The theory of explicit substitutions revisited

Delia Kesner

PPS, Université Paris VII

Motivations

Many different calculi with ES developed in the literature : a need to stand back in order to related first formalisms with last results/technology.

A first attempt

Syntax for $\lambda\mathtt{x}$-terms :

$$t, u ::= x \mid (t\ u) \mid \lambda x.t \mid t[x/u]$$

Reduction system :

$$
\begin{array}{llll}
(\lambda x.t)\ u & \rightarrow_{\mathtt{B}} & t[x/u] & \\
(t\ u)[x/v] & \rightarrow_{\mathtt{x}} & (t[x/v]\ u[x/v]) & \\
(\lambda y.t)[x/v] & \rightarrow_{\mathtt{x}} & \lambda y.t[x/v] & \text{if } y \notin \mathtt{fv}(v)\ \&\ x \neq y \\
x[x/u] & \rightarrow_{\mathtt{x}} & u & \\
t[x/u] & \rightarrow_{\mathtt{x}} & t & \text{if } x \notin \mathtt{fv}(t)
\end{array}
$$

Some observations

- This is the minimal behaviour we can expect to implement substitution.
- No modelisation of simultaneous substitution.
- Lambda are crossed by substitutions and named variables are used, so α-equivalence is needed.
- Different syntax or restricted notions of reduction which do not require α-conversion are more adapted for implementation.

Explicit substitution research

de Bruijn'72, Curien'83, Ehrhard'88, Field'90, Revesz'88,
Cardelli'89, Abadi'89, Lévy'89

Starting from 1989 :

Ayala, Bloo, Bonelli, de Paiva, David, Dougherty, Dowek, Ferreira,
Geuvers, Goubault, Guillaume, Hardin, Herbelin, Hirschkoff,
Kamareddine, Kesner, Kirchner, Lang, Lengrand, Lescanne,
Mackie, Melliès, Nadathur, Pagano, Pfenning, Puel, Ríos, Ritter,
Rose, Stehr, Tasistro, van Oostrom, . . .

Why so much calculi ?

We expect these calculi to enjoy some properties :

CR, SN, PSN, SIM, FC

In more detail

Take

Z : a calculus to handle explicit substitutions/ressources

B : some rules to start computation

We will consider different reduction relations

$$\lambda_Z = B \cup Z.$$

In more detail

(CR) Confluence on metaterms :

If	v	${}^{*}_{\lambda_Z}\!\!\longleftarrow$	t	$\longrightarrow^{*}_{\lambda_Z}$	u
Then	v	$\longrightarrow^{*}_{\lambda_Z}$	t'	${}^{*}_{\lambda_Z}\!\!\longleftarrow$	u

(SIM) Simulation of one-step β-reduction :

Let $T : \lambda \mapsto \lambda_Z$. If $t \longrightarrow_{\beta} t'$, then $T(t) \longrightarrow^{*}_{\lambda_Z} T(t')$.

(FC) Implementation of full composition :

Any term of the form $t[y/v]$ can be λ_Z-reduced to $t\{y/v\}$.

(SN) Strong Normalisation :

If t is well-typed in an appropriate type system, then there is no infinite λ_Z-reduction sequence starting at t.

(PSN) Preservation of Strong Normalisation :

Let $T : \lambda \mapsto \lambda_Z$. If t is β-strongly normalising, then $T(t)$ is λ_Z-strongly normalising.

Summary of properties

Calculus	CR	SN	PSN	SIM	FC
$\lambda_{\upsilon}\lambda_s\lambda_t\lambda_u\lambda_{\mathbf{x}}\lambda_d\lambda_{dn}\lambda_e\lambda_f$	No	Yes	Yes	Yes	No
$\lambda_{\sigma}\lambda_{\sigma SP}$	No	No	No	Yes	Yes
$\lambda_{\sigma\Uparrow}\lambda_{se}\lambda_{\mathcal{L}}$	Yes	No	No	Yes	Yes
λ_{ζ}	Yes	Yes	Yes	No	No
λ_l	Yes	Yes	Yes	Yes	No
λ`lxr`	?	Yes	Yes	Yes	Yes

λ`lxr` is combinatorial complex : 6 equations and 19 rules !

