	run		run		run		$(x \leftarrow \overline{m})$		$(y \leftarrow \overline{n})$		$(z \leftarrow \overline{p})$
$\xrightarrow{*}$	*		*		*		$(x \leftarrow \overline{8m})$		$(y \leftarrow \overline{8n})$		$(z \leftarrow \overline{8p})$

Context

Computational complexity and Linear Logic...

- ▶ Well-known **light logics** (ELL, SLL, LLL,...) have been used to characterize various complexity classes
- ▶ Complexity is tamed through a fine control of the **depth** (!-box nesting) of formulae during cut-elimination
- ▶ Several **type systems** and **formation rules** for λ -calculi that guarantee complexity properties of programs

Some recent works

- ▶ A polynomial time functional language with recursive definitions and pattern-matching
[Baillot, Gaboardi, Mogbil ESOP'10]
- ▶ A polynomial time higher-order π -calculus
[Dal Lago, Martini, Sangiorgi EXPRESS'10]

Context

Stratified region

- ▶ The notion of **stratified region** has been used to prove the termination of higher-order concurrent imperative programs [*Boudol CONCUR'07, Amadio APLAS'09*]
- ▶ Intuitively: avoid the definition of **circular region**

A diverging computation

	$\text{get}(r)^*$		$(r \leftarrow \lambda x.\text{get}(r)x)$
$\rightarrow$	$(\lambda x.\text{get}(r)x)^*$		$(r \leftarrow \lambda x.\text{get}(r)x)$
$\rightarrow$	$\text{get}(r)^*$		$(r \leftarrow \lambda x.\text{get}(r)x)$
$\rightarrow$	$\dots$		

Contribution

From light logics to side effects and concurrency

1. Revising the functional case

2. Towards side effects and concurrency

- ▶ The language is extended with **parallel composition** and **read/write** operations on regions
- ▶ A **revised notion of depth** leads to termination in elementary time under a call-by-value evaluation strategy (**no stratification** anymore !)

3. An elementary type system

- ▶ A **progress** property: programs cannot deadlock
- ▶ All **elementary functions** are typable [*Danos, Joinet 2001*]

1 - Elementary time in the functional case

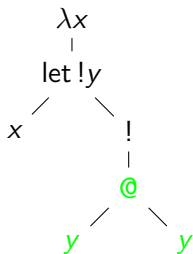
A modal λ -calculus

Syntax

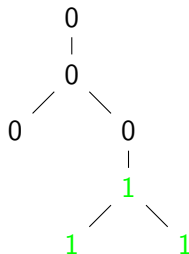
$M, N ::= x, y, z \dots \mid \lambda x.M \mid MN \mid !M \mid \text{let } !x = N \text{ in } M$

Depth

Take $M = \lambda x.\text{let } !y = x \text{ in } !(yy)$



syntax tree



depth of occurrences

$d(M) = 1$, i.e. maximum depth of its occurrences

A modal λ -calculus

Semantics

For any evaluation context E :

$$\begin{aligned} E[(\lambda x.M)N] &\rightarrow E[M[N/x]] \\ E[\text{let } !x = !N \text{ in } M] &\rightarrow E[M[N/x]] \end{aligned}$$

Learning from Elementary Linear Logic

Some observations

If $M \rightarrow M'$ by a redex at depth i , the following should hold:

- (1) the depth of the term does not increase
- (2) the duplication of occurrences should only happen at depth $> i$

Two constraints on terms

- ▶ λ -abstraction should be **affine** and bind at current depth

$$\lambda x.(\lambda y.xy) \quad \lambda x.(!x)(\lambda y.y) \quad \lambda x.(\lambda y.(xy)x)$$

- ▶ `let !` should bind at current depth $+1$

$$\text{let } !x = M \text{ in } !(xx) \quad \text{let } !x = M \text{ in } (!x)(yx)$$

Depth system

A depth judgement

$$\left. \begin{array}{l} \Gamma = x_1 : \delta_1, \dots, x_n : \delta_n \\ \Gamma \vdash^\delta M \end{array} \right\} \iff x_i \text{ occurs at depth } \delta_i \text{ in } !^\delta M$$

$$\frac{}{\Gamma, x : \delta \vdash^\delta x}$$

$$\frac{\Gamma, x : \delta \vdash^\delta M \quad \text{FO}(x, M) \leq 1}{\Gamma \vdash^\delta \lambda x. M}$$

$$\frac{\Gamma \vdash^\delta M \quad \Gamma \vdash^\delta N}{\Gamma \vdash^\delta MN}$$

