Types and Effects: from Monads to Differential Nets

Part II: Proof Nets, Multithreading, Differential Nets

Paolo Tranquilli

paolo.tranquilli@ens-lyon.fr

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon

Séminaire LCR
LIPN, 01/03/2010

Outline

Outline

1. Previously, on Types and Effects
2. Translating into Proof Nets
   - The target
   - The translation
3. Multithreading and Differential Nets
   - First attempt: communication by differential operator
   - Second go: infinitary nondeterminism (slides only)

The context

We study Λ_{reg}, a call-by-value calculus with two basic memory access ops (`set` and `get`) and a memory allocation/deallocation op (ν).

J. M. Lucassen and D. K. Gifford.
Polymorphic effect systems.
In *POPL '88: Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 47–57, New York, NY, USA, 1988. ACM.

Roberto M. Amadio.
On stratified regions.
In Zhenjiang Hu, editor, *APLAS*, volume 5904 of *Lecture Notes in Computer Science*, pages 210–225. Springer, 2009.

An abstraction of functional programming languages with references.

The syntax of Λ_{reg}

Functions are values:

$$U, V ::= x \mid \langle\rangle \mid \lambda x.M$$

Terms can also be memory management operations:

$$M, N ::= V \mid MN \mid \texttt{set}(r, M) \mid \texttt{get}(r) \mid \nu r{\Leftarrow}M.N$$

Call-by-value order enforced via evaluation contexts:

$$E, F ::= [\,] \mid EM \mid VE \mid \texttt{set}(r, E) \mid \nu r{\Leftarrow}E.M \mid \nu r{\Leftarrow}V.E$$

Evaluation

Intuition: νr's allocate, represent and garbage collect memory.

$$E[(\lambda x.M)V] \to E[M\{V/x\}]$$

$$\left.\begin{aligned} E\big[\nu r{\Leftarrow}V.F[\mathtt{set}(r,U)]\big] &\to E\big[\nu r{\Leftarrow}U.F[\langle\rangle]\big] \\ E\big[\nu r{\Leftarrow}V.F[\mathtt{get}(r)]\big] &\to E\big[\nu r{\Leftarrow}V.F[V]\big] \end{aligned}\right\} \text{ with } r \notin \mathrm{PR}(F),$$

$$E[\nu r{\Leftarrow}V.U] \to E[U]$$

where $\mathrm{PR}(E)$ are given by what νr's bind the hole.

An example

Power function in imperative style ($M; N := (\lambda d.N)M$):

$\mathtt{function}\ \mathtt{pow}(n, m)$	$\mathtt{pow} := \lambda n, m.$
$\quad r := 1;$	$\quad \nu r {\Leftarrow} \underline{1}.$
$\quad \mathtt{for}\ i := 1\ \mathtt{to}\ m$	$\quad m$
$\qquad r := n * r;$	$\qquad (\lambda d.\, \mathtt{set}(r, \mathtt{mult}\ n\ \mathtt{get}(r)))\ \langle\rangle\ ;$
$\quad \mathtt{return}\ r;$	$\quad \mathtt{get}(r)$

$$\mathtt{pow}\ \underline{3}\ \underline{2} \to \nu r{\Leftarrow}1.\underline{2}(\lambda d.\, \mathtt{set}(r, \mathtt{mult}\ \underline{3}\ \mathtt{get}(r)))\ \langle\rangle\ ; \mathtt{get}(r)$$

$$\xrightarrow{*} \nu r{\Leftarrow}1.\ \langle\rangle\ ; \mathtt{set}(r, \mathtt{mult}\ \underline{3}\ \mathtt{get}(r)); \mathtt{set}(r, \mathtt{mult}\ \underline{3}\ \mathtt{get}(r)); \mathtt{get}(r)$$

$$\xrightarrow{*} \nu r{\Leftarrow}1.\ \mathtt{set}(r, \mathtt{mult}\ \underline{3}\ \underline{1}); \mathtt{set}(r, \mathtt{mult}\ \underline{3}\ \mathtt{get}(r)); \mathtt{get}(r)$$

$$\xrightarrow{*} \nu r{\Leftarrow}\underline{1}.\ \mathtt{set}(r, \underline{3}); \mathtt{set}(r, \mathtt{mult}\ \underline{3}\ \mathtt{get}(r)); \mathtt{get}(r)$$

$$\xrightarrow{*} \nu r{\Leftarrow}\underline{3}.\ \mathtt{set}(r, \mathtt{mult}\ \underline{3}\ \mathtt{get}(r)); \mathtt{get}(r)$$

$$\xrightarrow{*} \nu r{\Leftarrow}\underline{9}.\ \mathtt{get}(r) \to \nu r{\Leftarrow}\underline{9}.\underline{9} \to \underline{9}$$

An example

Power function in imperative style ($M; N := (\lambda d.N)M$):

$$
\begin{array}{ll}
\texttt{function pow}(n, m) & \texttt{pow} := \lambda n, m. \\
\quad r := 1; & \quad \nu r \Leftarrow \underline{1}. \\
\quad \texttt{for}\; i := 1 \;\texttt{to}\; m & \quad m \\
\quad\quad r := n * r; & \quad\quad (\lambda d.\, \texttt{set}(r, \texttt{mult}\; n\, \texttt{get}(r)))\, \langle\rangle\, ; \\
\quad \texttt{return}\; r; & \quad \texttt{get}(r)
\end{array}
$$

$$
\begin{aligned}
\texttt{pow}\,\underline{3}\,\underline{2} &\to \nu r \Leftarrow 1.\underline{2}(\lambda d.\, \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r))\, \langle\rangle\, ; \texttt{get}(r) \\
&\xrightarrow{*} \nu r \Leftarrow 1.\, \langle\rangle\, ; \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r) \\
&\xrightarrow{*} \nu r \Leftarrow 1.\, \texttt{set}(r, \texttt{mult}\,\underline{3}\,\underline{1}); \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r) \\
&\xrightarrow{*} \nu r \Leftarrow \underline{1}.\, \texttt{set}(r, \underline{3}); \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r) \\
&\xrightarrow{*} \nu r \Leftarrow \underline{3}.\, \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r) \\
&\xrightarrow{*} \nu r \Leftarrow \underline{9}.\, \texttt{get}(r) \to \nu r \Leftarrow \underline{9}.\underline{9} \to \underline{9}
\end{aligned}
$$

An example

Power function in imperative style ($M; N := (\lambda d.N)M$):

$$\begin{array}{ll} \texttt{function pow}(n,m) & \texttt{pow} := \lambda n, m. \\ \quad r := 1; & \quad \nu r \Leftarrow \underline{1}. \\ \quad \texttt{for}\, i := 1\ \texttt{to}\ m & \quad m \\ \quad\quad r := n * r; & \quad\quad (\lambda d.\,\texttt{set}(r, \texttt{mult}\, n\, \texttt{get}(r)))\, \langle\rangle\, ; \\ \quad \texttt{return}\, r; & \quad \texttt{get}(r) \end{array}$$

$$\texttt{pow}\,\underline{3}\,\underline{2} \to \nu r \Leftarrow 1.\underline{2}(\lambda d.\,\texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)))\,\langle\rangle\,;\texttt{get}(r)$$

$$\xrightarrow{*} \nu r \Leftarrow 1.\,\langle\rangle\,;\texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r));\texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r));\texttt{get}(r)$$

$$\xrightarrow{*} \nu r \Leftarrow 1.\,\texttt{set}(r, \texttt{mult}\,\underline{3}\,\underline{1});\texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r));\texttt{get}(r)$$

$$\xrightarrow{*} \nu r \Leftarrow \underline{1}.\,\texttt{set}(r, \underline{3});\texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r));\texttt{get}(r)$$

$$\xrightarrow{*} \nu r \Leftarrow \underline{3}.\,\texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r));\texttt{get}(r)$$

$$\xrightarrow{*} \nu r \Leftarrow \underline{9}.\,\texttt{get}(r) \to \nu r \Leftarrow \underline{9}.\underline{9} \to \underline{9}$$

An example

Power function in imperative style ($M; N := (\lambda d.N)M$):

$$\begin{array}{ll} \texttt{function pow}(n, m) & \texttt{pow} := \lambda n, m. \\ \quad r := 1; & \quad \nu r \Leftarrow \underline{1}. \\ \quad \texttt{for } i := 1 \texttt{ to } m & \quad m \\ \quad\quad r := n * r; & \quad\quad (\lambda d.\, \texttt{set}(r, \texttt{mult}\, n\, \texttt{get}(r)))\, \langle\rangle\, ; \\ \quad \texttt{return } r; & \quad \texttt{get}(r) \end{array}$$

$$\begin{aligned} \texttt{pow}\, \underline{3}\, \underline{2} &\to \nu r \Leftarrow 1.\underline{2}(\lambda d.\, \texttt{set}(r, \texttt{mult}\, \underline{3}\, \texttt{get}(r)))\, \langle\rangle\, ; \texttt{get}(r) \\ &\xrightarrow{*} \nu r \Leftarrow 1.\, \langle\rangle\, ; \texttt{set}(r, \texttt{mult}\, \underline{3}\, \texttt{get}(r)); \texttt{set}(r, \texttt{mult}\, \underline{3}\, \texttt{get}(r)); \texttt{get}(r) \\ &\xrightarrow{*} \nu r \Leftarrow 1.\, \texttt{set}(r, \texttt{mult}\, \underline{3}\, \underline{1}); \texttt{set}(r, \texttt{mult}\, \underline{3}\, \texttt{get}(r)); \texttt{get}(r) \\ &\xrightarrow{*} \nu r \Leftarrow \underline{1}.\, \texttt{set}(r, \underline{3}); \texttt{set}(r, \texttt{mult}\, \underline{3}\, \texttt{get}(r)); \texttt{get}(r) \\ &\xrightarrow{*} \nu r \Leftarrow \underline{3}.\, \texttt{set}(r, \texttt{mult}\, \underline{3}\, \texttt{get}(r)); \texttt{get}(r) \\ &\xrightarrow{*} \nu r \Leftarrow \underline{9}.\, \texttt{get}(r) \to \nu r \Leftarrow \underline{9}.\underline{9} \to \underline{9} \end{aligned}$$