Why λ`lxr` enjoys all the good properties we expect ?

Which is the essential computational dynamics of λ`lxr` ?

What is the logical meaning of a sound explicit substitution calculi ?

Typed λx (revisited)

$$\frac{}{x : A \vdash x : A} \ (ax)$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \rightarrow B} \ (\rightarrow i1) \qquad \frac{\Gamma \vdash t : B}{\Gamma \vdash \lambda x.t : A \rightarrow B} \ (\rightarrow i2)$$

$$\frac{\Gamma \vdash t : B \rightarrow A \quad \Delta \vdash u : B}{\Gamma \uplus \Delta \vdash (t\ u) : A} \ (\rightarrow e)$$

$$\frac{\Gamma \vdash u : B \quad \Delta, x : B \vdash t : A}{\Gamma \uplus \Delta \vdash t[x/u] : A} \ (cut1) \qquad \frac{\Gamma \vdash u : B \quad \Delta \vdash t : A}{\Gamma \uplus \Delta \vdash t[x/u] : A} \ (cut2)$$

We denote by $\Gamma \vdash_{\lambda\mathtt{rx}} t : A$ the derivability/typing relation.

A refined notion of reduction

$$
\begin{array}{llll}
(\lambda x.t)\ u & \rightarrow_{\mathtt{B}} & t[x/u] & \\
x[x/u] & \rightarrow_{\mathtt{rx}} & u & \\
t[x/u] & \rightarrow_{\mathtt{rx}} & t & \text{if } x \notin \mathtt{fv}(t) \\
(\lambda y.t)[x/v] & \rightarrow_{\mathtt{rx}} & \lambda y.t[x/v] & \text{if } y \notin \mathtt{fv}(v)\ \&\ x \neq y \\
(t\ u)[x/v] & \rightarrow_{\mathtt{rx}} & (t[x/v]\ u[x/v]) & \text{if } x \in \mathtt{fv}(t)\ \&\ x \in \mathtt{fv}(u) \\
(t\ u)[x/v] & \rightarrow_{\mathtt{rx}} & (t\ u[x/v]) & \text{if } x \notin \mathtt{fv}(t)\ \&\ x \in \mathtt{fv}(u) \\
(t\ u)[x/v] & \rightarrow_{\mathtt{rx}} & (t[x/v]\ u) & \text{if } x \in \mathtt{fv}(t)\ \&\ x \notin \mathtt{fv}(u) \\
t[x/u][y/v] & \rightarrow_{\mathtt{rx}} & t[y/v][x/u[y/v]] & \text{if } y \in \mathtt{fv}(u)\ \&\ y \in \mathtt{fv}(t) \\
t[x/u][y/v] & \rightarrow_{\mathtt{rx}} & t[x/u[y/v]] & \text{if } y \in \mathtt{fv}(u)\ \&\ y \notin \mathtt{fv}(t)
\end{array}
$$

Operational semantics for $\lambda\mathtt{rx}$

First define a natural equivalence for $\lambda\mathtt{rx}$:

$$t[x/u][y/v] \equiv t[y/v][x/u] \text{ if } y \notin \mathtt{fv}(u) \;\&\; x \notin \mathtt{fv}(v)$$

Then define a reduction relation modulo as follows :

$$t \longrightarrow_{\lambda\mathtt{rx}} t' \text{ iff } t \equiv u \longrightarrow_{\mathtt{B}\cup\mathtt{rx}} u' \equiv t'$$

Coming back to the summary

Calculus	CR	SN	PSN	SIM	FC
$\lambda_v \lambda_s \lambda_t \lambda_u \lambda_{\mathtt{x}} \lambda_d \lambda_{dn} \lambda_e \lambda_f$	No	Yes	Yes	Yes	No
$\lambda_\sigma \lambda_{\sigma SP}$	No	No	No	Yes	Yes
$\lambda_{\sigma\Uparrow} \lambda_{se} \lambda_{\mathcal{L}}$	Yes	No	No	Yes	Yes
λ_ζ	Yes	Yes	Yes	No	No
λ_l	Yes	Yes	Yes	Yes	No
$\lambda\mathtt{lxr}$	?	Yes	Yes	Yes	Yes
$\lambda\mathtt{rx}$	Yes	Yes	Yes	Yes	Yes

