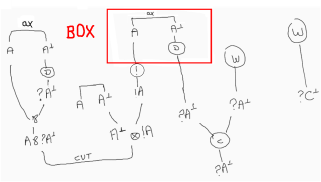
LAMBDA CALCULI THROUGH THE LENS OF LINEAR LOGIC

Lesson 2: **A Lambda-Calculus Inspired from Linear Logic**

Delia KESNER
Email : kesner@irif.fr
URL : www.irif.fr/~kesner

Université Paris Cité and CNRS

What this talk is about?

Graphical Formalism	Term Calculus
Girard's intuitionistic **MELL** **p**roof-**n**ets	Functional language called pn
	Lambda Calculus with Explicit Substitutions (ES) $(\lambda y.\lambda x.yx)[x\backslash w]u$
Only **BOXES** can be erased/duplicated	Only **ARGUMENTS** can be erased/duplicated
Linear Context (outside **BOX**) **Non-Linear** Context (inside **BOX**)	**Linear** Context (outside **ARGUMENT**) **Non-Linear** Context (inside **ARGUMENT**)
Reduction **free of bureaucracy** **Local** Reduction	Reduction **free of commutative rules** Reduction **at a distance**: crosses **contexts**
Convenient for **semantical studies** and **abstract reasoning**	Convenient for language **implementation** and **inductive reasoning**

Agenda for Today

Agenda

The λ-Calculus - Syntax

$$\begin{array}{ll} \textbf{Terms}: & t,u ::= x \mid \lambda x.t \mid tu \\ \textbf{Contexts}: & \mathsf{C} ::= \Diamond \mid \lambda x.\mathsf{C} \mid \mathsf{C}t \mid t\mathsf{C} \end{array}$$

- We use $\mathbf{fv}(t)$ (resp. $\mathbf{bv}(t)$) to denote the set of free (resp. bound) variables of t.
- We use a **meta-level** operation $t\{\{x\backslash u\}\}$ which simultaneously replaces all the **free** occurrences of x in t by u.
- We work modulo **alpha-conversion** (renaming of bound variables) generated by the equation: $\lambda x.t \equiv \lambda y.t\{\{x\backslash y\}\}$ where y is fresh. For example $\lambda x.xz = \lambda y.yz$.
- $\mathsf{C}\langle t\rangle$ denotes the context C where the hole $\Diamond$ has been replaced by t. Possible capture of free variables, e.g. $(\lambda x.\Diamond)\langle x\rangle = \lambda x.x$.
- $\mathsf{C}\langle\langle t\rangle\rangle$ denotes the context C where the hole $\Diamond$ has been replaced by t without capturing any free variables. e.g. $(\lambda x.\Diamond)\langle\langle y\rangle\rangle = \lambda x.y$ and $(\lambda x.\Diamond)\langle\langle x\rangle\rangle$ not defined.

The λ-Calculus - Operational Semantics

- Only one rewriting rule:

$$(\lambda x.t)\ u \mapsto_\beta t\{\!\{x\backslash u\}\!\}$$

- **The reduction relation $\to_\beta$** is generated by the relation $\mapsto_\beta$ closed by **all** contexts C:
  if $t \mapsto_\beta u$, then $\mathtt{C}\langle t\rangle \to_\beta \mathtt{C}\langle u\rangle$.

Alternative Definition:

$$\frac{}{(\lambda x.t)\ u \to_\beta t\{\!\{x\backslash u\}\!\}} \qquad \frac{t \to_\beta u}{\lambda x.t \to_\beta \lambda x.u} \qquad \frac{t \to_\beta u}{tv \to_\beta uv} \qquad \frac{t \to_\beta u}{vt \to_\beta vu}$$

Both definitions are taken modulo **alpha-conversion**.

Lambda-Calculus is Turing complete.

Examples

Let $\mathtt{Id} := \lambda z.z$. Then,

Erasing case: $(\lambda y.x)z \rightarrow_\beta \; x$

Duplicating case: $(\lambda y.yy)z \rightarrow_\beta \; zz$

Non-terminating case: $(\lambda y.yy)(\lambda y.yy) \rightarrow_\beta \; (\lambda y.yy)(\lambda y.yy) \rightarrow_\beta \; \ldots$

Contextual case: $\lambda z.(\lambda x.y)(\mathtt{Id}\;\mathtt{Id}) \rightarrow_\beta \; \lambda z.(\lambda x.y)\mathtt{Id} \rightarrow_\beta \; \lambda z.y$

Types: $A ::= \iota \mid A \to B$

Typing Rules (Natural Deduction Style):

$$\frac{}{x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i} \text{(axiom)}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B} (\to \ \text{intro}) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B} (\to \ \text{elim})$$

- Rule (axiom) uses weakening.
- Rule ($\to$ elim) is additive (implicit contraction).
- We denote by $\Gamma \vdash_\lambda t : A$ the corresponding derivability relation.
- A term t is **(simply) typable** if there exists a derivation $\Gamma \vdash_\lambda t : A$.