An example

Power function in imperative style ($M; N := (\lambda d.N)M$):

$$\begin{array}{ll}
\texttt{function pow}(n, m) & \texttt{pow} := \lambda n, m. \\
\quad r := 1; & \quad \nu r \Leftarrow \underline{1}. \\
\quad \texttt{for}\, i := 1\ \texttt{to}\ m & \quad m \\
\qquad r := n * r; & \qquad (\lambda d.\,\texttt{set}(r, \texttt{mult}\, n\, \texttt{get}(r)))\,\langle\rangle\,; \\
\quad \texttt{return}\, r; & \quad \texttt{get}(r)
\end{array}$$

$$\begin{aligned}
\texttt{pow}\,\underline{3}\,\underline{2} &\to \nu r \Leftarrow 1.\underline{2}(\lambda d.\,\texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r))\,\langle\rangle\,; \texttt{get}(r) \\
&\xrightarrow{*} \nu r \Leftarrow 1.\,\langle\rangle\,; \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r) \\
&\xrightarrow{*} \nu r \Leftarrow 1.\,\texttt{set}(r, \texttt{mult}\,\underline{3}\,\underline{1}); \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r) \\
&\xrightarrow{*} \nu r \Leftarrow \underline{1}.\,\texttt{set}(r, \underline{3}); \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r) \\
&\xrightarrow{*} \nu r \Leftarrow \underline{3}.\,\texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r) \\
&\xrightarrow{*} \nu r \Leftarrow \underline{9}.\,\texttt{get}(r) \to \nu r \Leftarrow \underline{9}.\underline{9} \to \underline{9}
\end{aligned}$$

An example

Power function in imperative style ($M; N := (\lambda d.N)M$):

```
function pow(n, m)          pow := λn, m.
    r := 1;                      νr⇐1.
    for i := 1 to m              m
        r := n * r;                  (λd. set(r, mult n get(r))) ⟨⟩ ;
    return r;                    get(r)
```

$$\mathtt{pow}\,\underline{3}\,\underline{2} \to \nu r{\Leftarrow}1.\underline{2}(\lambda d.\,\mathtt{set}(r, \mathtt{mult}\,\underline{3}\,\mathtt{get}(r))\,\langle\rangle\,;\mathtt{get}(r)$$

$$\xrightarrow{*} \nu r{\Leftarrow}1.\,\langle\rangle\,;\mathtt{set}(r, \mathtt{mult}\,\underline{3}\,\mathtt{get}(r)); \mathtt{set}(r, \mathtt{mult}\,\underline{3}\,\mathtt{get}(r)); \mathtt{get}(r)$$

$$\xrightarrow{*} \nu r{\Leftarrow}1.\,\mathtt{set}(r, \mathtt{mult}\,\underline{3}\,\underline{1}); \mathtt{set}(r, \mathtt{mult}\,\underline{3}\,\mathtt{get}(r)); \mathtt{get}(r)$$

$$\xrightarrow{*} \nu r{\Leftarrow}\underline{1}.\,\mathtt{set}(r, \underline{3}); \mathtt{set}(r, \mathtt{mult}\,\underline{3}\,\mathtt{get}(r)); \mathtt{get}(r)$$

$$\xrightarrow{*} \nu r{\Leftarrow}\underline{3}.\,\mathtt{set}(r, \mathtt{mult}\,\underline{3}\,\mathtt{get}(r)); \mathtt{get}(r)$$

$$\xrightarrow{*} \nu r{\Leftarrow}\underline{9}.\,\mathtt{get}(r) \to \nu r{\Leftarrow}\underline{9}.\underline{9} \to \underline{9}$$

An example

Power function in imperative style ($M; N := (\lambda d.N)M$):

$$\begin{array}{ll}
\texttt{function pow}(n, m) & \texttt{pow} := \lambda n, m. \\
\quad r := 1; & \quad \nu r \Leftarrow \underline{1}. \\
\quad \texttt{for}\, i := 1\ \texttt{to}\ m & \quad m \\
\qquad r := n * r; & \qquad (\lambda d.\, \texttt{set}(r, \texttt{mult}\, n\, \texttt{get}(r)))\, \langle\rangle\, ; \\
\quad \texttt{return}\, r; & \quad \texttt{get}(r)
\end{array}$$

$$\begin{aligned}
\texttt{pow}\,\underline{3}\,\underline{2} &\to \nu r \Leftarrow 1.\underline{2}(\lambda d.\, \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r))\,\langle\rangle\,; \texttt{get}(r) \\
&\xrightarrow{*} \nu r \Leftarrow 1.\, \langle\rangle\,; \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r) \\
&\xrightarrow{*} \nu r \Leftarrow 1.\, \texttt{set}(r, \texttt{mult}\,\underline{3}\,\underline{1}); \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r) \\
&\xrightarrow{*} \nu r \Leftarrow \underline{1}.\, \texttt{set}(r, \underline{3}); \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r) \\
&\xrightarrow{*} \nu r \Leftarrow \underline{3}.\, \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r) \\
&\xrightarrow{*} \nu r \Leftarrow \underline{9}.\, \texttt{get}(r) \to \nu r \Leftarrow \underline{9}.\underline{9} \to \underline{9}
\end{aligned}$$

An example

Power function in imperative style ($M; N := (\lambda d.N)M$):

$\texttt{function pow}(n, m)$	$\texttt{pow} := \lambda n, m.$
$r := 1;$	$\nu r \Leftarrow \underline{1}.$
$\texttt{for}\, i := 1 \,\texttt{to}\, m$	m
$r := n * r;$	$(\lambda d.\, \texttt{set}(r, \texttt{mult}\, n\, \texttt{get}(r)))\, \langle\rangle\, ;$
$\texttt{return}\, r;$	$\texttt{get}(r)$

$$\texttt{pow}\,\underline{3}\,\underline{2} \to \nu r \Leftarrow 1.\underline{2}(\lambda d.\, \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r))\,\langle\rangle\,; \texttt{get}(r)$$

$$\xrightarrow{*} \nu r \Leftarrow 1.\, \langle\rangle\,; \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r)$$

$$\xrightarrow{*} \nu r \Leftarrow 1.\, \texttt{set}(r, \texttt{mult}\,\underline{3}\,\underline{1}); \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r)$$

$$\xrightarrow{*} \nu r \Leftarrow \underline{1}.\, \texttt{set}(r, \underline{3}); \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r)$$

$$\xrightarrow{*} \nu r \Leftarrow \underline{3}.\, \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r)$$

$$\xrightarrow{*} \nu r \Leftarrow \underline{9}.\, \texttt{get}(r) \to \nu r \Leftarrow \underline{9}.\underline{9} \to \underline{9}$$

An example

Power function in imperative style ($M; N := (\lambda d.N)M$):

$$\begin{array}{ll}
\texttt{function pow}(n,m) & \texttt{pow} := \lambda n,m. \\
\quad r := 1; & \quad \nu r \Leftarrow \underline{1}. \\
\quad \texttt{for } i := 1 \texttt{ to } m & \quad m \\
\qquad r := n * r; & \qquad (\lambda d.\, \texttt{set}(r, \texttt{mult}\, n\, \texttt{get}(r)))\, \langle\rangle\, ; \\
\quad \texttt{return } r; & \quad \texttt{get}(r)
\end{array}$$

$$\begin{aligned}
\texttt{pow}\,\underline{3}\,\underline{2} &\to \nu r \Leftarrow 1.\underline{2}(\lambda d.\,\texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r))\,\langle\rangle\,; \texttt{get}(r) \\
&\xrightarrow{*} \nu r \Leftarrow 1.\,\langle\rangle\,; \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r) \\
&\xrightarrow{*} \nu r \Leftarrow 1.\,\texttt{set}(r, \texttt{mult}\,\underline{3}\,\underline{1}); \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r) \\
&\xrightarrow{*} \nu r \Leftarrow \underline{1}.\,\texttt{set}(r, \underline{3}); \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r) \\
&\xrightarrow{*} \nu r \Leftarrow \underline{3}.\,\texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r) \\
&\xrightarrow{*} \nu r \Leftarrow \underline{9}.\,\texttt{get}(r) \to \nu r \Leftarrow \underline{9}.\underline{9} \to \underline{9}
\end{aligned}$$

An example

Power function in imperative style ($M; N := (\lambda d.N)M$):

$\texttt{function pow}(n, m)$	$\texttt{pow} := \lambda n, m.$
$r := 1;$	$\nu r \Leftarrow \underline{1}.$
$\texttt{for}\, i := 1\ \texttt{to}\ m$	m
$r := n * r;$	$(\lambda d.\,\texttt{set}(r, \texttt{mult}\, n\, \texttt{get}(r)))\,\langle\rangle\,;$
$\texttt{return}\, r;$	$\texttt{get}(r)$

$$\texttt{pow}\,\underline{3}\,\underline{2} \to \nu r \Leftarrow 1.\underline{2}(\lambda d.\,\texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r))\,\langle\rangle\,; \texttt{get}(r)$$

$$\xrightarrow{*} \nu r \Leftarrow 1.\,\langle\rangle\,; \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r)$$

$$\xrightarrow{*} \nu r \Leftarrow 1.\,\texttt{set}(r, \texttt{mult}\,\underline{3}\,\underline{1}); \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r)$$

$$\xrightarrow{*} \nu r \Leftarrow \underline{1}.\,\texttt{set}(r, \underline{3}); \texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r)$$

$$\xrightarrow{*} \nu r \Leftarrow \underline{3}.\,\texttt{set}(r, \texttt{mult}\,\underline{3}\,\texttt{get}(r)); \texttt{get}(r)$$

$$\xrightarrow{*} \nu r \Leftarrow \underline{9}.\,\texttt{get}(r) \to \nu r \Leftarrow \underline{9}.\underline{9} \to \underline{9}$$

Types and effects

- Types: $A ::= 1 \mid A \xrightarrow{e} B$, e set of accessed regions.
- $R = r_1 : A_1, \ldots, r_k : A_k$ is a region context.
- Typing judgments $R; \Gamma \vdash M : A, e$: means M accesses e.

$$\frac{}{R; \Gamma, x : A \vdash x : A, \emptyset} \qquad \frac{}{R; \Gamma \vdash \langle\rangle : 1, \emptyset}$$

$$\frac{R; \Gamma, x : A \vdash M : B, e}{R : \Gamma \vdash \lambda x.M : A \xrightarrow{e} B, \emptyset} \qquad \frac{R; \Gamma \vdash M : A \xrightarrow{e_3} B, e_1 \qquad R; \Gamma \vdash N : A, e_2}{R : \Gamma \vdash MN : B, e_1 \cup e_2 \cup e_3}$$

$$\frac{R, r : A; \Gamma \vdash M : A, e}{R, r : A; \Gamma \vdash \mathtt{set}(r, M) : 1, e \cup \{r\}} \qquad \frac{}{R, r : A; \Gamma \vdash \mathtt{get}(r) : A, \{r\}}$$

$$\frac{R, r : A; \Gamma \vdash M : A, e_1 \qquad R, r : A; \Gamma \vdash N : B, e_2}{R, r : A; \Gamma \vdash \nu r \Leftarrow M.N : B, e_1 \cup (e_2 \setminus \{r\})}$$

$$\frac{R; \Gamma \vdash M : A, e \qquad e \subsetneq f \subseteq \mathrm{dom}(R)}{R; \Gamma \vdash M : A, f}$$

