Lambdi Calculi through the Lens of Linear Logic — Delia Kesner

Lecture 2, A Lambda-Calculus Inspired from Linear Logic Proof-Nets - *July 1, 2025*

What this talk is about We want to find a correspondence between Simply-Typed λ-Calculus and Proof-Nets defined in previous lecture. Girard's intuitionistic MELL proof-nets and the pn-calculus offer two perspectives on computation, aligned across five dimensions. In terms of resource management, only boxes in proof-nets and only arguments in pn-calculus may be erased or duplicated. Both systems distinguish between linear and non-linear contexts: linear contexts lie outside boxes or arguments, while non-linear ones are internal. Reduction in proof-nets is local and avoids bureaucratic overhead, whereas in the pn-calculus it operates across contexts and is free of commutative rules. This leads to different operational styles—local rewriting in the graphical setting versus at-a-distance reduction in the syntactic one. Finally, each system serves a different purpose: proof-nets support semantical studies and abstract reasoning, while pn-calculus is better suited for implementation concerns and inductive analysis.

Contents

1 $\lambda-$Calculus: A Brief Reminder. 2

2 A Lambda Calculus with Explicit Substitutions (pn-calculus) 3

2.1 pn-Calculus Reduction Rules 4

3 Static and Dynamic Translation to MELL 4

3.1 Explicit Substitutions - Multiplicative System 5

3.2 Translation principles 5

3.3 Equivalence on pn-terms 6

3.4 Dynamic Translation: Term vs Proof-Net Reductions 7

4 Properties of the Calculus 10

5 Conclusion 11

References. 12

1 $\lambda-$Calculus: A Brief Reminder

Syntax of STLC (Simply-Typed Lambda-Calculus)

$$t, u ::= x \mid \lambda x.t \mid tu$$
$$C ::= \diamond \mid \lambda x.C \mid C\ t \mid t\ C$$

We also define following functions:

- $\mathbf{fv}(v)$ is the set of free variables of v
- $\mathbf{bv}(v)$ is the set of bound variables of v

Substitution and context application For this and other calculi, we use the following conventions:

- $t\,[\![x\backslash u]\!]$ is the capture-avoiding substitution, that replaces all free occurrences of x in t by u.
- we work modulo alpha-conversion, i.e.: $\lambda x.t \equiv \lambda y.t\{x\backslash y\}$ with y fresh.
- $C\,\langle t\rangle$ is a term created by replacing the hole ($\diamond$) with t, allowing binders in C to bind any free variables of t
- $C\,\langle\langle t\rangle\rangle$ is like $C\,\langle t\rangle$, but we do not allow for capture (it is undefined in such cases)

Operational Semantics We only have one rewriting rule the β-reduction:

$$(\lambda x.t)u \mapsto_\beta t\,[\![x\backslash u]\!]$$

The reduction relation $\to_\beta$ is generated by the relation $\mapsto_\beta$ closed under all contexts C if:

$$\forall C, t, u.t \mapsto_\beta u \implies C\,\langle t\rangle \to_\beta C\,\langle u\rangle$$

Examples

- Erasing: $(\lambda y.x)z \to_\beta x$
- Duplication: $(\lambda y.yy)z \to_\beta zz$
- Non-termination: $(\lambda y.yy)(\lambda y.yy) \to_\beta (\lambda y.yy)(\lambda y.yy) \to_\beta \ldots$
- Reduction under a non-trivial context (a lambda), where $\texttt{Id} := \lambda x.x$:

$$\lambda z.(\lambda x.y)(\texttt{Id Id}) \to_\beta \lambda z.(\lambda x.y)\texttt{Id} \to_\beta \lambda z.y$$

Types in STLC

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

Typing rules

$$\frac{}{x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i} \text{ (axiom)} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B} (\to \text{ intro})$$

$$\frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B} (\to \text{ elim})$$

We should notice here that the (axiom) rule implies weakening and the ($\to$ elim) rule implies contraction. We do not want to have these properties, or at least we want to make their use explicit, but we still want to keep other properties of STLC, like Confluence, Subject Reduction and Strong Normalization. To that end, we define intermediate calculus, that can later be translated directly into Proof-Nets.