Fragility of composition : how PSN/SN can be lost

Consider the weaker rule

$$t[x/u][y/v] \quad \longrightarrow \quad t[x/u[y/v]] \quad \text{if } y \notin \mathtt{fv}(t)$$

instead of our rule

$$t[x/u][y/v] \quad \longrightarrow \quad t[x/u[y/v]] \quad \text{if } y \notin \mathtt{fv}(t) \;\&\; y \in \mathtt{fv}(u)$$

Mèllies has shown that there is a typable term that admits an infinite reduction sequence in the system containing the $\longrightarrow$ rule.

Connections with Linear Logic

Control of ressources in Linear Logic/Languages

- In logic : every hypothesis must be consumed *exactly once* in a proof (*two* occurrences of A cannot be derived from just *one*).
- In a programming language : it is not possible to *duplicate* variables.

A larger fragment, called Multiplicative Exponential Linear Logic (MELL), is able to encode intuitionistic and classical logics so that weakening/erasure and contraction/duplication become explicit operations.

Multiplicative Exponential Linear Logic (Girard)

The set of formulae is defined by the following grammar :

$$A, B \ ::= \ p \mid p^{\perp} \mid ?A \mid !A \mid A \otimes B \mid A \parr B$$

Linear negation of formulae is defined by

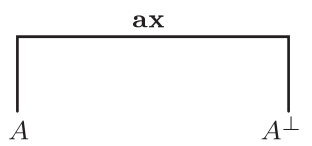
$p^{\perp}$	$:=$	$p^{\perp}$	$(?A)^{\perp}$	$:=$	$!(A^{\perp})$	$(A \otimes B)^{\perp}$	$:=$	$A^{\perp} \parr B^{\perp}$
$(p^{\perp})^{\perp}$	$:=$	p	$(!A)^{\perp}$	$:=$	$?(A^{\perp})$	$(A \parr B)^{\perp}$	$:=$	$A^{\perp} \otimes B^{\perp}$

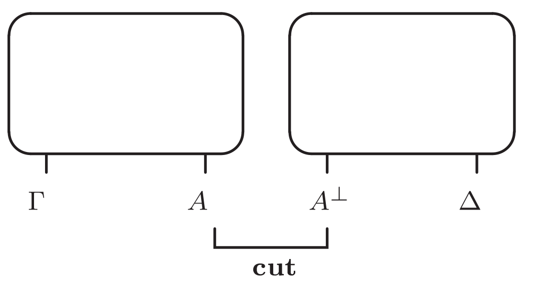
Proofs can be denoted for example by

- Trees of sequents — which contain too many syntactic details, or by
- Proof-nets — which eliminate unnecessary bureaucracy

Proof-Nets - Syntax

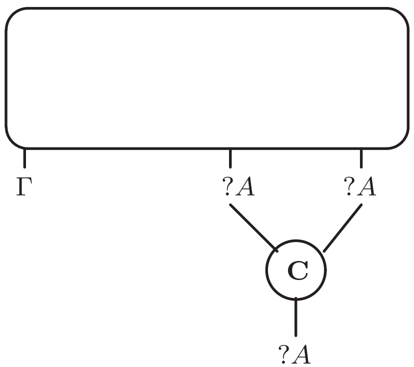
Axiom and Cut :

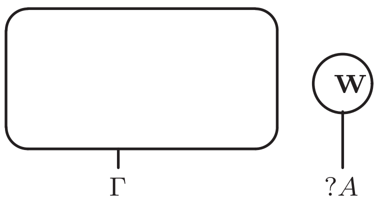




Proof-Nets - Syntax

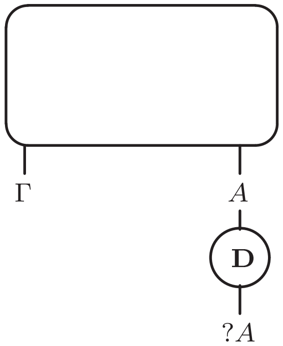
Contraction and Weakening :