$$\frac{\Gamma \vdash^\delta N \quad \Gamma, x : (\delta + 1) \vdash^\delta M}{\Gamma \vdash^\delta \text{let } !x = N \text{ in } M}$$

$$\frac{\Gamma \vdash^{\delta+1} M}{\Gamma \vdash^\delta !M}$$

Some examples

A well-formed term

$$\frac{\frac{x : \delta \vdash^{\delta} x}{x : \delta \vdash^{\delta} \text{let } !y = x \text{ in } !(yy)} \quad \frac{\frac{\frac{x : \delta, y : \delta + 1 \vdash^{\delta+1} y \quad x : \delta, y : \delta + 1 \vdash^{\delta+1} y}{x : \delta, y : \delta + 1 \vdash^{\delta+1} yy}}{x : \delta, y : \delta + 1 \vdash^{\delta} !(yy)}}{x : \delta \vdash^{\delta} \text{let } !y = x \text{ in } !(yy)} \quad \frac{}{\vdash^{\delta} \lambda x. \text{let } !y = x \text{ in } !(yy)}$$

Some examples

A non well-formed term

$$\frac{\frac{\frac{x : \delta \vdash^{\delta} x}{x : \delta, y : \delta + 1 \vdash^{\delta+1} y} \quad \frac{x : \delta, y : \delta + 1 \vdash^{\delta+2} y}{x : \delta, y : \delta + 1 \vdash^{\delta+1} !y}}{x : \delta, y : \delta + 1 \vdash^{\delta+1} y!y} \quad \frac{x : \delta, y : \delta + 1 \vdash^{\delta+1} y!y}{x : \delta, y : \delta + 1 \vdash^{\delta} !(y!y)}}{x : \delta \vdash^{\delta} \text{let } !y = x \text{ in } !(y!y)} \quad \vdash^{\delta} \lambda x. \text{let } !y = x \text{ in } !(y!y)$$

Subject reduction

If $\Gamma \vdash^\delta M$ and $M \rightarrow M'$ then $\Gamma \vdash^\delta M'$

Termination by lexicographic ordering

Measure

Let M be a term with $\alpha = d(M)$ and $\omega_i(M)$ be the number of occurrences at depth i of M .

$$\mu_\alpha(M) = (\omega_\alpha(M), \omega_{\alpha-1}(M), \dots, \omega_0(M))$$

Take the (well-founded) lexicographic order $>$ from right to left:
deeper occurrences have less weight than shallow ones.

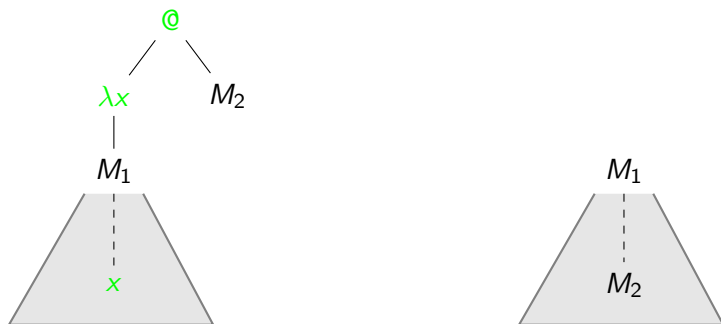
Termination

Let M be a well-formed term with $\alpha = d(M)$.

If $M \rightarrow M'$ then $\mu_\alpha(M) > \mu_\alpha(M')$.

Termination by lexicographic ordering

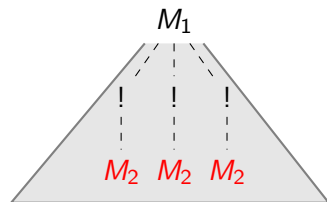
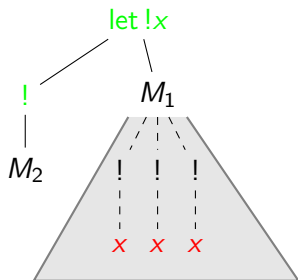
$$M = E[(\lambda x.M_1)M_2] \quad \rightarrow \quad M' = E[M_1[M_2/x]]$$



$\Rightarrow \mu_\alpha(M) > \mu_\alpha(M')$ since 3 occurrences disappear at depth i

Termination by lexicographic ordering

$$M = E[\text{let } !x = !M_2 \text{ in } M_1] \quad \rightarrow \quad M' = E[M_1[M_2/x]]$$