Types and effects

- Types: $A ::= 1 \mid A \xrightarrow{e} B$, e set of accessed regions.
- $R = r_1 : A_1, \ldots, r_k : A_k$ is a region context.
- Typing judgments $R; \Gamma \vdash M : A, e$: means M accesses e.

$$\frac{}{R; \Gamma, x : A \vdash x : A, \emptyset} \qquad \frac{}{R; \Gamma \vdash \langle\rangle : 1, \emptyset}$$

$$\frac{R; \Gamma, x : A \vdash M : B, e}{R : \Gamma \vdash \lambda x.M : A \xrightarrow{e} B, \emptyset} \qquad \frac{R; \Gamma \vdash M : A \xrightarrow{e_3} B, e_1 \qquad R; \Gamma \vdash N : A, e_2}{R : \Gamma \vdash MN : B, e_1 \cup e_2 \cup e_3}$$

$$\frac{R, r : A; \Gamma \vdash M : A, e}{R, r : A; \Gamma \vdash \mathtt{set}(r, M) : 1, e \cup \{r\}} \qquad \frac{}{R, r : A; \Gamma \vdash \mathtt{get}(r) : A, \{r\}}$$

Effects annotate arrow type and are reset

$$\frac{R, r : A; \Gamma \vdash M : A, e_1 \qquad R, r : A; \Gamma \vdash N : B, e_2}{R, r : A; \Gamma \vdash \nu r \Leftarrow M.N : B, e_1 \cup (e_2 \setminus \{r\})}$$

$$\frac{R; \Gamma \vdash M : A, e \qquad e \subsetneq f \subseteq \mathrm{dom}(R)}{R; \Gamma \vdash M : A, f}$$

Types and effects

- Types: $A ::= 1 \mid A \xrightarrow{e} B$, e set of accessed regions.
- $R = r_1 : A_1, \ldots, r_k : A_k$ is a region context.
- Typing judgments $R; \Gamma \vdash M : A, e$: means M accesses e.

$$\frac{}{R; \Gamma, x : A \vdash x : A, \emptyset} \qquad \frac{}{R; \Gamma \vdash \langle\rangle : 1, \emptyset}$$

$$\frac{R; \Gamma, x : A \vdash M : B, e}{R : \Gamma \vdash \lambda x.M : A \xrightarrow{e} B, \emptyset} \qquad \frac{R; \Gamma \vdash M : A \xrightarrow{e_3} B, e_1 \qquad R; \Gamma \vdash N : A, e_2}{R : \Gamma \vdash MN : B, e_1 \cup e_2 \cup e_3}$$

$$\frac{R, r : A; \Gamma \vdash M : A, e}{R, r : A; \Gamma \vdash \mathtt{set}(r, M) : 1, e \cup \{r\}} \qquad \frac{}{R, r : A; \Gamma \vdash \mathtt{get}(r) : A, \{r\}}$$

$$\frac{R, r : A; \Gamma \vdash M : A, e_1 \qquad \ldots N : B, e_2}{R, r : A; \Gamma \vdash \nu r {\Leftarrow} M.N : B, e_1 \cup (e_2 \setminus \{r\})}$$

Accessed regions are noted

$$\frac{R; \Gamma \vdash M : A, e \qquad e \subsetneq f \subseteq \mathrm{dom}(R)}{R; \Gamma \vdash M : A, f}$$

Types and effects

- Types: $A ::= 1 \mid A \xrightarrow{e} B$, e set of accessed regions.
- $R = r_1 : A_1, \ldots, r_k : A_k$ is a region context.
- Typing judgments $R; \Gamma \vdash M : A, e$: means M accesses e.

$$\frac{}{R; \Gamma, x : A \vdash x : A, \emptyset} \qquad \frac{}{R; \Gamma \vdash \langle\rangle : 1, \emptyset}$$

$$\frac{R; \Gamma, x : A \vdash M : B, e}{R : \Gamma \vdash \lambda x.M : A \xrightarrow{e} B, \emptyset} \qquad \frac{R; \Gamma \vdash M : A \xrightarrow{e_3} B, e_1 \qquad R; \Gamma \vdash N : A, e_2}{R : \Gamma \vdash MN : B, e_1 \cup e_2 \cup e_3}$$

$$\frac{R, r : A; \Gamma \vdash M : A, e}{R, r : A; \Gamma \vdash \mathtt{set}(r, M) : 1, e \cup \{r\}} \qquad \frac{}{R, r : A; \Gamma \vdash \mathtt{get}(r) : A, \{r\}}$$

$$\frac{R, r : A; \Gamma \vdash M : A, e_1 \qquad R, r : A; \Gamma \vdash N : B, e_2}{R, r : A; \Gamma \vdash \nu r \Leftarrow M.N : B, e_1 \cup (e_2 \setminus \{r\})}$$

Allocations/deallocations hide effects on region

$$R; \Gamma \vdash M : A, f$$

Types and effects

- Types: $A ::= 1 \mid A \xrightarrow{e} B$, e set of accessed regions.
- $R = r_1 : A_1, \ldots, r_k : A_k$ is a region context.
- Typing judgments $R; \Gamma \vdash M : A, e$: means M accesses e.

$$\frac{}{R; \Gamma, x : A \vdash x : A, \emptyset} \qquad \frac{}{R; \Gamma \vdash \langle\rangle : 1, \emptyset}$$

$$\frac{R; \Gamma, x : A \vdash M : B, e}{R : \Gamma \vdash \lambda x.M : A \xrightarrow{e} B, \emptyset} \qquad \frac{R; \Gamma \vdash M : A \xrightarrow{e_3} B, e_1 \qquad R; \Gamma \vdash N : A, e_2}{R : \Gamma \vdash MN : B, e_1 \cup e_2 \cup e_3}$$

$$\frac{R, r : A; \Gamma \vdash M : A, e}{R, r : A; \Gamma \vdash \mathtt{set}(r, M) : 1, e \cup \{r\}} \qquad \frac{}{R, r : A; \Gamma \vdash \mathtt{get}(r) : A, \{r\}}$$

$$R, r : A; \Gamma \vdash M : A, e_1 \qquad R, r : A; \Gamma \vdash N : B, e_2$$

$R, r :$ Dummy effects can be added $\{r\})$

$$\frac{R; \Gamma \vdash M : A, e \qquad e \subsetneq f \subseteq \mathrm{dom}(R)}{R; \Gamma \vdash M : A, f}$$

Stratification

- Types and effects assure type and memory safety, but not termination.
- Typed fixpoints! In particular endless loop:

$$r : 1 \xrightarrow{\{r\}} A; \vdash \nu r \Leftarrow \lambda x.\, \mathtt{get}(r)x.\, \mathtt{get}(r)\, \langle\rangle : 1, \emptyset$$

$$\nu r \Leftarrow \lambda x.\, \mathtt{get}(r)x.\, \mathtt{get}(r)\, \langle\rangle \to \nu r \Leftarrow \lambda x.\, \mathtt{get}(r)x.(\lambda x.\, \mathtt{get}(r)x)\, \langle\rangle$$
$$\to \nu r \Leftarrow \lambda x.\, \mathtt{get}(r)x.\, \mathtt{get}(r)\, \langle\rangle \to \cdots$$

- Boudol/Amadio's proposal to avoid self-reference and ensure normalization: stratification of the region context ($R \vdash$).

$$\frac{}{\emptyset \vdash} \qquad \frac{R \vdash A \qquad r \notin \mathrm{dom}(R)}{R, r : A \vdash}$$

$$\frac{R \vdash}{R \vdash 1} \qquad \frac{R \vdash A \qquad R \vdash B \qquad e \subseteq \mathrm{dom}(R)}{R \vdash A \xrightarrow{e} B}$$

The localized monadic translation

Translation M° from typed programs to resursively typed λ-terms with pairs, via localized state monads $T_e A = \prod_{r \in e} X_r \to (\prod_{r \in e} X_r \times A)$.

Theorem

M evaluates to V iff M° evaluates to V°.

Region contexts are translated into systems of equations R°, by $r : A \mapsto X_r = A^\circ$.

Theorem

R is stratified iff R° is solvable (i.e. M° simply typed!).

I.e. absence of stratification is equivalent to actually needing recursive types.

Corollary (reproved)

If R is stratified, a typed program always terminates.

The localized monadic translation

Translation M° from typed programs to resursively typed λ-terms with pairs, via localized state monads $T_e A = \prod_{r\in e} X_r \to (\prod_{r \in e} X_r \times A)$.

Theorem

M evaluates to V iff M° evaluates to V°.

Region contexts are translated into systems of equations R°, by $r : A \mapsto X_r = A^\circ$.

Theorem

R is stratified iff R° is solvable (i.e. M° simply typed!).

I.e. absence of stratification is equivalent to actually needing recursive types.

Corollary (reproved)

If R is stratified, a typed program always terminates.

The localized monadic translation

Translation M° from typed programs to resursively typed λ-terms with pairs, via localized state monads $T_e A = \prod_{r \in e} X_r \to (\prod_{r \in e} X_r \times A)$.

Theorem

M evaluates to V iff M° evaluates to V°.

Region contexts are translated into systems of equations R°, by $r : A \mapsto X_r = A^\circ$.

Theorem

R is stratified iff R° is solvable (i.e. M° simply typed!).

I.e. absence of stratification is equivalent to actually needing recursive types.

Corollary (reproved)

If R is stratified, a typed program always terminates.

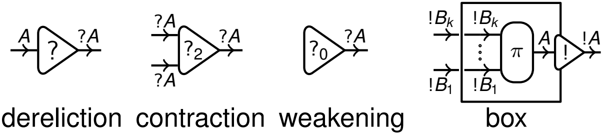
Outline

1. Previously, on Types and Effects
2. Translating into Proof Nets
   - The target
   - The translation
3. Multithreading and Differential Nets
   - First attempt: communication by differential operator
   - Second go: infinitary nondeterminism (slides only)

The target

- Proof nets are the parallel representation of linear logic proofs.
- Types: $X \mid X^{\perp} \mid 1 \mid \perp \mid A \otimes B \mid A \parr B \mid !A \mid ?A$ with duality $A^{\perp}$, linear arrow $A \multimap B = A^{\perp} \parr B$, systems of equations $X_i \doteq A_i$.