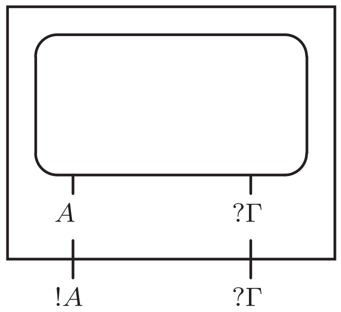




Proof-Nets - Syntax

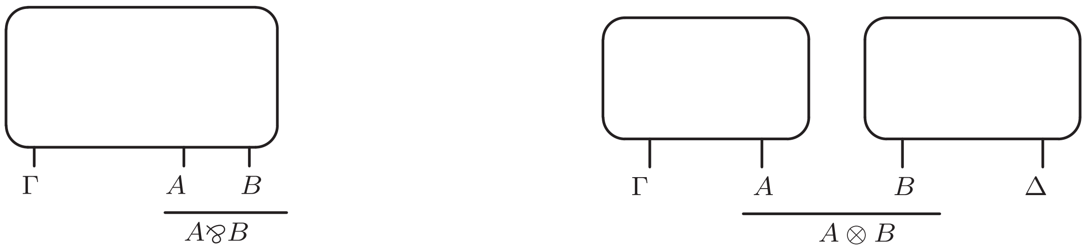
Dereliction and Box :





Proof-Nets - Syntax

Par and Times :

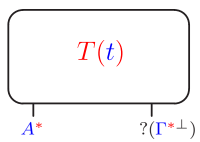


Proof-Nets - The reduction relation

- Reduction rules are used to perform cut elimination.
- Equivalence equations are used to identify proofs that only differ in structural details.

The resulting reduction relation is written R/E.

From λrx-terms to MELL proof-nets

Encoding	a $\lambda\mathtt{rx}$	into a MELL
$\_^{*}$	Type	Formula
$T(\_)$	Typed Term	Proof-net
	$\Gamma \vdash t : A$	

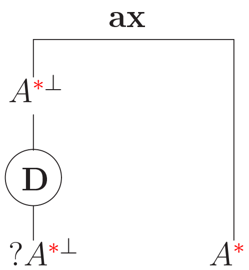
Encoding types

$$A^{*} \quad := \quad A \qquad \text{if } A \text{ is an atomic type}$$

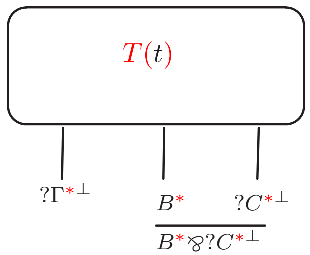
$$(A \to B)^{*} \quad := \quad ?((A^{*})^{\perp}) \parr B^{*}$$