- ▶ 2 **occurrences** disappear at depth i
- ▶ the number of x in M_1 is bounded by $k = \omega_{i+1}(M)$
- ▶ M_2 contains occurrences at depth $\geq i + 1$

$$\begin{aligned} \mu_\alpha(M') &\leq (k \cdot \omega_\alpha(M), \dots, k \cdot \omega_{i+1}(M), \omega_i(M) - 2, \omega_{i-1}(M) \dots, \omega_0(M)) \\ &< \mu_\alpha(M) \end{aligned}$$

A combinatorial proof of elementary time

Tower functions

We define a family of tower functions by induction on n :

$$\begin{aligned} t_\alpha() &= 0 \\ t_\alpha(x_1, x_2, \dots, x_n) &= (\alpha \cdot x_1)^{2^{t_\alpha(x_2, \dots, x_n)}} \quad n \geq 1 \end{aligned}$$

Elementary bound

Let M be a well-formed term with $\alpha = d(M)$.

If $M \rightarrow M'$ then $t_\alpha(\mu_\alpha(M)) > t_\alpha(\mu_\alpha(M'))$.

Elementary time normalisation

The normalisation of well-formed terms can be performed in time elementary in the size of the terms.

2 - Towards multithreading and side effects

A modal λ -calculus with multithreading and regions

An abstract language

- ▶ **Regions** are sets of dynamically generated values such as references or channels
- ▶ The language is highly **non-deterministic** and entails complexity bounds for more concrete ones

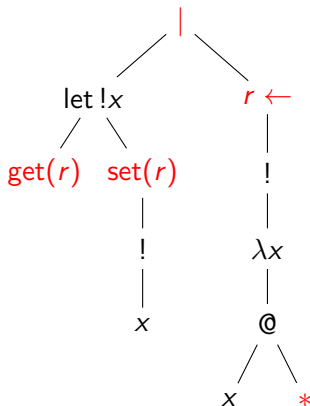
Syntax

$$\begin{aligned} V &::= * \mid r, r', r'' \dots \mid x, y, z, \dots \mid \lambda x. M \mid !V \\ M &::= V \mid MM \mid !M \mid \text{let } !x = M \text{ in } M \\ &\quad \text{set}(r, V) \mid \text{get}(r) \mid (M \mid M) \end{aligned}$$

- ▶ **stores** are multisets of affectations ($r \leftarrow V$)
- ▶ **programs** are multisets of terms and stores

Syntax tree

Let $P = \text{let } !x = \text{get}(r) \text{ in } \text{set}(r, !x) \mid (r \leftarrow !(\lambda x. x*))$



We have $d(P) = 1$

Operational semantics

- ▶ E are **call-by-value** evaluation contexts
- ▶ Writing amounts to **add** rather than overwrite:

$$E[\text{set}(r, V)] \rightarrow E[*] \mid (r \leftarrow V)$$

- ▶ One can **consume** from the store:

$$E[\text{get}(r)] \mid (r \leftarrow V) \rightarrow E[V]$$

- ▶ Or one can **copy** from the store:

$$E[\text{let } !x = \text{get}(r) \text{ in } M] \mid (r \leftarrow !V) \rightarrow E[M[V/x]] \mid (r \leftarrow !V)$$

- ▶ Non-deterministic **thread scheduling**

Non-deterministic read

	let !x = get(r) in set(r, !x)		(r ← !(λx.x*))
→	set(r, !(λx.x*))		(r ← !(λx.x*))
→	*		(r ← !(λx.x*)) (r ← !(λx.x*))

or

	let !x = get(r) in set(r, !x)		(r ← !(λx.x*))
→	let !x = !(λx.x*) in set(r, !x)		
→	set(r, !(λx.x*))		
→	*		(r ← !(λx.x*))

The impact of side effects on depth

Recall what ELL suggests

If $P \rightarrow P'$ by a redex at depth i , the following should hold:

- ▶ the depth of the program does not increase
- ▶ the duplication of occurrences should only happen at depth $> i$

The impact of side effects on depth

$$E[\text{set}(r, !V)] \rightarrow E[*] \mid (r \leftarrow !V)$$

Recall what ELL suggests

If $P \rightarrow P'$ by a redex at depth i , the following should hold:

- ▶ the depth of the program does not increase
- ▶ 😞 the duplication of occurrences should only happen at depth $> i$

The impact of side effects on depth

$$(\lambda x. \text{set}(r, x); !\text{get}(r))!V \xrightarrow{*} !!V$$

Recall what ELL suggests

If $P \rightarrow P'$ by a redex at depth i , the following should hold:

- ▶ 😞 the depth of the program does not increase
- ▶ the duplication of occurrences should only happen at depth $> i$