- Cells:



- Proof nets formed matching wires and enforcing a correctness criterion.

The target

- Proof nets are the parallel representation of linear logic proofs.
- Types: $X \mid X^\perp \mid 1 \mid \perp \mid A \otimes B \mid A \parr B \mid\, !A \mid\, ?A$ with duality $A^\perp$, linear arrow $A \multimap B = A^\perp \parr B$, systems of equations $X_i \doteq A_i$.

one tensor bottom par

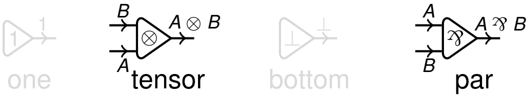
- Cells:

dereliction contraction weakening box

- Proof nets formed matching wires and enforcing a correctness criterion.

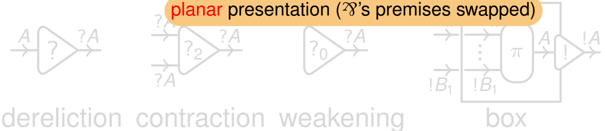
The target

- Proof nets are the parallel representation of linear logic proofs.
- Types: $X \mid X^{\perp} \mid 1 \mid \perp \mid A \otimes B \mid A ⅋ B \mid !A \mid ?A$ with duality $A^{\perp}$, linear arrow $A \multimap B = A^{\perp} ⅋ B$, systems of equations $X_i \doteq A_i$.



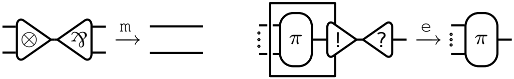
planar presentation (⅋'s premises swapped)

- Cells:

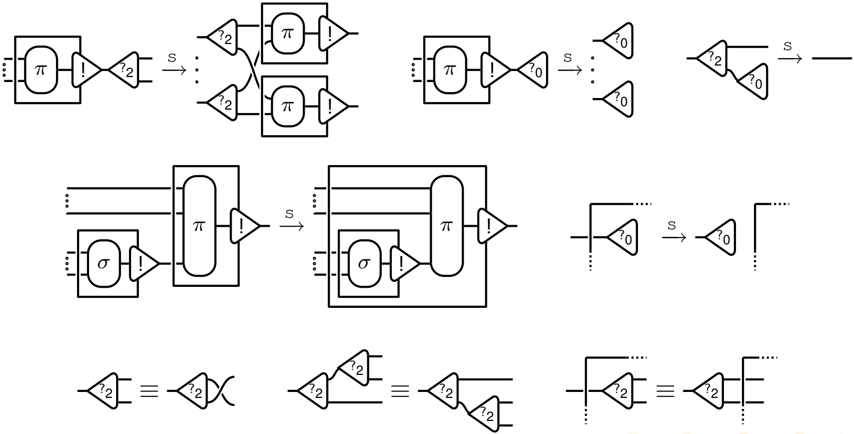


- Proof nets formed matching wires and enforcing a correctness criterion.

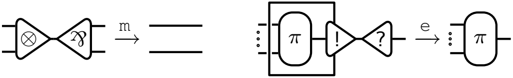
Surface reduction



at depth 0

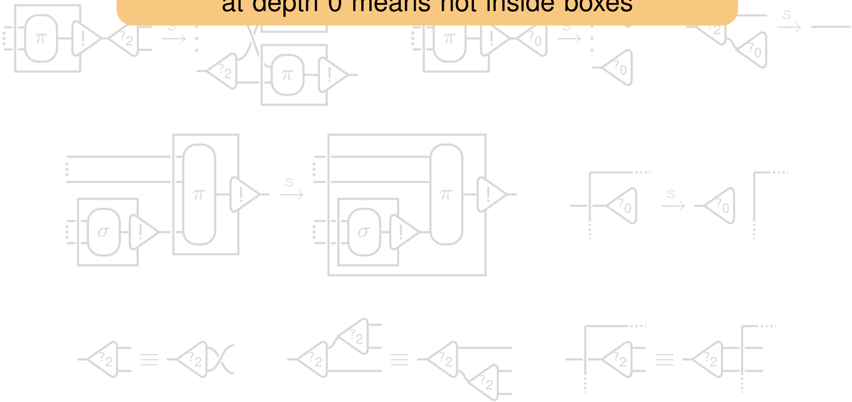


Surface reduction

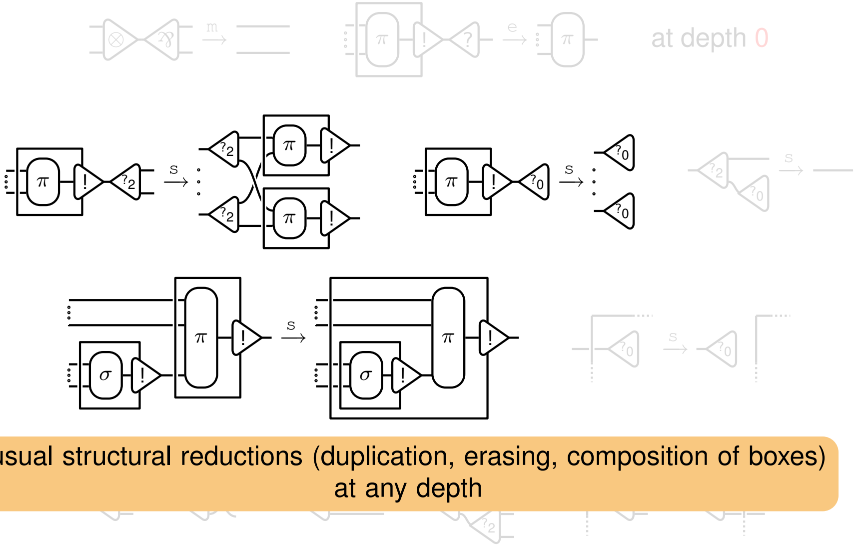


at depth 0

logical reductions (multiplicative and exponential)
at depth 0 means not inside boxes

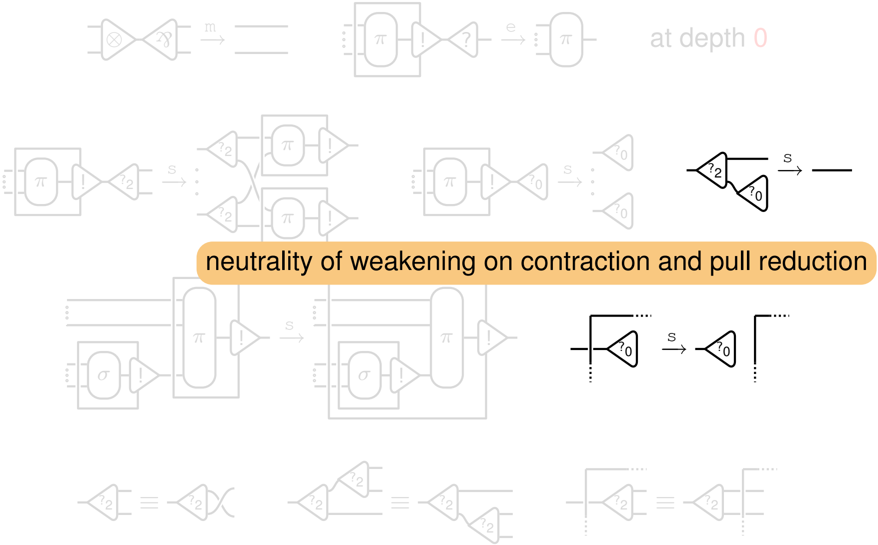


Surface reduction

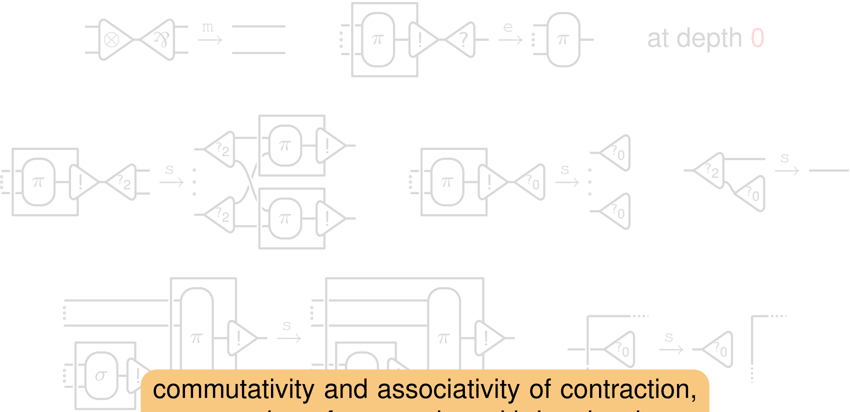


usual structural reductions (duplication, erasing, composition of boxes) at any depth

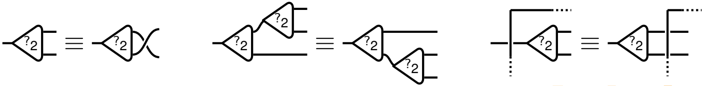
Surface reduction



Surface reduction



commutativity and associativity of contraction,
commuting of contraction with box borders



The results

We present a translation $M^\bullet$ from typed Λ_{reg} programs M to (resursively) typed proof nets.

Theorem

If $M \to N$ *then* $M^\bullet \xrightarrow{\mathtt{e}}\xrightarrow{\mathtt{m*}}\xrightarrow{\mathtt{s*}} N^\bullet$.

Theorem

$M^\bullet$ *normalizes by surface reduction to* π *iff* $\pi = V^\bullet$ *and* $M \xrightarrow{*} V$.

Notice that surface reduction has no fixed sequential strategy.

The results

We present a translation $M^\bullet$ from typed Λ_{reg} programs M to (resursively) typed proof nets.

Theorem

If $M \to N$ *then* $M^\bullet \xrightarrow{\mathtt{e}}\xrightarrow{\mathtt{m*}}\xrightarrow{\mathtt{s*}} N^\bullet$.

Theorem

$M^\bullet$ *normalizes by surface reduction to* π *iff* $\pi = V^\bullet$ *and* $M \xrightarrow{*} V$.

Notice that surface reduction has no fixed sequential strategy.