2 A Lambda Calculus with Explicit Substitutions (pn-calculus)

To go from λ-calculus to MELL Proof-Nets, as seen in lecture 1. We use an intermediate language (i.e. λ-calculus $\Rightarrow$ Intermediate language $\Rightarrow$ MELL Proof-Net). The language needs to handle lambda-terms with explicit substitution, equivlence, reduction rules and explicit management of resources (erasure and duplication). There are several alternatives (Λ-calculus, Linear Substitution Calculus, and more), but we will focus on pn-calculus, with the following syntax:

Terms and Term contexts

$$\begin{aligned} t, u, v &::= x \mid \lambda x.t \mid tu \mid t\,[x\backslash u] \\ C &::= \diamond \mid \lambda x.C \mid Ct \mid tC \mid C\,[x\backslash t] \mid t\,[x\backslash C] \end{aligned}$$

where

- $t\,[x\backslash u]$ is an **Explicit Substitution**
- we extend alpha-conversion with: $t\,[x\backslash u] \equiv t\,[\![x\backslash y]\!]\,[y\backslash u]$

We will call a term **pure** if it contains no explicit substitution.

Contexts We want to define the reduction relation by cases on the structure of the whole program, rather than locally. This creates the need for a more refined definition of contexts. We split them into **linear** contexts, in which the hole does not appear on the position of a function argument or inside the substitution, so they will never cause copying of the inserted term, and **non-linear** contexts, that may copy a term. We also distinguish **substitution** contexts as a subset of the linear contexts.

- **Linear:** $H ::= \diamond \mid \lambda x.H \mid Ht \mid H[x\backslash t]$
- **Non-Linear:** $A ::= t\diamond \mid t[y\backslash\diamond]$
- **Substitution:** $L ::= \diamond \mid L[x\backslash t]$

2.1 pn-Calculus Reduction Rules

There are only five operational rules for the pn-reduction relation where the β-reduction rule is decomposed into five atomic actions. The rules operate at a distance to bypass the linear and non-linear contexts. Note that only **arguments** can be erased/duplicated.

$$\begin{aligned}
\text{(Fire)} \quad & L\langle\lambda x.t\rangle u \to_{dB} L\langle t[x\backslash u]\rangle \\
\text{(Erase)} \quad & t[x\backslash u] \to_{gc} t \quad \text{if } x \notin fv(t) \\
\text{(Linear Subst.)} \quad & H\langle x\rangle[x\backslash u] \to_{lsubs} H\langle u\rangle \\
\text{(Jump)} \quad & H\langle A\langle t\rangle\rangle[x\backslash u] \to_{arg} H\langle A\langle t[x\backslash u]\rangle\rangle \\
\text{(Duplicate)} \quad & H\langle A\langle t\rangle\rangle[x\backslash u] \to_{dup} H\langle A[x\backslash u]\langle t\rangle[x\backslash u]\rangle
\end{aligned}$$

3 Static and Dynamic Translation to MELL

When conducting the static translations we need think about terms, arguments and boxes as follows:

- **Linear context:** terms (outside arguments) → Proof-Nets (outside BOX)
- **Non-linear context:** terms (arguments) → Proof-Nets (inside BOX)
- Arguments of applications and substitution → BOX (can be erased/duplicated)

3.1 Explicit Substitutions - Multiplicative System

To use explicit substitutions without weakening and contraction we introduce these rules in the multiplicative system. However, we could also use the additive system, since they are equivalent. Note that if $\Gamma \vdash t : A$ then $\mathbf{fv}(t) = \mathbf{dom}(\Gamma)$.

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

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

$$\frac{\Gamma \vdash u : B \quad \Delta, x : B \vdash t : A}{\Gamma \cup \Delta \vdash t\,[x\backslash u] : A}\,\text{cut1} \qquad \frac{\Gamma \vdash u : B \quad \Delta \vdash t : A \quad x \notin \mathbf{fv}(t)}{\Gamma \cup \Delta \vdash t[x\backslash u] : A}\,\text{cut2}$$

3.2 Translation principles

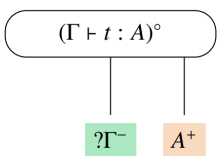
Translation of types The static translations introduces two new notations: the superscript minus and plus. The minus (i.e. A^-), means that the type resides at the left of the turnstile, whilst the plus (i.e. A^+) reside to the right. Also note the unary connectives ! (named bang) and ? denoting that a formula can be used for an unbounded number of times.