The impact of side effects on depth

$$M_r = \text{let } !z = \text{get}(r) \text{ in } !(z*)$$

$$\begin{array}{l} \xrightarrow{*} \quad !(M_r) \mid (r \leftarrow !(\lambda y.M_r)) \\ \xrightarrow{*} \quad !!(M_r) \mid (r \leftarrow !(\lambda y.M_r)) \\ \xrightarrow{*} \quad !!!(M_r) \mid (r \leftarrow !(\lambda y.M_r)) \\ \xrightarrow{*} \quad \dots \end{array}$$

Recall what ELL suggests

If $P \rightarrow P'$ by a redex at depth i , the following should hold:

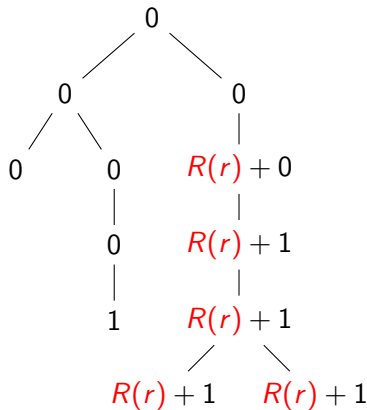
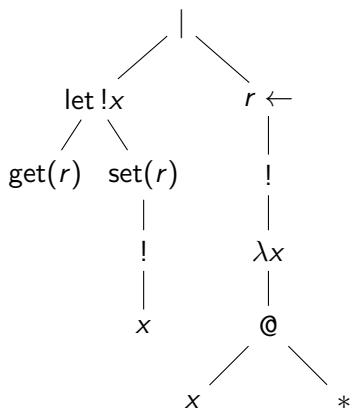
- ▶ 😞 the depth of the program does not increase
- ▶ the duplication of occurrences should only happen at depth $> i$

A revised notion of depth

- ▶ Define a **region context** $R = r_1 : \delta_1, \dots, r_n : \delta_n$
- ▶ The depth is computed **relatively** to a region context

A revised notion of depth

- Define a **region context** $R = r_1 : \delta_1, \dots, r_n : \delta_n$
- The depth is computed **relatively** to a region context



We have $d(P) = R(r) + 1$

Extended depth system

- ▶ **New depth judgement:** $R; \Gamma \vdash^\delta M$
- ▶ **Key point:** Every read and write on a given region should happen at the depth specified by the region context

$$\frac{}{R; \Gamma \vdash^\delta r} \quad \frac{}{R; \Gamma \vdash^\delta *}$$
$$\frac{}{R, r : \delta; \Gamma \vdash^\delta \text{get}(r)} \quad \frac{R, r : \delta; \Gamma \vdash^\delta V}{R, r : \delta; \Gamma \vdash^\delta \text{set}(r, V)}$$
$$\frac{R, r : \delta; \Gamma \vdash^\delta V}{R, r : \delta; \Gamma \vdash^0 (r \leftarrow V)} \quad \frac{R; \Gamma \vdash^\delta P_i \quad i = 1, 2}{R; \Gamma \vdash^\delta (P_1 \mid P_2)}$$

Some examples

A well-formed program

$$\frac{\frac{R; \Gamma, x : 1 \vdash^1 x}{R; \Gamma, x : 1 \vdash^0 !x} \quad \frac{R; \Gamma \vdash^0 \text{get}(r) \quad R; \Gamma, x : 1 \vdash^0 \text{set}(r, !x)}{R; \Gamma \vdash^0 \text{let } !x = \text{get}(r) \text{ in set}(r, !x)} \quad \frac{\vdots}{R; \Gamma \vdash^0 (r \leftarrow !(\lambda x. x*))}}{R; \Gamma \vdash^0 \text{let } !x = \text{get}(r) \text{ in set}(r, !x) \mid (r \leftarrow !(\lambda x. x*))}$$

with $R(r) = 0$

Some examples

The following reduction becomes harmless

$$E[\text{set}(r, !V)] \rightarrow E[*] \mid (r \leftarrow !V)$$

😊 the duplication of occurrences should only happen at depth $> i$

The following two programs are not well-formed

- ▶ $(\lambda x. \text{set}(r, x); !\text{get}(r))!V$
- ▶ $!(M_r) \mid (r \leftarrow !(\lambda y. M_r))$ with $M_r = \text{let } !z = \text{get}(r) \text{ in } !(z*)$

Properties

Subject reduction

If $R; \Gamma \vdash^\delta P$ and $P \rightarrow P'$ then $R; \Gamma \vdash^\delta P'$