Some Salient Remarks about Simply Typed Lambda-Calculus

- Typical **Curry-Howard** correspondence (λ-calculus corresponds to minimal intuitionistic logic).
- Provides only **monomorphic** information.
- Lack expressivity power but typability is **decidable**.
- Typability **IMPLIES** Strong Normalization, but the converse does not hold.

E.g. the term $\lambda x.xx$ **is not** typable

The λ-Calculus - Some Typical Meta-Properties

Theorem (Confluence)

If $t \rightarrow_{\beta}^{*} u$ and $t \rightarrow_{\beta}^{*} v$, then there is t' such that $u \rightarrow_{\beta}^{*} t'$ and $v \rightarrow_{\beta}^{*} t'$.

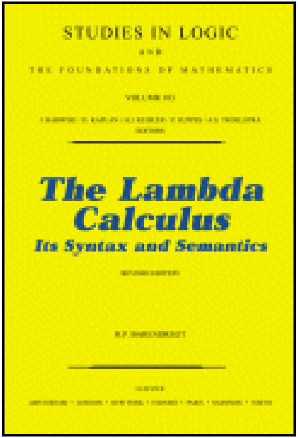
Theorem (Subject Reduction for Simple Types)

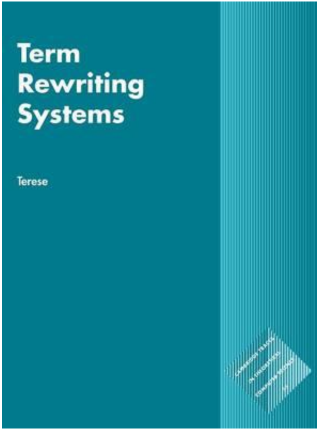
If $\Gamma \vdash_{\lambda} t : A$ and $t \rightarrow_{\beta} t'$, then $\Gamma \vdash_{\lambda} t' : A$.

Theorem (Strong Normalization)

If $\Gamma \vdash_{\lambda} t : A$, then $t \in \mathtt{SN}(\beta)$, i.e. there is no infinite β-reduction sequence starting at t (every β-reduction sequence starting at t terminates).

To Go Further, Recommended Readings





Our Goal Today

λ-calculus $\Rightarrow$ **intermediate language** $\Rightarrow$ MELL **P**roof-**N**ets

- MELL is an extension of Linear Logic being able to capture intuitionistic and classical logic.
- In MELL, weakening and contraction are handled in an explicit way by means of the modalities $?$ and $!$.

The intermediate language:

- Lambda-terms with explicit substitutions + equivalence + reduction rules
- Explicit management of resources (erasure and duplication)
- Different alternatives: Λ-calculus, Linear Substitution Calculus, $\lambda\mathtt{ex}$-calculus, $\mathtt{pn}$-calculus, $\lambda\mathtt{lxr}$-calculus, ...

Here, we focus on the $\mathtt{pn}$-calculus.

Agenda

The pn-Calculus: Basic Syntax

Terms	t, u, v	$::=$	$x \mid \lambda x.t \mid tu \mid t[x\backslash u]$
Term Contexts	$\mathtt{C}$	$::=$	$\diamond \mid \lambda x.\mathtt{C} \mid \mathtt{C}t \mid t\mathtt{C} \mid \mathtt{C}[x\backslash t] \mid t[x\backslash \mathtt{C}]$

- $[\_\backslash\_]$ is called an **Explicit Substitution** (**ES**)
- **Free** and **bound** variables (written $\mathbf{fv}(\_)$ and $\mathbf{bv}(\_)$ resp.).
- **Alpha-conversion**:

$$\begin{array}{lcll} \lambda x.t & \equiv & \lambda y.t\{\!\{x\backslash y\}\!\} & y \text{ fresh} \\ t[x\backslash u] & \equiv & t\{\!\{x\backslash y\}\!\}[y\backslash u] & y \text{ fresh} \end{array}$$

  Example $(x\ y)[x\backslash \lambda z.zw] = (x'\ y)[x'\backslash \lambda z'.z'\ w]$.
- **Pure** terms: terms without explicit substitutions.
- Notations: $\mathtt{C}\langle t\rangle$ (possible capture of free variables) and $\mathtt{C}\langle\!\langle t\rangle\!\rangle$ (capture-free).