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

Remark

$$\begin{aligned} (?(A^-) \parr B^+)^\perp &= !A^+ \otimes B^- \\ (?A^-)^\perp &= !A^+ \end{aligned}$$

Translation of derivations Let $\Gamma = x_1 : B_1, ..., x_n : B_n$ then $\Gamma \vdash t : A$ translates to a MELL Proof-Net: $(\Gamma \vdash t : A)^\circ$ with interfaces $?\Gamma^-$, A^+ where $?\Gamma^-$ means $?B_1^-, ..., ?B_n^-$

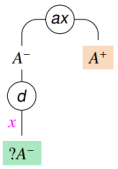


Example translation of rules Two examples translation of the rules (Section 3.1), all rule translations can be found in the slides p.29-34 at https://www.cs.uoregon.edu/research/summerschool/summer25/_lectures/Kesner_Lesson2.pdf

Original derivation:

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

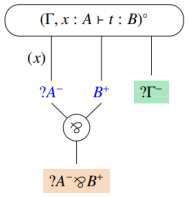
Proof-Net Translation:



Original derivation:

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

Proof-Net Translation:



3.3 Equivalence on pn-terms

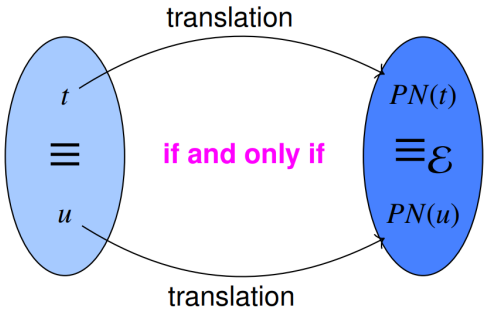
For pn-terms we use (Regnier's definition [1] of) σ-equivalence:

$$\begin{array}{rll} \lambda x.t \equiv \lambda y.t\{\{x\backslash y\}\} & & y \text{ fresh} \\ t[x\backslash u] \equiv t\{\{x\backslash y\}\}[y\backslash u] & & y \text{ fresh} \\ H\,\langle t\rangle\,[x\backslash u] \equiv H\,\langle t[x\backslash u]\rangle & & \text{if } x \notin \mathbf{fv}(H) \text{and no capture of free variables} \end{array}$$

Paticular instances

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

Relation between pn-calculus and Linear logic Proof-Net Since we have a translation from pn-calculus to MELL Proof-Nets we will have two notions of equivalence. For pn-terms we have Reigner's σ-equivalence ($\equiv$). Whilst for the Proof-Nets we have structural equivalence ($\equiv_\epsilon$). A σ-equivalence holds if and only if its corresponding structural equivalence holds given its translation.



3.4 Dynamic Translation: Term vs Proof-Net Reductions

Dynamic part of the translation consists on showing that reduction relation is preserved by the translation, i.e. that for every reduction rule relating terms in our intermediate calculus, we have a reduction rule that relates result graphs of the translation. The exact mapping of rules is summarized below.

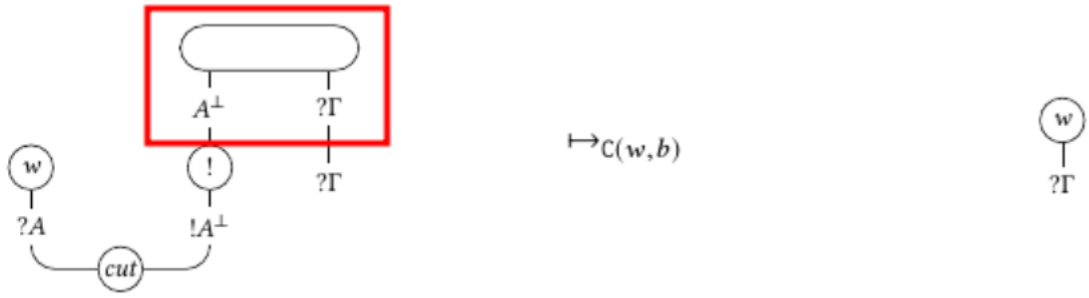
pn-term reduction	**Proof-net equivalent**
Fire a redex	Multiplicative Cut Rules
Erase	Weakening-Box
Linearly substitute	Dereliction-Box
Duplicate	Contraction-Box
Jump into argument	Box-Box

Most of the rules are mapped one-to-one. The only exception are multiplicative cut rules. This gives us a fine-grained interpretation of Intuitionistic Proof-Nets.