Encoding Typing Derivations - some examples

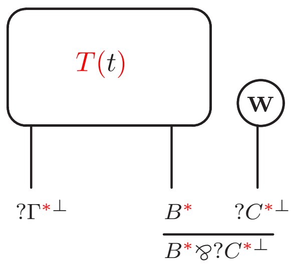
$T(x : A \vdash x : A)$ is



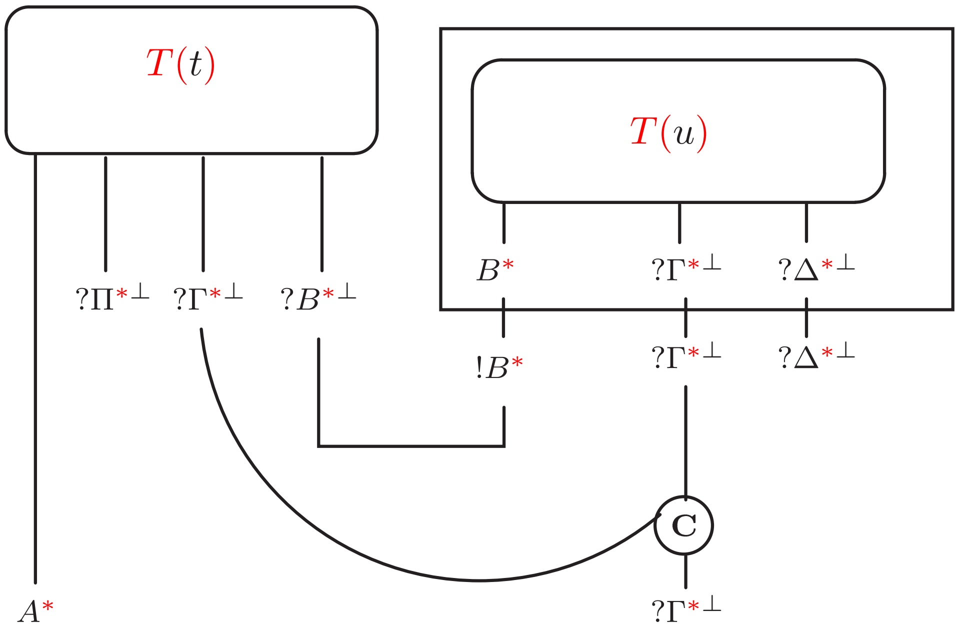
$T(\Gamma \vdash \lambda x.t : B \to C)$ where $x \in \mathtt{fv}(t)$ is



$T(\Gamma \vdash \lambda x.t : B \rightarrow C)$ where $x \notin \mathtt{fv}(t)$ is



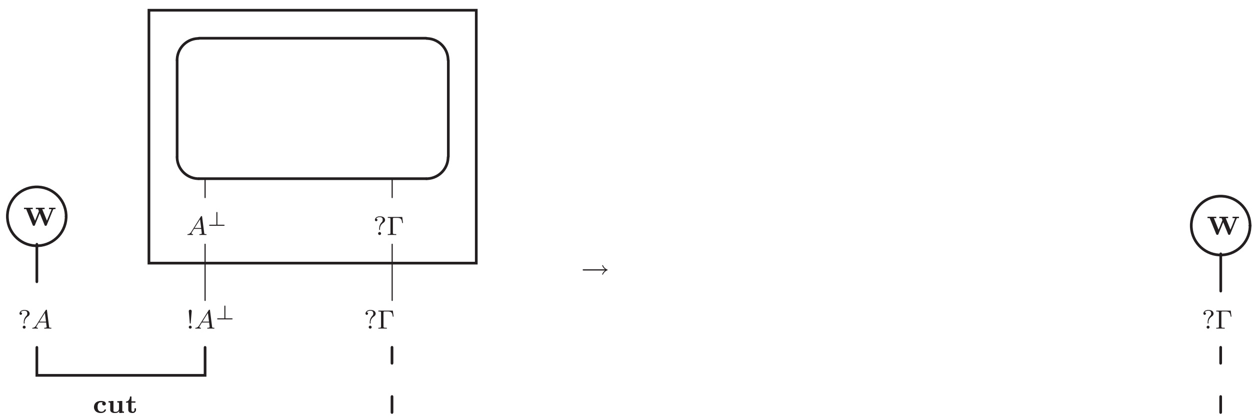
$T(\Pi, \Gamma, \Delta \vdash t[x/u] : A)$ where $x \in \mathtt{fv}(t)$ is