The Key Notions of Contexts

Contexts make it possible to reason **at a distance**.

Linear Contexts: (Outside Arguments) $\mathtt{H} ::= \diamond \mid \lambda x.\mathtt{H} \mid \mathtt{H}t \mid \mathtt{H}[x\backslash t]$

Non-Linear Contexts: (Arguments) $\mathtt{A} ::= t\diamond \mid t[y\backslash\diamond]$

- **Substitution Contexts (Special Linear Contexts):** $\mathtt{L} = \diamond \mid \mathtt{L}[x\backslash t]$
- **Examples:**
  - $\mathtt{H}_1 = \lambda y.(\diamond y)z$
  - $\mathtt{L}_1 = \diamond[x_1\backslash x_2][x_2\backslash z]$
  - $\mathtt{A}_1 = y\diamond$.

Decomposing Terms Using Linear/Non-Linear Contexts

Property

Given $x \in \mathbf{fv}(u)$, then u can be written in one of the following forms:

- $\mathtt{H}\langle x\rangle$, with $x \notin \mathbf{fv}(\mathtt{H})$,
- $\mathtt{H}\langle \mathtt{A}\langle t\rangle\rangle$, with $x \notin \mathbf{fv}(\mathtt{H})$, $x \notin \mathbf{fv}(\mathtt{A})$, $x \in \mathbf{fv}(t)$,
- $\mathtt{H}\langle \mathtt{A}\langle t\rangle\rangle$, with $x \notin \mathbf{fv}(\mathtt{H})$, $x \in \mathbf{fv}(\mathtt{A})$, $x \in \mathbf{fv}(t)$,

Example:

- $u = \lambda y.\underline{x}yz$ can be written as $\mathtt{H}\langle x\rangle$, with $\mathtt{H} = \lambda y.\Diamond yz$,
- $u = ((y(\underline{x}y))z)$ can be written as $\mathtt{H}\langle \mathtt{A}\langle t\rangle\rangle$, with $\mathtt{H} = \Diamond z$, $\mathtt{A} = y\Diamond$ and $t = xy$
- $u = (((\underline{x}y)\underline{x})z)$ can be written as $\mathtt{H}\langle \mathtt{A}\langle t\rangle\rangle$, with $\mathtt{H} = \Diamond z$, $\mathtt{A} = (xy)\Diamond$ and $t = x$.

The pn-Calculus: Intuitions

- There are **only five basic operational rules** constituting the pn-reduction relation.
- They operate **at a distance**: they bypass **linear** and **non-linear** contexts.
- Only **arguments** of terms can be erased/duplicated.
- Intuition behind the five basic operational rules:
  - The β-reduction rules is **decomposed/refined** into different atomic actions
  - One possible action is to **fire a redex**, by delaying the computation of the created substitution.
  - Another possible action is to **erase** an argument.
  - Another possible action is to **linearly substitute** an occurrence of some variable by a term.
  - Another possible action is to **jump into an argument**.
  - Another possible action is to **duplicate** some argument

Fire a Redex

Rule: $\mathtt{L}\langle \lambda x.t\rangle u \quad \mapsto_{\mathtt{dB}} \quad \mathtt{L}\langle t[x\backslash u]\rangle$

Erase

Rule: $t[x\backslash u] \quad \mapsto_{\mathtt{gc}} \quad t \qquad x \notin \mathbf{fv}(t)$

Linearly Substitute

Rule: $\mathtt{H}\langle x\rangle[x\backslash u] \quad \mapsto_{\mathtt{lsubs}} \quad \mathtt{H}\langle u\rangle \qquad x \notin \mathbf{fv}(\mathtt{H})$

Jump into an Argument

Rule: $\mathtt{H}\langle \mathtt{A}\langle t\rangle\rangle[x\backslash u] \quad \mapsto_{\mathrm{arg}} \quad \mathtt{H}\langle \mathtt{A}\langle t[x\backslash u]\rangle\rangle \qquad x \in \mathbf{fv}(t), x \notin \mathbf{fv}(\mathtt{H}), \mathbf{fv}(\mathtt{A})$

Duplicate

Rule: $\mathtt{H}\langle \mathtt{A}\langle t\rangle\rangle[x\backslash u] \quad \mapsto_{\mathrm{dup}} \quad \mathtt{H}\langle \mathtt{A}_{[x\backslash u]}\langle t\rangle[x\backslash u]\rangle \qquad x \notin \mathbf{fv}(\mathtt{H}), x \in \mathbf{fv}(t), \mathbf{fv}(\mathtt{A})$