Call-by-value translation

- Regular λ-calculus has two translations into linear logic, allowing its parallel evaluation.
- They are based on the two Girard's translations of intuitionistic logic:

$$(A \to B)^{\blacktriangle} = !A^{\blacktriangle} \multimap B^{\blacktriangle}, \qquad (A \to B)^{\bullet} = !(A^{\bullet} \multimap B^{\bullet})$$

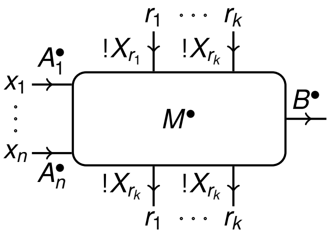
- In fact, the former corresponds to call-by-name (arguments are duplicable), the latter to call-by-value (functions are duplicable).

J. Maraist, M. Odersky, D. N. Turner, and P. Wadler.
Call-by-name, call-by-value, call-by-need and the linear lambda calculus.
Theor. Comput. Sci., 228(1-2):175–210, 1999.

- We will therefore extend the call-by-value translation.

General form of the translation

- $R; x_1 : A_1, \ldots, x_n : A_n \vdash M : B, \{r_1, \ldots, r_k\}$ gets translated to a net

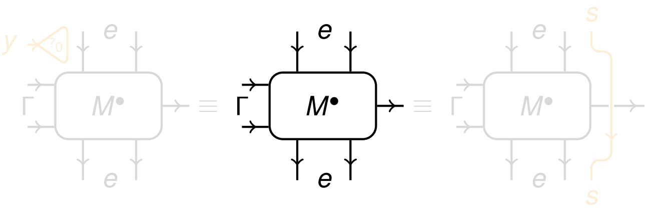


(we will show the translation of types and effects later)

- It is useful to visualize programs as processing streams of regions going top to bottom.

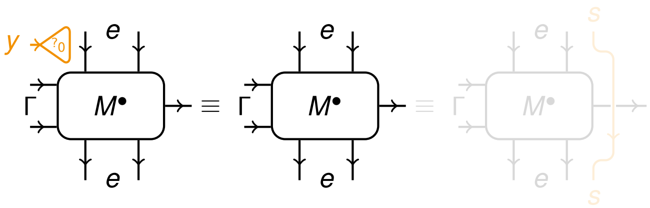
Dummy variables and dummy effects

We consider translations up to dummy variables and dummy effects.



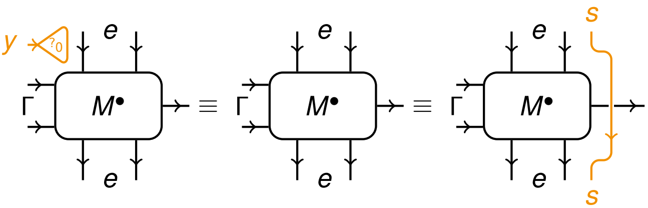
Dummy variables and dummy effects

We consider translations up to dummy variables and dummy effects.



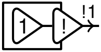
Dummy variables and dummy effects

We consider translations up to dummy variables and dummy effects.



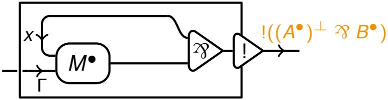
The translation: variable and unit

$x^\bullet =$ 

$\langle\rangle^\bullet =$ 

Types: $1^\bullet = !1$.

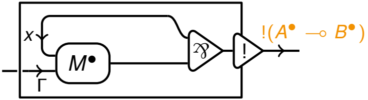
The translation: abstraction

$(\lambda x.M)^{\bullet} =$ 

Usual call-by-value translation extended by encapsulating the effects.

Types: $e^{\bullet} = \bigotimes_{r \in e} !X_r, \quad (A \xrightarrow{e} B)^{\bullet} = !(A^{\bullet} \multimap e^{\bullet} \multimap (e^{\bullet} \otimes B^{\bullet}))$.

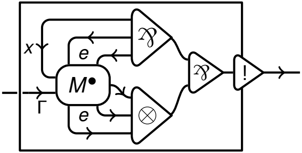
The translation: abstraction

$(\lambda x.M)^\bullet =$ 

Usual call-by-value translation extended by encapsulating the effects.

Types: $e^\bullet = \bigotimes_{r \in e} !X_r, \quad (A \xrightarrow{e} B)^\bullet = !(A^\bullet \multimap e^\bullet \multimap (e^\bullet \otimes B^\bullet)).$

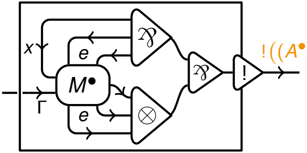
The translation: abstraction

$(\lambda x.M)^\bullet =$ 

Usual call-by-value translation extended by encapsulating the effects.

Types: $e^\bullet = \bigotimes_{r \in e} !X_r, \quad (A \xrightarrow{e} B)^\bullet = !(A^\bullet \multimap e^\bullet \multimap (e^\bullet \otimes B^\bullet)).$

The translation: abstraction

$(\lambda x.M)^\bullet =$ 

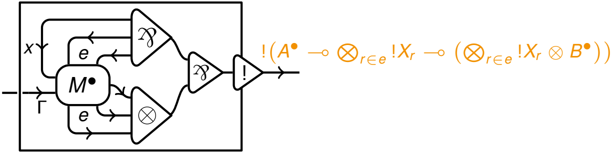
$!((A^\bullet)^\perp \parr \parr_{r\in e} ?X_r^\perp \parr (\bigotimes_{r\in e} !X_r \otimes B^\bullet))$

Usual call-by-value translation extended by encapsulating the effects.

Types: $e^\bullet = \bigotimes_{r\in e} !X_r, \quad (A \xrightarrow{e} B)^\bullet = !(A^\bullet \multimap e^\bullet \multimap (e^\bullet \otimes B^\bullet)).$

The translation: abstraction

$(\lambda x.M)^\bullet =$

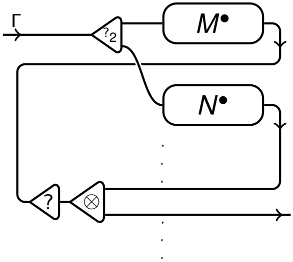


Usual call-by-value translation extended by encapsulating the effects.

Types: $e^\bullet = \bigotimes_{r\in e} !X_r, \quad (A \xrightarrow{e} B)^\bullet = !(A^\bullet \multimap e^\bullet \multimap (e^\bullet \otimes B^\bullet)).$

The translation: application

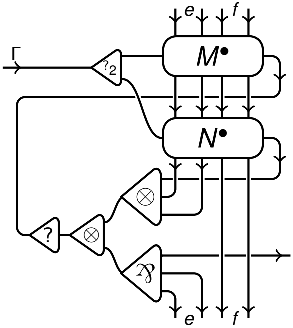
Suppose $M : A \to B, \emptyset$ and $N : A, \emptyset$.

$(MN)^\bullet =$ 

Usual translation extended by extracting effects and linking in evaluation order.

The translation: application

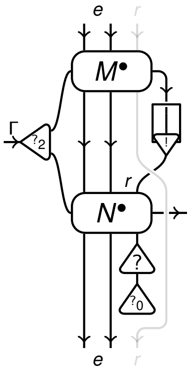
Suppose $M : A \xrightarrow{e} B, e + f$ and $N : A, e + f$.

$(MN)^\bullet =$ 

Usual translation extended by extracting effects and linking in evaluation order.

The translation of memory operations: $\nu r \Leftarrow M.N$, $\texttt{set}(r, M)$, $\texttt{get}(r)$.

$(\nu r \Leftarrow M.N)^\bullet =$

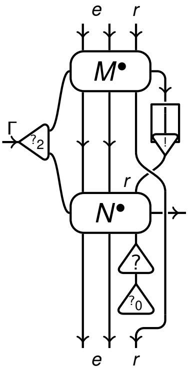


$(\texttt{set}(r, M))^\bullet =$

$(\texttt{get}(r))^\bullet =$

The translation of memory operations: $\nu r \Leftarrow M.N$, $\texttt{set}(r, M)$, $\texttt{get}(r)$.

$(\nu r \Leftarrow M.N)^\bullet =$

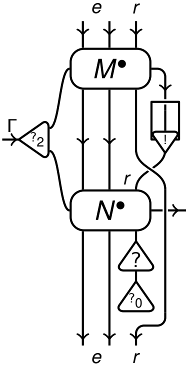


$(\texttt{set}(r, M))^\bullet =$

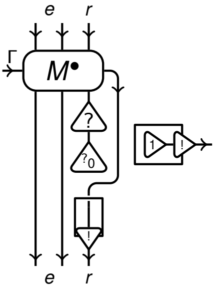
$(\texttt{get}(r))^\bullet =$

The translation of memory operations: $\nu r{\Leftarrow}M.N$, $\mathtt{set}(r, M)$, $\mathtt{get}(r)$.

$(\nu r{\Leftarrow}M.N)^\bullet =$



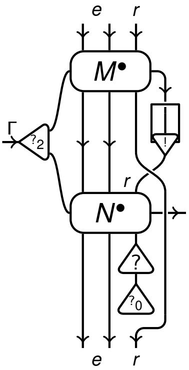
$(\mathtt{set}(r, M))^\bullet =$



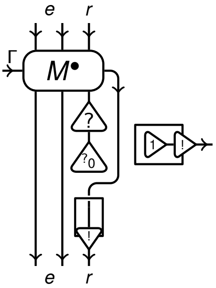
$(\mathtt{get}(r))^\bullet =$

The translation of memory operations: $\nu r \Leftarrow M.N$, $\mathtt{set}(r, M)$, $\mathtt{get}(r)$.

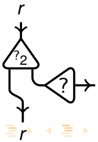
$(\nu r \Leftarrow M.N)^\bullet =$



$(\mathtt{set}(r, M))^\bullet =$



$(\mathtt{get}(r))^\bullet =$



The translation: summing up

- Sets of regions: $e^\bullet = \bigotimes_{r\in e} !X_r$.
- Types: $1^\bullet = !1 \qquad (A \xrightarrow{e} B)^\bullet = !(A^\bullet \multimap e^\bullet \multimap (e^\bullet \otimes B^\bullet))$
  (we consider $(A \xrightarrow{\emptyset} B)^\bullet = !(A^\bullet \multimap B^\bullet))$
- Region contexts: $(r_1 : A_1, \ldots, r_k : A_k)^\bullet = (X_{r_1} \doteq A_1^\bullet, \ldots, X_{r_k} \doteq A_k^\bullet)$.