Erasure rules

$$t\,[x\backslash u] \mapsto_{\mathrm{gc}} t \quad \text{where } x \notin \mathbf{fv}(t)$$

maps to

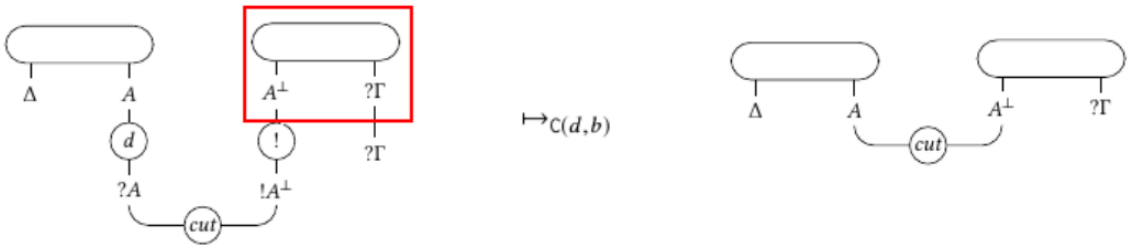


Explicit substitution for an unused variable translates to a proof-net where the wire corresponding to the variable connects to W and *cut* nodes. This leads us to a proof-net redex where we can use $C(w, b)$ rule. We can drop substitutions for variables that are not free. Likewise, if we proved something via an indirect weakening (i.e. using a proposition that we proved via weakening), we can just skip the intermediate proof and use weakening directly.

Linear substitution rule

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

maps to

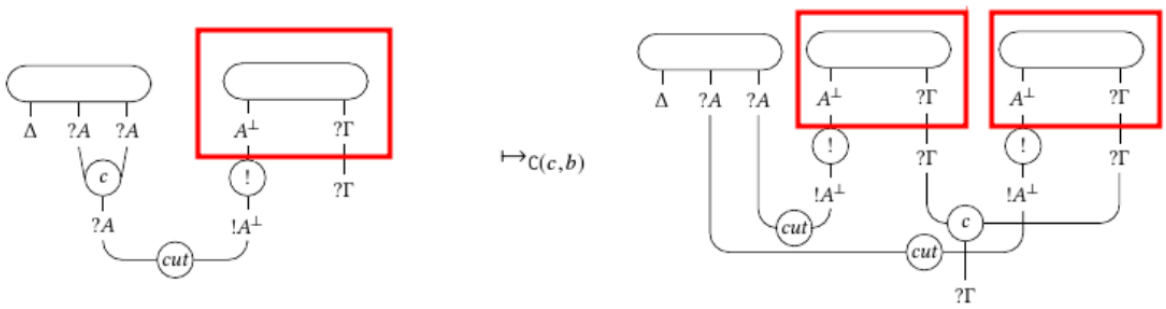


Here the variable substituted for in the translated term is used linearly. This means that its wire in the result of translation is not touched by the proof-net constructors, therefore it must start in the dereliction (D) node coming from the translation of the axiom rule. This wire is connected by translation of the *cut*1 rule typing the explicit substitution, and forms a redex with the ! node.

Duplication rules

$$H\,\langle A\,\langle t\rangle\rangle\,[x\backslash u] \mapsto_{\text{lsubs}} H\,\langle A_{[x\backslash u]}\,\langle t\,[x\backslash u]\rangle\rangle \quad \text{where } x \notin \mathbf{fv}(H), x \in \mathbf{fv}(A) \cap \mathbf{fv}(t)$$

maps to

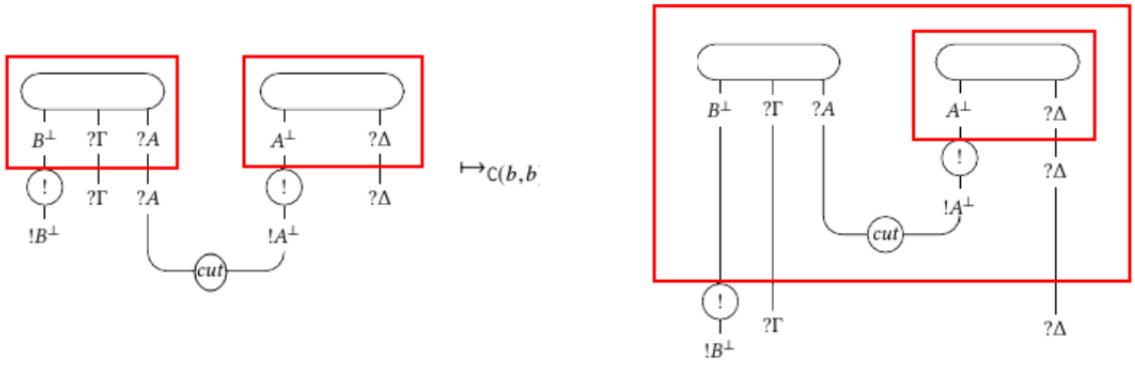


Variable x in the term on the left may be used both by t and by A, what translates to a term where the wire representing the variable connects to the copying node. Such proof-net reduces by duplicating subproofs related to the type of x and that corresponds to substituting u for x separately in A and in t.