Definition

Example

$$
\begin{array}{ll}
((\lambda z.\lambda y.\lambda x.yxx)wu)v & \to_{\mathtt{dB}} \\
((\lambda y.\lambda x.yxx)[z\backslash w]u)v & \to_{\mathtt{gc}} \\
((\lambda y.\lambda x.yxx)u)v & \to_{\mathtt{dB}} \\
(\lambda x.yxx)[y\backslash u]v & \to_{\mathtt{dB}} \\
((yx)x)[x\backslash v][y\backslash u] & \to_{\mathtt{lsubs}} \\
((ux)x)[x\backslash v] & \to_{\mathtt{dup}} \\
((ux)[x\backslash v]x)[x\backslash v] & \to_{\mathrm{arg}} \\
((ux)[x\backslash v]x[x\backslash v]) & \to_{\mathtt{lsubs}} \\
((ux)[x\backslash v]v) & \to_{\mathrm{arg}} \\
(ux[x\backslash v])v & \to_{\mathtt{lsubs}} \quad ((uv)v)
\end{array}
$$

A New Definition of the Substitution Operation

Defining the substitution operation $t\{\{x\backslash u\}\}$:

- By induction on the **structure** of t.
- By induction on the **# of free occurrences** of x in t.
- Our notion of substitution is a **MIX** of the two:

Structure of t : Rules $\{$ $\rightarrow_{\text{arg}}$: **Jump into an Argument**

# of free occurrences of x in t: Rules

- $\rightarrow_{\texttt{gc}}$: **Erase** (0 ocurrences)
- $\rightarrow_{\texttt{lsubs}}$: **Linearly Substitute** (1 ocurrence)
- $\rightarrow_{\texttt{dup}}$: **Duplicate** (more than 1 ocurrence)

Some Properties

Lemma (Stability of Free Variables)

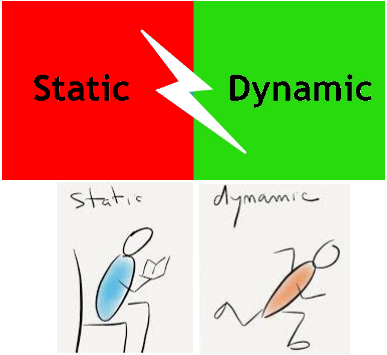
- *If* $t \rightarrow_{\mathtt{dB,var,arg,dup}} u$*, then* $\mathbf{fv}(t) = \mathbf{fv}(u)$*.*
- *If* $t \rightarrow_{\mathtt{gc}} u$*, then* $\mathbf{fv}(t) \supseteq \mathbf{fv}(u)$*.*

Remark In λ-calculus no special rule for erasing.

Has anyone noticed any similarity between the operational semantics of pn and the operational semantics of MELL proof nets?

Agenda

Relation Between the pn-Calculus and MELL Proof-Nets



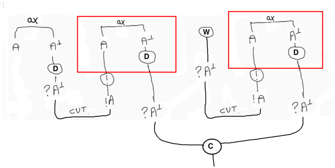
Static Translation	**Dynamic Translation**
Typed pn-Terms to **Linear Logic Proof-Nets**	**Reduction Steps** on **pn-Terms** to **Reduction Steps** on **Linear Logic Proof-Nets**

The Static Translation - Some Intuitions

- Linear contexts of terms (outside **arguments**) are translated to linear contexts of proof-nets (outside **BOXES**).
- Non-linear contexts of terms (**arguments**) are translated to non-linear contexts of proof-nets (**BOXES**).
- **Arguments** of applications and substitutions (that can be erased/duplicated) are translated to **BOXES** (that can be erased/duplicated).
- Linear variables are translated to **D**erelicted axioms (variable y).
- Void variables are translated to **W**eakening (variable z).
- Duplicated variables are translated to **C**ontraction (variable w).

Example:

$y[y\backslash w][z\backslash w]$



$$\frac{}{x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i} \text{(axiom)}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B} (\to \text{ intro}) \qquad \frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B} (\to \text{ elim})$$

$$\frac{\Gamma \vdash u : B \quad \Gamma, x : B \vdash t : A}{\Gamma \vdash t[x\backslash u] : A} \text{(cut)}$$

Remark

- The axiom rule uses weakening.
- The binary rules use contraction.
- If $\Gamma \vdash t : A$ is derivable, then $\mathbf{fv}(t) \subseteq \mathbf{dom}(\Gamma)$.