Theorem

R is stratified iff $R^\bullet$ is solvable (i.e. $M^\bullet$ simply typed!).

Theorem

If $M \to N$ then $M^\bullet \xrightarrow{\mathtt{e}}\xrightarrow{\mathtt{m*}}\xrightarrow{\mathtt{s*}} N^\bullet$.

Theorem

$M^\bullet$ normalizes by surface reduction to π iff $\pi = V^\bullet$ and $M \xrightarrow{*} V$.

Proof nets as parallel evaluators

- Proof nets instantiate as connections the dependencies described by effects.
- E.g. $M : A, \{s\}$, $N : B, \{r\}$, and $\texttt{set}(r, V); M; N$. After unfolding the seq. composition...
- N can be safely evaluated before or at the same time of M.
- The third result

Theorem

$M^\bullet$ normalizes by surface reduction to π iff $\pi = V^\bullet$ and $M \xrightarrow{*} V$.

ensures sequential semantics is preserved.

Proof nets as parallel evaluators

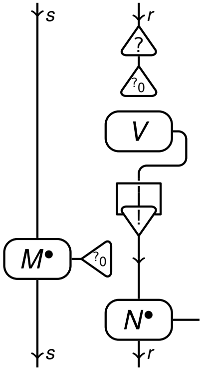
- Proof nets instantiate as connections the dependencies described by effects.
- E.g. $M : A, \{s\}$, $N : B, \{r\}$, and $\mathtt{set}(r, V); M; N$. After unfolding the seq. composition...
- N can be safely evaluated before or at the same time of M.
- The third result

Theorem

$M^\bullet$ normalizes by surface reduction to π iff $\pi = V^\bullet$ and $M \xrightarrow{*} V$.

ensures sequential semantics is preserved.

$$(\mathtt{set}(r, V); M; N)^\bullet$$
$$\downarrow^*$$



Proof nets as parallel evaluators

- Proof nets instantiate as connections the dependencies described by effects.
- E.g. $M : A, \{s\}$, $N : B, \{r\}$, and $\texttt{set}(r, V); M; N$. After unfolding the seq. composition...
- N can be safely evaluated before or at the same time of M.
- The third result

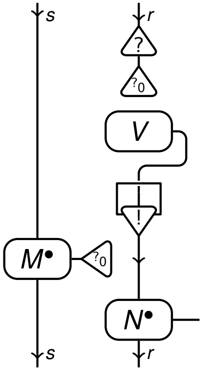
Theorem

$M^\bullet$ normalizes by surface reduction to π iff $\pi = V^\bullet$ and $M \xrightarrow{*} V$.

ensures sequential semantics is preserved.

$(\texttt{set}(r, V); M; N)^\bullet$

$\downarrow^*$



Proof nets as parallel evaluators

- Proof nets instantiate as connections the dependencies described by effects.
- E.g. $M : A, \{s\}$, $N : B, \{r\}$, and $\texttt{set}(r, V); M; N$. After unfolding the seq. composition...
- N can be safely evaluated before or at the same time of M.
- The third result

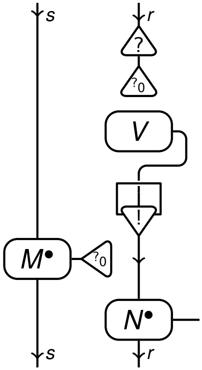
Theorem

$M^\bullet$ normalizes by surface reduction to π iff $\pi = V^\bullet$ and $M \xrightarrow{*} V$.

ensures sequential semantics is preserved.

$(\texttt{set}(r, V); M; N)^\bullet$

$\downarrow^*$



Outline

Outline

1. Previously, on Types and Effects
2. Translating into Proof Nets
   - The target
   - The translation
3. Multithreading and Differential Nets
   - First attempt: communication by differential operator
   - Second go: infinitary nondeterminism (slides only)

Work in progress. . .

Multithreading

- Parallel threads cooperating via references.
- Terms: $\ldots \mid (M|N)$ (and values: $\ldots \mid (U|V)$).
- Evaluation contexts: $\ldots \mid (E|M) \mid (M|E)$.
- Maximal evaluation context not unique anymore $\rightsquigarrow$ concurrency:

$$\nu r \Leftarrow \mathtt{true}.(\mathtt{get}(r) | \mathtt{set}(r, \mathtt{not}\ \mathtt{get}(r)))$$

$$\swarrow^{*} \qquad \searrow^{*}$$

$$\mathtt{true} | \langle\rangle \qquad\qquad \mathtt{false} | \langle\rangle$$

- Thread control operations possible but left aside (e.g. joining, or "worker" threads).

Multithreading

- Parallel threads cooperating via references.
- Terms: $\ldots \mid (M|N)$ (and values: $\ldots \mid (U|V)$).
- Evaluation contexts: $\ldots \mid (E|M) \mid (M|E)$.
- Maximal evaluation context not unique anymore $\rightsquigarrow$ concurrency:

$$\nu r \Leftarrow \texttt{true}.\big(\texttt{get}(r) \,|\, \texttt{set}(r, \texttt{not}\ \texttt{get}(r))\big)$$

$$\swarrow^{*} \qquad\qquad \searrow^{*}$$

$$\texttt{true} \,|\, \langle\rangle \qquad\qquad \texttt{false} \,|\, \langle\rangle$$

- Thread control operations possible but left aside (e.g. joining, or “worker” threads).

Multithreading

- Parallel threads cooperating via references.
- Terms: $\ldots \mid (M|N)$ (and values: $\ldots \mid (U|V)$).
- Evaluation contexts: $\ldots \mid (E|M) \mid (M|E)$.
- Maximal evaluation context not unique anymore $\rightsquigarrow$ concurrency:

$$\nu r \Leftarrow \texttt{true}.\big(\texttt{get}(r) \,|\, \texttt{set}(r, \texttt{not}\;\texttt{get}(r))\big)$$

$$\swarrow^{*} \qquad\qquad \searrow^{*}$$

$$\texttt{true} \,|\, \langle\rangle \qquad\qquad \texttt{false} \,|\, \langle\rangle$$

- Thread control operations possible but left aside (e.g. joining, or “worker” threads).

Types for multithreading

- In types, one introduces a "thread behaviour":
  - Types: $\ldots \mid A \xrightarrow{e} \mathbb{B}$;
  - $\mathbb{B}$ is the behaviour of parallel threads, of any type.
- $A \xrightarrow{e} \mathbb{B} \rightsquigarrow$ threads cannot be arguments directly.
- Example : $(\mathtt{Nat}_A = (A \to A) \to A \to A)$

$$\mathtt{npar} := \lambda n, p.\, n\,(\lambda f, d.\, f\langle\rangle | p\langle\rangle)\, p\,\langle\rangle : \mathtt{Nat}_{1 \xrightarrow{e} \mathbb{B}} \to (1 \xrightarrow{e} \mathbb{B}) \xrightarrow{e} \mathbb{B}$$

$$\mathtt{npar}\,\underline{n}\,(\lambda d.M) \xrightarrow{*} \underbrace{M | \cdots | M}_{n+1}$$

Types for multithreading

- In types, one introduces a “thread behaviour”:
  - Types: $\ldots \mid A \xrightarrow{e} \mathbb{B}$;
  - $\mathbb{B}$ is the behaviour of parallel threads, of any type.
- $A \xrightarrow{e} \mathbb{B} \rightsquigarrow$ threads cannot be arguments directly.
- Example : $(\mathtt{Nat}_A = (A \to A) \to A \to A)$

$$\mathtt{npar} := \lambda n, p.\, n\,(\lambda f, d.\, f\langle\rangle | p\langle\rangle)\, p\,\langle\rangle : \mathtt{Nat}_{1 \xrightarrow{e} \mathbb{B}} \to (1 \xrightarrow{e} \mathbb{B}) \xrightarrow{e} \mathbb{B}$$

$$\mathtt{npar}\,\underline{n}\,(\lambda d.M) \xrightarrow{*} \underbrace{M | \cdots | M}_{n+1}$$

Types for multithreading

- In types, one introduces a "thread behaviour":
  - Types: $\ldots \mid A \xrightarrow{e} \mathbb{B}$;
  - $\mathbb{B}$ is the behaviour of parallel threads, of any type.
- $A \xrightarrow{e} \mathbb{B} \rightsquigarrow$ threads cannot be arguments directly.
- Example : $(\mathtt{Nat}_A = (A \to A) \to A \to A)$

$$\mathtt{npar} := \lambda n, p.\, n\,(\lambda f, d.\, f\langle\rangle | p\langle\rangle)\, p\,\langle\rangle : \mathtt{Nat}_{1 \xrightarrow{e} \mathbb{B}} \to (1 \xrightarrow{e} \mathbb{B}) \xrightarrow{e} \mathbb{B}$$

$$\mathtt{npar}\, \underline{n}\,(\lambda d.M) \xrightarrow{*} \underbrace{M | \cdots | M}_{n+1}$$

The base idea

- Parallel threads live in a “communication soup”.
- The sequentiality of each thread is similar to prefixing.
- Proof nets are parallel but deterministic, i.e. not suitable for concurrency…
- … but nowadays we have differential nets!

Thomas Ehrhard and Laurent Regnier.
Differential interaction nets.
Theor. Comput. Sci., 364(2):166–195, 2006.

The base idea

- Parallel threads live in a “communication soup”.
- The sequentiality of each thread is similar to prefixing.
- Proof nets are parallel but deterministic, i.e. not suitable for concurrency. . .
- . . . but nowadays we have differential nets!

Thomas Ehrhard and Laurent Regnier.
Differential interaction nets.
Theor. Comput. Sci., 364(2):166–195, 2006.

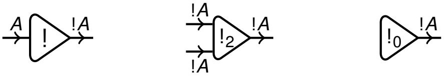
The base idea

- Parallel threads live in a "communication soup".
- The sequentiality of each thread is similar to prefixing.
- Proof nets are parallel but deterministic, i.e. not suitable for concurrency. . .
- . . . but nowadays we have differential nets!

Thomas Ehrhard and Laurent Regnier.
Differential interaction nets.
Theor. Comput. Sci., 364(2):166–195, 2006.

The target: differential nets

- Extension of proofnets with one-use resources/differential operator.
- New cells:

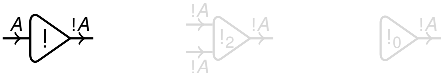


codereliction cocontraction coweakening

- We will use two specific instances of second order: $\forall X.(X \multimap X)$ (for "transistors") and $\exists X.X$ (for $\mathbb{B}$).