Jump rule

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

maps to



The left-hand side of the relation on terms translates to a proof-net where wires corresponding to the outputs of t and u are boxed (i.e. connected to ! vertices). In this case $C(b, b)$ reduction can be applied and the subgraph non-free box (resulting from the translation of u) is pushed into the subgraph resulting from the translation of t.

4 Properties of the Calculus

Full Composition The calculus supports *full composition* of substitutions:

$$t[x\backslash u] \longrightarrow^* t\{\{x\backslash u\}\}$$

That is, every term with an explicit substitution reduces to the corresponding term where the substitution is completely performed.

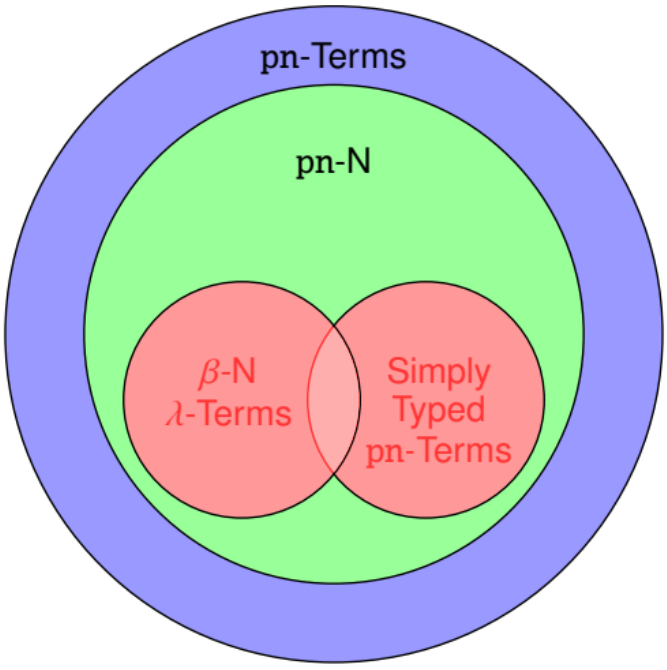
Strong Bisimulation The syntactic equivalence relation $\equiv$ on pn-terms is a *strong bisimulation* with respect to the reduction relation $\to_{pn}$:

$$t \equiv u \text{ and } t \to_{pn} t' \Rightarrow \exists u' \text{ such that } u \to_{pn} u' \text{ and } t' \equiv u'$$

Confluence The pn-calculus is *confluent* on both closed and open terms, modulo the equivalence relation $\equiv$. That is, if a term reduces to different terms, there exists a common reduct modulo $\equiv$.

Normalization Properties

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



Discussion

Designing a calculus of explicit substitutions that is both confluent on open terms and enjoys the preservation of β-normalization (PN property) was a long-standing open problem. The classical $\lambda\sigma$-calculus by Abadi-Cardelli-Curien-Lévy fails to satisfy the PN property (a result due to Melliès). Other calculi have been proposed with similar goals, but the pn-calculus stands out due to its tight and faithful correspondence with Girard's proof-nets.

5 Conclusion

The approach presented in this lecture bridges the gap between term syntaxes and graphical formalisms in the context of functional programming. It introduces a new hybrid notion of substitution that blends structural induction on the shape of terms with induction on the number of free variables to be substituted. This refined notion allows the term calculus to retain all the desirable properties one would expect from a well-behaved system. Furthermore, the calculus serves as a precise formal tool to explain the operational behavior of Girard's intuitionistic linear logic proof-nets. While other approaches are possible—such as adding explicit weakening and contraction rules to the term syntax or modifying proof-net reductions using implicit quantification—these alternatives either sacrifice the one-to-one dynamic correspondence with proof-nets or lead to more complex operational semantics. Finally, one promising direction for extension is toward classical logic, in particular through the use of polarized proof-nets.

References

[1] L. Regnier, "Une équivalence sur les lambda- termes," *Theoretical Computer Science*, vol. 126, no. 2, pp. 281–292, 1994, ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(94)90012-4. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0304397594900124.