Simple Types for Explicit Substitutions - Multiplicative System

$$\frac{}{x : A \vdash x : A}\ \text{(ax)} \qquad \frac{\Gamma \vdash t : A \to B \quad \Delta \vdash u : A}{\Gamma \cup \Delta \vdash tu : B}\ (\to \text{ e})$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\to \text{ i1}) \qquad \frac{\Gamma \vdash t : B \quad x \notin \mathbf{fv}(t)}{\Gamma \vdash \lambda x.t : A \to B}\ (\to \text{ i2})$$

$$\frac{\Gamma \vdash u : B \quad \Delta, x : B \vdash t : A}{\Gamma \cup \Delta \vdash t[x\backslash u] : A}\ \text{(cut1)}$$

$$\frac{\Gamma \vdash u : B \quad \Delta \vdash t : A \quad x \notin \mathbf{fv}(t)}{\Gamma \cup \Delta \vdash t[x\backslash u] : A}\ \text{(cut2)}$$

Remark

- No weakening and no contraction logical rules.
- If $\Gamma \vdash t : A$, then $\mathbf{fv}(t) = \mathbf{dom}(\Gamma)$.
- The additive and the multiplicative systems are equivalent.
- Reduction preserves types.
- Typed pn-terms are strongly normalizing.

Principles of the Translation

(Call-by-Name) Translation of Types

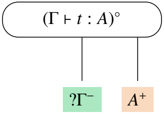
$$
\begin{array}{lcl}
\iota^+ & := & \iota \\
(A \to B)^+ & := & ?(A^-) \parr B^+ \\
A^- & := & (A^+)^\perp
\end{array}
$$

Remark:

- $(?(A^-) \parr B^+)^\perp = !A^+ \otimes B^-$.
- $(?A^-)^\perp = !A^+$.

Translation of Derivations

Let $\Gamma = x_1 : B_1, \ldots, x_n : B_n$. Then $\Gamma \vdash t : A$ translates to a MELL Proof-Net written $(\Gamma \vdash t : A)^\circ$ with interface $?\Gamma^-, A^+$, where $?\Gamma^-$ means $?B_1^-, \ldots, ?B_n^-$

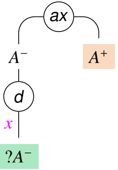


Translating (ax)

Original derivation:

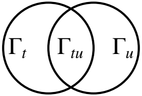
$$\frac{}{x : A \vdash x : A}\ (\mathrm{ax})$$

Proof-Net Translation:

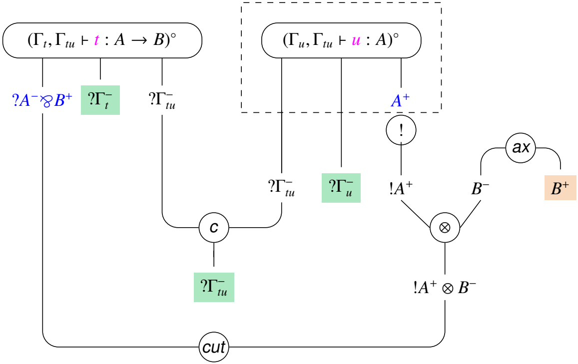


Original derivation:

$$\frac{\Gamma_t, \Gamma_{tu} \vdash t : A \to B \qquad \Gamma_u, \Gamma_{tu} \vdash u : A \qquad \Gamma_t \# \Gamma_u}{\Gamma_t , \Gamma_u , \Gamma_{tu} \vdash tu : B} \;(\to e)$$



Proof-Net Translation:

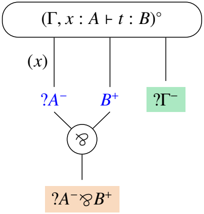


Translating ($\rightarrow$ i1)

Original derivation:

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

Proof-Net Translation:

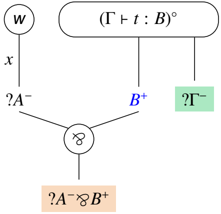


Translating ($\rightarrow$ i2)

Original derivation:

$$\frac{\Gamma \vdash t : B \qquad x \notin \mathbf{fv}(t)}{\Gamma \vdash \lambda x.t : A \rightarrow B} (\rightarrow \text{ i2})$$

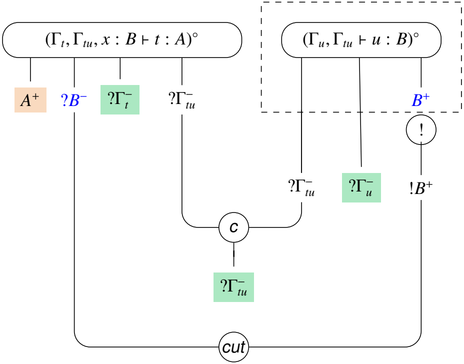
Proof-Net Translation:



Original derivation:

$$\frac{\Gamma_u, \Gamma_{tu} \vdash u : B \qquad \Gamma_t, \Gamma_{tu}, x : B \vdash t : A \qquad \Gamma_t \# \Gamma_u}{\Gamma_u, \Gamma_t, \Gamma_{tu} \vdash t[x \backslash u] : A} \text{(cut1)}$$

Proof-Net Translation:

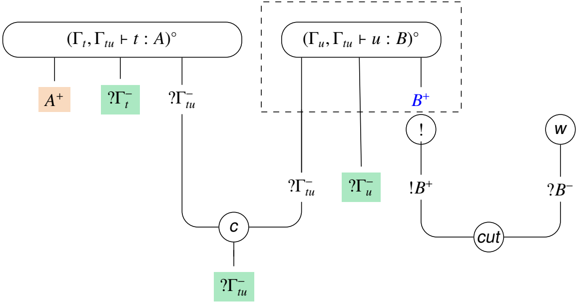


Translating (cut2)

Original derivation:

$$\frac{\Gamma_u, \Gamma_{tu} \vdash u : B \qquad \Gamma_t, \Gamma_{tu} \vdash t : A \qquad \Gamma_t \# \Gamma_u \qquad x \notin \mathbf{fv}(t)}{\Gamma_u, \Gamma_t, \Gamma_{tu} \vdash t[x \backslash u] : A}\ (\text{cut2})$$

Proof-Net Translation:



An Equivalence on pn-Terms

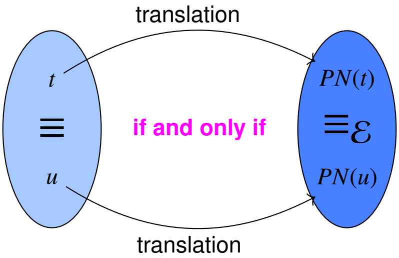
Equivalence (known as σ-equivalence by Regnier):

$$\lambda x.t \equiv \lambda y.t\{\{x\backslash y\}\} \quad y \text{ fresh}$$
$$t[x\backslash u] \equiv t\{\{x\backslash y\}\}[y\backslash u] \quad y \text{ fresh}$$
$$\mathtt{H}\langle t\rangle[x\backslash u] \equiv \mathtt{H}\langle t[x\backslash u]\rangle \quad \text{if } x \notin \mathbf{fv}(\mathtt{H}) \text{ and no capture of free variables}$$

Particular Instances:

$$t[y\backslash v][x\backslash u] \equiv t[x\backslash u][y\backslash v] \quad \text{if } y \notin \mathbf{fv}(u) \;\&\; x \notin \mathbf{fv}(v)$$
$$(\lambda y.t)[x\backslash u] \equiv \lambda y.t[x\backslash u] \quad \text{if no capture of free variables}$$
$$(tv)[x\backslash u] \equiv t[x\backslash u]v \quad \text{if } x \notin \mathbf{fv}(v)$$

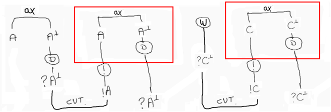
Static Relation Between the pn-Calculus and Linear Logic Proof-Nets

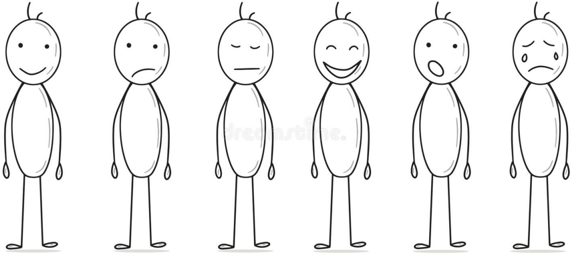


$\equiv$ captures **σ-equivalence** on pn-Terms	$\equiv_{\mathcal{E}}$ captures **structural equivalence** on MELL Proof-Nets

All the following pn-terms are all translated to the same Proof-Net:

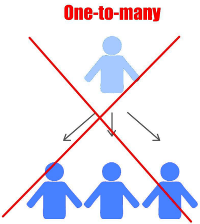
$(\lambda x.(\lambda y.y)w)z$	$((\lambda y.y)w)[x\backslash z]$	$y[y\backslash w][x\backslash z]$
$(\lambda x.(\lambda y.y))zw$	$(\lambda y.y)[x\backslash z]w$	$y[x\backslash z][y\backslash w]$
$(\lambda y.(\lambda x.y)z)w$	$(\lambda y.y[x\backslash z])w$	$(\lambda x.y)[y\backslash w]z$
$(\lambda y.(\lambda x.y))wz$	$(\lambda x.y[y\backslash w])z$	$((\lambda x.y)z)[y\backslash w]$





Agenda

Dynamic Relation Between pn-Terms and Linear Logic Proof-Nets



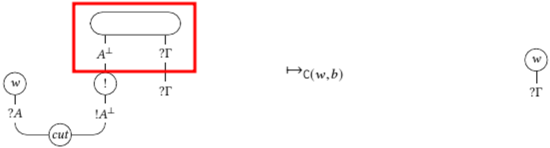
Reduction Rule on Terms	Reduction Rule on Proof-Nets
Fire a Redex	Multiplicative Cut Rules
Erase	Weakening-Box
Linearly Substitute	Dereliction-Box
Duplicate	Contraction-Box
Jump into an Argument	Box-Box

Erase

Erase a Term:

$$t[x\backslash u] \mapsto_{\mathsf{gc}} t \qquad x \notin \mathbf{fv}(t)$$

Erase a Proof-Net:

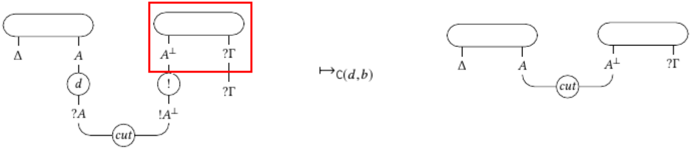


Linearly Substitute

Linearly Substitute:

$$\mathtt{H}\,\langle x\rangle[x\backslash u] \mapsto_{\mathtt{lsubs}} \mathtt{H}\,\langle u\rangle \qquad x \notin \mathbf{fv}(\mathtt{H})$$

Remove Modality:

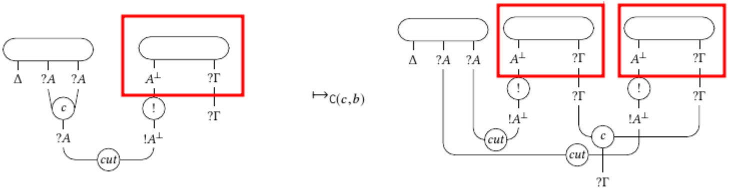


Duplicate

Duplicate a Term:

$$\mathtt{H}\langle\, \mathtt{A}\, \langle t\rangle\rangle[x\backslash u] \mapsto_{\mathrm{dup}} \mathtt{H}\langle\, \mathtt{A}_{[x\backslash u]}\, \langle t\rangle[x\backslash u]\rangle \qquad x \notin \mathbf{fv}(\mathtt{H}), x \in \mathbf{fv}(t), \mathbf{fv}(\mathtt{A})$$

Duplicate a Proof-Net:

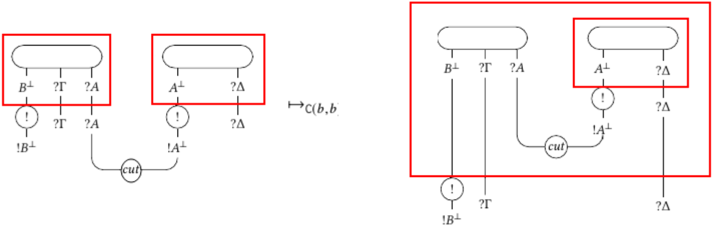


Jump into an Argument

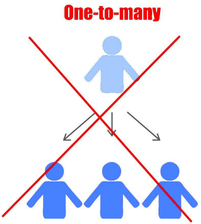
Jump into an Argument:

$$\mathtt{H}\langle\,\mathtt{A}\,\langle t\rangle\rangle[x\backslash u] \mapsto_{\text{arg}} \mathtt{H}\langle\,\mathtt{A}\,\langle t[x\backslash u]\rangle\rangle \qquad x \in \mathbf{fv}(t), x \notin \mathbf{fv}(\mathtt{H}), \mathbf{fv}(\mathtt{A})$$

Nesting:

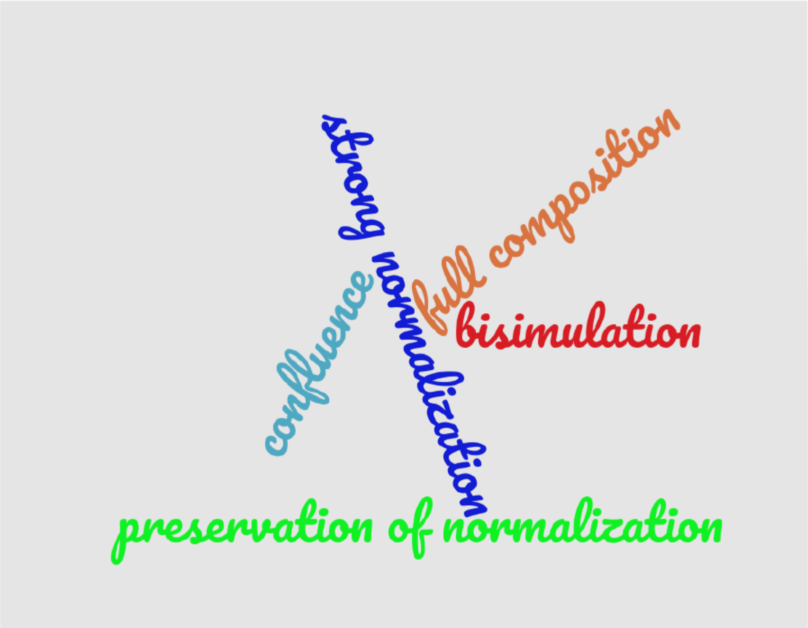


Dynamic Relation Between pn-Terms and Linear Logic Proof-Nets



This gives a **fine-grained** computational interpretation of Intuitionistic Proof-Nets.

Agenda


strong normalization
full composition
bisimulation
confluence
preservation of normalization

Full Composition

Property

The pn-calculus enjoys **full composition**:
every pn-term of the form $t[x\backslash u]$ reduces to $t\{\{x\backslash u\}\}$

Strong Bisimulation

Property

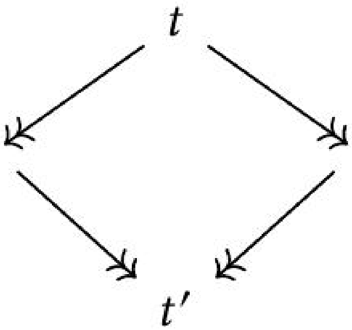
The relation $\equiv$ on pn-terms is a **strong bisimulation** w.r.t. the reduction relation $\rightarrow_{\mathrm{pn}}$:

$$\begin{array}{ccc} t & \equiv & u \\ \downarrow & & \downarrow \\ t' & \equiv & u' \end{array}$$

Confluence

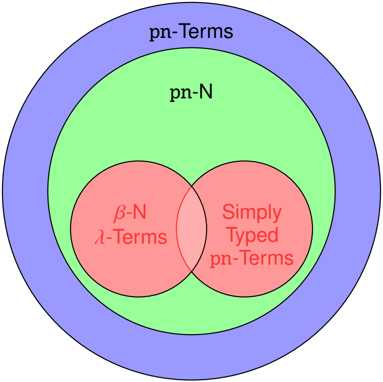
Property

The pn-calculus is **confluent** on **closed** and **open** terms (modulo the congruence $\equiv$).



Normalization Properties

- **PN**: The new reduction relation enjoys **preservation of β-normalization**: if t is β-normalizing (β-N), then t is also pn-normalizing (pn-N).
- **SN**: (Simply) typed pn-terms are pn-**normalizing**.



Discussion

- The design of a calculus with explicit substitution enjoying confluence on open terms and PN at the same time was an open open for many years.
- In particular, the calculus $\lambda\sigma$ of Abadi-Cardelli-Curien-Lévy does not enjoy PN, a result due to Melliès.
- Other calculi with explicit substitution enjoy the same properties we discussed today, but none of them has a so tight relation with Girard's Proof-Nets.

Agenda

Conclusion

Our Approach:

- **Bridges the gap** between term syntaxes and graphical formalisms for functional programming.
- Derives a **new (hybrid) notion of substitution** which mixes structural induction on terms with induction on the number of free variables to be substituted.
- The term calculus enjoys all the **good properties** one would expect from such kind of calculi.
- A nice formalism to academically **explain** the dynamics of Girard's intuitionistic linear logic proof-nets.

Other Possible Approaches:

- Add explicit weakenings and contractions to the term syntax:
  no more one-to-one dynamic correspondence.
- Change the reduction rules on proof-nets by using implicit quantification over boxes:
  complex operational semantics.

Possible Extensions:

- **Classical Logic:** Polarized Proof-Nets.


THANK
YOU
FOR
YOUR
ATTENTION


Time for
Questions