The target: differential nets

- Extension of proofnets with one-use resources/differential operator.
- New cells:



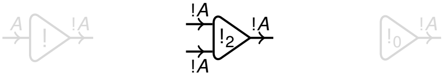
codereliction   cocontraction   coweakening

One-use resource, asked many times, used excalty once.

Differential operator $\frac{\partial f}{\partial x}\Big|_{x=0}$.

The target: differential nets

- Extension of proofnets with one-use resources/differential operator.
- New cells:



codereliction cocontraction coweakening

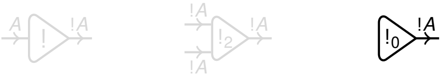
- We will use two … order: $\forall X.(X \multimap X)$
  (for "transistors" …

Joining of resources.

Evaluation in a sum $x + y$.

The target: differential nets

- Extension of proofnets with one-use resources/differential operator.
- New cells:



codereliction cocontraction coweakening

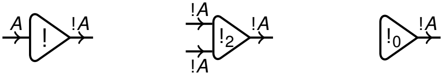
- We will use two speci[...]cond order: $\forall X.(X \multimap X)$ (for "transistors") and [...]

Empty resource.

Evaluation in 0.

The target: differential nets

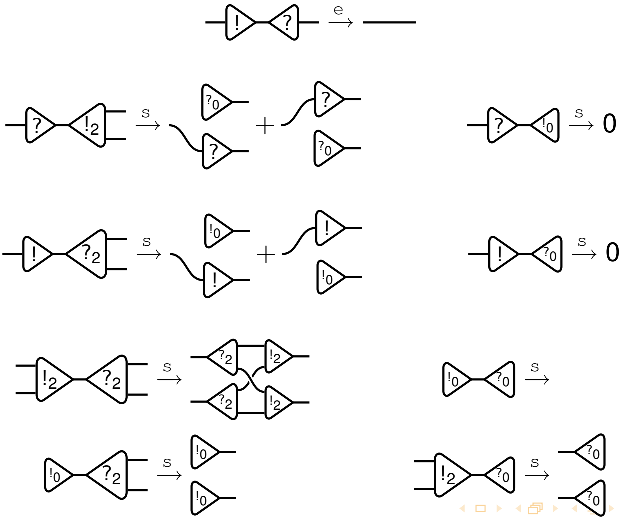
- Extension of proofnets with one-use resources/differential operator.
- New cells:



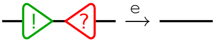
codereliction cocontraction coweakening

- We will use two specific instances of second order: $\forall X.(X \multimap X)$ (for "transistors") and $\exists X.X$ (for $\mathbb{B}$).

New reductions

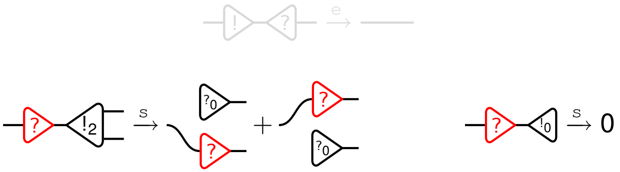


New reductions



A query meets a one-use resource and is answered

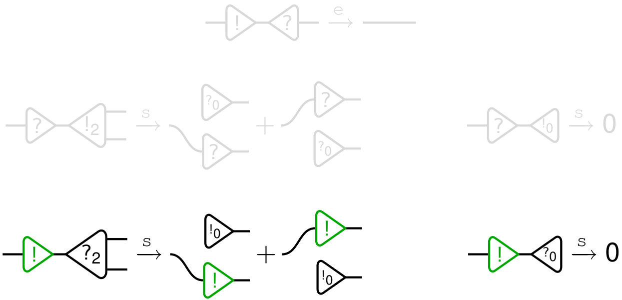
New reductions



A query chooses between two sets of resources. . .

. . . or fails facing no resource (starvation)

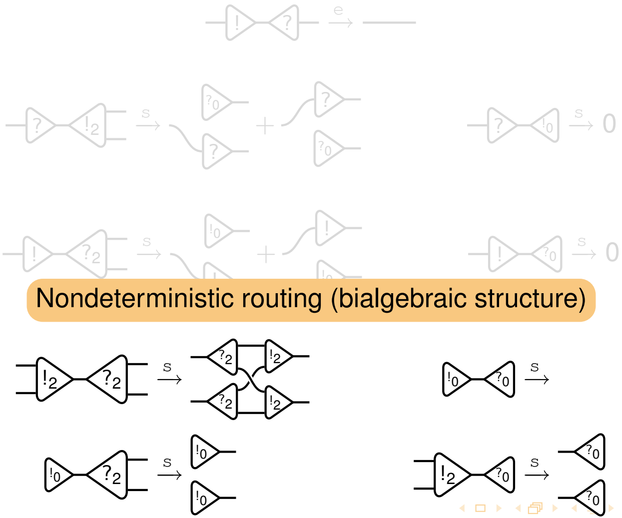
New reductions



A one-use resource is asked by more queries and goes to either one. . .

. . . or is not asked and gives a failure (linearity!)

New reductions



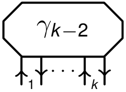
Sums and boxes

- So reduction introduces sums, representing different nondeterministic internal choices.
- In the nets we will consider:
  - no sum will appear inside boxes;
  - no cocontraction, coweakening or codereliction on auxiliary port will appear (a relief!).

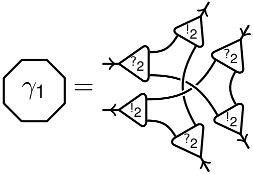
Differential nets and π-calculus

Thomas Ehrhard and Olivier Laurent.
Interpreting a finitary pi-calculus in differential interaction nets.
In *CONCUR*, volume 4703 of *LNCS*, pages 333–348. Springer, 2007.

- Translation of a finitary fragment of π-calculus in differential nets.
- One of the basic structures: communication zones

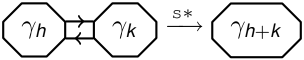


- E.g.:

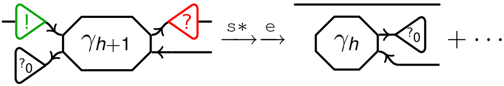


Properties of communication zones

They fuse:

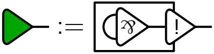


They allow queries and resources to communicate:

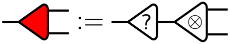


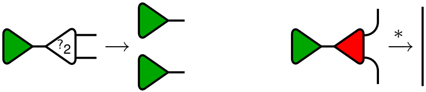
Signals, transistors, broadcast

- Signal:

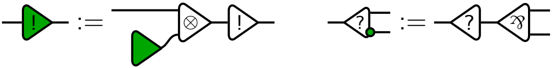


- Transistor:



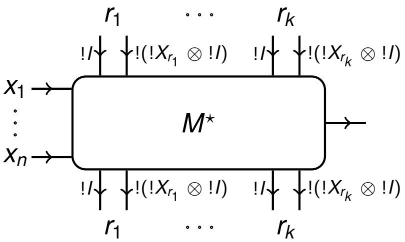


- Broadcast and reception:



General form of the translation

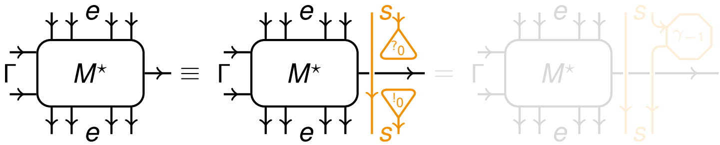
- Let $I = \forall\alpha(\alpha \multimap \alpha)$:



- There are two channels for each region:
  - One transports the actual data, on a "first come first served" basis; data travels with a signal, to be released when hold on data is achieved;
  - The other passes the signal enforcing sequentiality of each thread, on a per region basis.

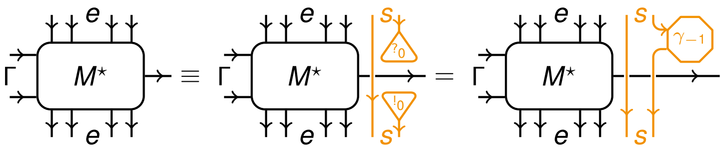
Dummy effects

In adding dummy effects signal passes through, data is cut off:



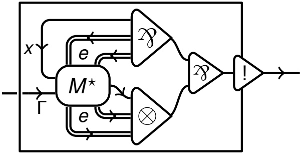
Dummy effects

In adding dummy effects signal passes through, data is cut off:



On to the new translation: variable, unit, abstraction

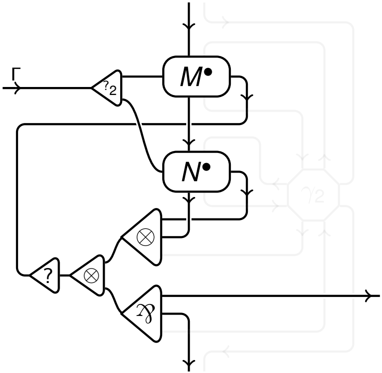
- Variable (axiom) and unit (boxed 1) remain the same.
- Abstraction too, signal is encapsulated along data:

$(\lambda x.M)^\star =$ 

The new translation: application

For simplicity, suppose $M : A \xrightarrow{\{r\}} B, \{r\}$ and $N : A, \{r\}$.
We adapt the previous translation...

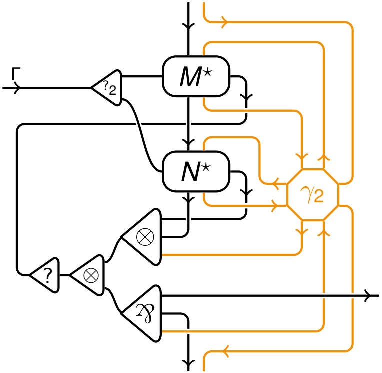
$(MN)^\bullet =$



... by passing signal only and leaving data to communication zones.

The new translation: application

For simplicity, suppose $M : A \xrightarrow{\{r\}} B, \{r\}$ and $N : A, \{r\}$.
We adapt the previous translation...

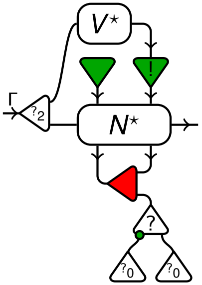
$(MN)^\star =$ 

... by passing signal only and leaving data to communication zones.

The new translation: νr

1. $\nu r \Leftarrow V.N$ broadcasts V to N and sends it a signal;
2. waits for N to give signal back which activates the garbage collection.