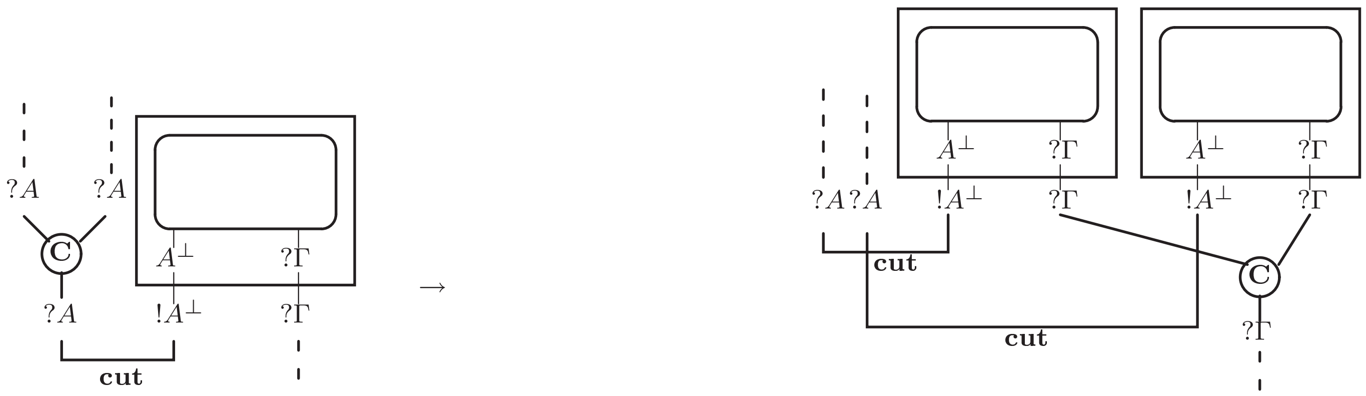
Some reduction rules

Decrease the complexity of the cut-formula

Erase a box

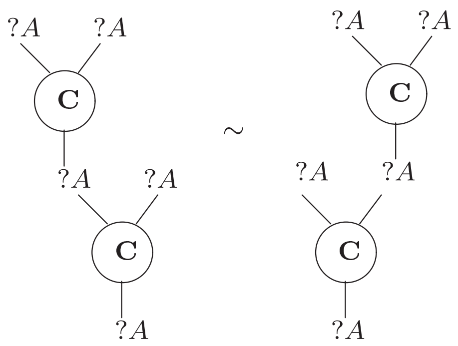


Duplicate a box



Some equivalence equations

Associativity of contraction :



Strong normalisation

Moreover, $T(\_)$ allows the simulation :

– If $t \equiv t'$ then $T(t) = T(t')$

– If $t \longrightarrow_{\mathtt{B}} t'$ then $T(t) \longrightarrow^{+}_{R/E} T(t')$

– If $t \longrightarrow_{\mathtt{rx}} t'$ then $T(t) \longrightarrow^{*}_{R/E} T(t')$

Since $\longrightarrow_{R/E}$ is strongly normalising on proof-nets, then we can conclude with the promised result

Corollary The reduction relation $\lambda\mathtt{rx}$ is strongly normalising for $\lambda\mathtt{rx}$-typed terms.

The key tools

– Equivalence relation on terms modelling simultaneous substitution.
– Controlled composition of substitutions.

A reduction system without equations

Terms *and* **Substitutions**

$$t ::= x \mid (t\ t) \mid \lambda x.t \mid t[s] \mid t(s)$$

$$s ::= id \mid x/u.s \mid s \circ s$$

Reduction Rules

$(\lambda x.t)\ u$	$\longrightarrow$	$t[x/u]$	
$(t\ u)[s]$	$\longrightarrow$	$(t[s]\ u[s])$	
$(\lambda x.t)[s]$	$\longrightarrow$	$\lambda x.t[s]$	
$x[(x/u).s]$	$\longrightarrow$	u	
$t[(x/u).s]$	$\longrightarrow$	$t[s]$	If $x \notin \mathtt{fv}(t)$
$t[s][p]$	$\longrightarrow$	$t[s \circ p]$	
$(s \circ p) \circ q$	$\longrightarrow$	$s \circ (p \circ q)$	
$id \circ s$	$\longrightarrow$	s	
$x[id]$	$\longrightarrow$	x	
$(x/u.s) \circ p$	$\longrightarrow$	$x/u(t).(s \circ p)$	
$u(id)$	$\longrightarrow$	id	
$u(x/v.s)$	$\longrightarrow$	$u[x/v](s)$	If $x \in \mathtt{fv}(u)$

To be translated to de Bruijn...

Conclusion

- Difficult problems in the domain of explicit substitution have been solved with logical tools.
- Linear Logic provides a natural framework to model (low level) languages to implement functional programming.
- Explicit operators for erasure, duplication and substitution provide fine operators for control ressources.