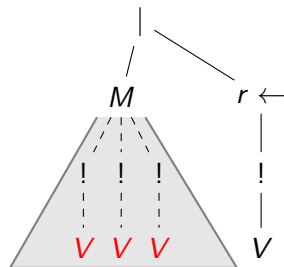
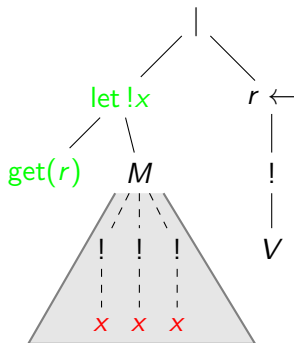
Termination

Let P be a well-formed term with $\alpha = d(P)$.

If $P \rightarrow P'$ then $\mu_\alpha(P) > \mu_\alpha(P')$.

Termination: the crucial case

$$E[\text{let } !x = \text{get}(r) \text{ in } M] \mid (r \leftarrow !V) \rightarrow E[M[V/x]] \mid (r \leftarrow !V)$$



$\mu_\alpha(P')$ is strictly decreasing exactly as in the functional $\text{let } !$ rule

Towards elementary time

Elementary bound

Let P be a well-formed program with $\alpha = d(P)$.

If $P \rightarrow P'$ then $t_\alpha(\mu_\alpha(P)) > t_\alpha(\mu_\alpha(P'))$.

Elementary time normalisation

The normalisation of well-formed programs can be performed in time elementary in the size of the terms.

3 - An elementary affine type system

Towards a type system

Well-formed programs can deadlock:

$$\text{let } !y = (\lambda x.x) \text{ in } !(yy)$$

We introduce types:

$t, t', \dots$	(Type variables)
$\alpha ::= \mathbf{B} \mid A$	(Types)
$A ::= t \mid \mathbf{1} \mid A \multimap \alpha \mid !A \mid \forall t.A \mid \text{Reg}_r A$	(Value-types)
$\Gamma ::= x_1 : (\delta_1, A_1), \dots, x_n : (\delta_n, A_n)$	(Variable contexts)
$R ::= r_1 : (\delta_1, A_1), \dots, r_n : (\delta_n, A_n)$	(Region contexts)

An elementary affine type system (excerpt)

$$\begin{array}{c}
 \frac{R; \Gamma \vdash^{\delta+1} M : \mathbf{A}}{R; \Gamma \vdash^{\delta} !M : \mathbf{A}} \qquad \frac{R; \Gamma \vdash^{\delta} M : \mathbf{!A} \quad R; \Gamma, x : (\delta + 1, \mathbf{A}) \vdash^{\delta} N : \mathbf{B}}{R; \Gamma \vdash^{\delta} \text{let } !x = M \text{ in } N : \mathbf{B}} \\
 \\
 \frac{r : (\delta, \mathbf{A}) \in R \quad R \vdash \Gamma}{R; \Gamma \vdash^{\delta} \text{get}(r) : \mathbf{A}} \qquad \frac{r : (\delta, \mathbf{A}) \in R \quad R; \Gamma \vdash^{\delta} V : \mathbf{A}}{R; \Gamma \vdash^{\delta} \text{set}(r, V) : \mathbf{1}} \\
 \\
 \frac{r : (\delta, \mathbf{A}) \in R \quad R; \Gamma \vdash^{\delta} V : \mathbf{A}}{R; \Gamma \vdash^0 (r \leftarrow V) : \mathbf{B}} \qquad \frac{P_i \text{ not a store } i = 1, 2 \quad R; \Gamma \vdash^{\delta} P_i : \alpha_i}{R; \Gamma \vdash^{\delta} (P_1 \mid P_2) : \mathbf{B}}
 \end{array}$$

- **additive** contexts
- no distinction between affine/intuitionistic hypotheses

Properties

Subject reduction

If $R; \Gamma \vdash^\delta P : \alpha$ and $P \rightarrow P'$ then $R; \Gamma \vdash^\delta P' : \alpha$.

Progress

If P is a stuck and (closed) typable program then

$$P \quad \equiv \quad M_1 \mid \cdots \mid M_m \mid S_1 \mid \cdots \mid S_n$$

where M_i is either a value or a stuck read (region is empty)

Conclusion

- ▶ A nice **combinatorial proof** of elementary time normalization
- ▶ **Effectful iteration** on inductive data types is typable

Future work

- ▶ Implementing **real-world** multithreaded algorithms
- ▶ We are working on a **polynomial time** depth system (SLL)