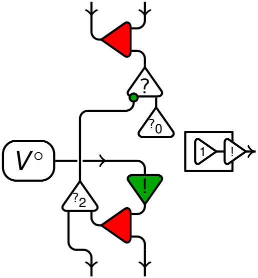
$(\nu r \Leftarrow V.N)^\star =$



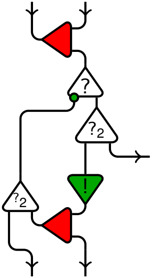
The new translation: set and get

1. Memory ops wait for signal to unlock,
2. then wait for exclusive access to data,
3. then release a signal and broadcast data back.

$(\mathtt{set}(r, V))^\star =$



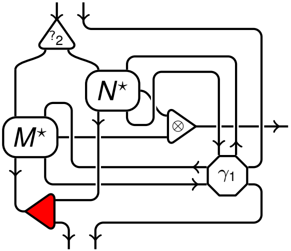
$(\mathtt{get}(r))^\star =$



The translation of parallel composition

1. A received signal is sent to both terms at the same time, while data is handled by a communication zone;
2. will send the signal when both terms have (implementation not completely symmetric).

$(M|N)^\star =$



A bit of discussion

Theorem

$M \to N \implies M^\star \xrightarrow{+} N^\star + \cdots.$

- The bisimulation result is yet to be precised and proved: probably based on some notion of observable reduction.
- Unlike π-calculus and its translation, the prefixing here is selective: only operations on the same region are blocked! In a way, more parallel than π-calculus.

A bit of discussion

Theorem

$$M \to N \implies M^\star \xrightarrow{+} N^\star + \cdots .$$

- The bisimulation result is yet to be precised and proved: probably based on some notion of observable reduction.
- Unlike π-calculus and its translation, the prefixing here is selective: only operations on the same region are blocked! In a way, more parallel than π-calculus.

A bit of discussion

Theorem

$M \to N \implies M^\star \xrightarrow{+} N^\star + \cdots.$

- The bisimulation result is yet to be precised and proved: probably based on some notion of observable reduction.
- Unlike π-calculus and its translation, the prefixing here is selective: only operations on the same region are blocked! In a way, more parallel than π-calculus.

Cycles

- Like π-calculus' translation, switching cycles (i.e. incorrect nets) appear very easily.

$$\mathtt{set}(r, \mathtt{get}(r)); \mathtt{set}(r, \mathtt{get}(r))$$

- arrows indicate switching paths (dependecies) through r's data wires.
- backward arrow is wrong, can be avoided using directed communication zones:

- With them single threads are correct.

Cycles

- Like π-calculus' translation, switching cycles (i.e. incorrect nets) appear very easily.

$$\texttt{set}(r, \texttt{get}(r)); \texttt{set}(r, \texttt{get}(r))$$

- arrows indicate switching paths (dependecies) through r's data wires.

- backward arrow is wrong, can be avoided using directed communication zones:

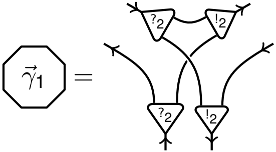
- With them single threads are correct.

Cycles

- Like π-calculus' translation, switching cycles (i.e. incorrect nets) appear very easily.

$$\mathtt{set}(r, \mathtt{get}(r)); \mathtt{set}(r, \mathtt{get}(r))$$

- arrows indicate switching paths (dependecies) through *r*'s data wires.
- backward arrow is wrong, can be avoided using directed communication zones:



- With them single threads are correct.

Cycles in parallel composition

- However, no such workaround for parallel composition:

$$\mathtt{set}(r, \mathtt{get}(r)) | \mathtt{set}(r, \mathtt{get}(r))$$

- Here switching paths shows actual potential dependecies.
- The dashed ones however are mutually exclusive! But this cannot be detected by switching acyclicity (reduction preserves switching paths).
- This happens with any threads updating the same region.
- The same thing happens in π-calculus, e.g.

$$c(x).\overline{c}\langle x\rangle \,|\, c(x).\overline{c}\langle x\rangle$$

- No property derivable from nets!!

Cycles in parallel composition

- However, no such workaround for parallel composition:

$$\texttt{set}(r, \texttt{get}(r)) \mid \texttt{set}(r, \texttt{get}(r))$$

- Here switching paths shows actual potential dependecies.
- The dashed ones however are mutually exclusive! But this cannot be detected by switching acyclicity (reduction preserves switching paths).
- This happens with any threads updating the same region.
- The same thing happens in π-calculus, e.g.

$$c(x).\overline{c}\langle x\rangle \mid c(x).\overline{c}\langle x\rangle$$

- No property derivable from nets!!

Cycles in parallel composition

- However, no such workaround for parallel composition:

$$\texttt{set}(r, \texttt{get}(r)) \mid \texttt{set}(r, \texttt{get}(r))$$

- Here switching paths shows actual potential dependecies.
- The dashed ones however are mutually exclusive! But this cannot be detected by switching acyclicity (reduction preserves switching paths).
- This happens with any threads updating the same region.
- The same thing happens in π-calculus, e.g.

$$c(x).\overline{c}\langle x\rangle \mid c(x).\overline{c}\langle x\rangle$$

- No property derivable from nets!!

Cycles in parallel composition

- However, no such workaround for parallel composition:

$$\texttt{set}(r, \texttt{get}(r)) \mid \texttt{set}(r, \texttt{get}(r))$$

- Here switching paths shows actual potential dependecies.
- The dashed ones however are mutually exclusive! But this cannot be detected by switching acyclicity (reduction preserves switching paths).
- This happens with any threads updating the same region.
- The same thing happens in π-calculus, e.g.

$$c(x).\overline{c}\,\langle x\rangle \mid c(x).\overline{c}\,\langle x\rangle$$

- No property derivable from nets!!

What can be done?

- Prove that these cycles do not disturb the observable reduction used for bisimulation? I.e. relax correctness criterion.
- Add structure to nets, e.g. syntactic mutual exclusion edges?
- Find subcalculi that fit in switching acyclicity? (but threads updating a same variable are hard to leave out)

Another approach

- Let us concentrate on termination.
- Take two typed & stratified sequential programs M and N accessing region r.
- $\nu r \Leftarrow V.M$ terminates.
- $\nu r \Leftarrow V.N$ terminates too.
- What about $\nu r \Leftarrow V.(M|N)$? Interleaved reduction...

$$\nu r \Leftarrow V.(M|N) \xrightarrow{+} \nu r \Leftarrow V_1.(M_1|N) \xrightarrow{+} \nu r \Leftarrow V_2.(M_1|N_1) \xrightarrow{+}$$

$$\cdots \xrightarrow{+} r \Leftarrow V_{2k+1}.(M_{k+1}|N_k) \xrightarrow{+} r \Leftarrow V_{2k+2}.(M_{k+1}|N_{k+1}) \cdots$$

- What if we were able to prove that $\nu r \Leftarrow \{V_0, V_1, \ldots, V_k, \ldots\}.M$ terminates?
(i.e. $\nu r \Leftarrow \mu.\,\mathtt{get}(r) \to V$ nondeterministically for any $V \in \mu$.)

Another approach

- Let us concentrate on termination.
- Take two typed & stratified sequential programs M and N accessing region r.
- $\nu r{\Leftarrow}V.M$ terminates.
- $\nu r{\Leftarrow}V.N$ terminates too.
- What about $\nu r{\Leftarrow}V.(M|N)$? Interleaved reduction...

$$\nu r{\Leftarrow}V.(M|N) \xrightarrow{+} \nu r{\Leftarrow}V_1.(M_1|N) \xrightarrow{+} \nu r{\Leftarrow}V_2.(M_1|N_1) \xrightarrow{+} \cdots \xrightarrow{+} r{\Leftarrow}V_{2k+1}.(M_{k+1}|N_k) \xrightarrow{+} r{\Leftarrow}V_{2k+2}.(M_{k+1}|N_{k+1}) \cdots$$

- What if we were able to prove that $\nu r{\Leftarrow}\{V_0, V_1, \ldots, V_k, \ldots\}.M$ terminates?
(i.e. $\nu r{\Leftarrow}\mu.\,\mathtt{get}(r) \to V$ nondeterministically for any $V \in \mu$.)

Another approach

- Let us concentrate on termination.
- Take two typed & stratified sequential programs M and N accessing region r.
- $\nu r {\Leftarrow} V.M$ terminates.
- $\nu r {\Leftarrow} V.N$ terminates too.
- What about $\nu r {\Leftarrow} V.(M|N)$? Interleaved reduction…

$$\nu r {\Leftarrow} V.(M|N) \xrightarrow{+} \nu r {\Leftarrow} V_1.(M_1|N) \xrightarrow{+} \nu r {\Leftarrow} V_2.(M_1|N_1) \xrightarrow{+} \cdots \xrightarrow{+} r {\Leftarrow} V_{2k+1}.(M_{k+1}|N_k) \xrightarrow{+} r {\Leftarrow} V_{2k+2}.(M_{k+1}|N_{k+1}) \cdots$$

- What if we were able to prove that $\nu r {\Leftarrow} \{V_0, V_1, \ldots, V_k, \ldots\}.M$ terminates?
  (i.e. $\nu r {\Leftarrow} \mu.\,\mathtt{get}(r) \to V$ nondeterministically for any $V \in \mu$.)

Another approach

- Let us concentrate on termination.
- Take two typed & stratified sequential programs M and N accessing region r.
- $\nu r {\Leftarrow} V.M$ terminates.
- $\nu r {\Leftarrow} V.N$ terminates too.
- What about $\nu r {\Leftarrow} V.(M|N)$? Interleaved reduction. . .

$$\nu r{\Leftarrow}V.(M|N) \xrightarrow{+} \nu r{\Leftarrow}V_1.(M_1|N) \xrightarrow{+} \nu r{\Leftarrow}V_2.(M_1|N_1) \xrightarrow{+} \cdots \xrightarrow{+} r{\Leftarrow}V_{2k+1}.(M_{k+1}|N_k) \xrightarrow{+} r{\Leftarrow}V_{2k+2}.(M_{k+1}|N_{k+1}) \cdots$$

- What if we were able to prove that $\nu r{\Leftarrow}\{V_0, V_1, \ldots, V_k, \ldots\}.M$ terminates?
(i.e. $\nu r{\Leftarrow}\mu.\,\mathtt{get}(r) \to V$ nondeterministically for any $V \in \mu$.)

Another approach

- Let us concentrate on termination.
- Take two typed & stratified sequential programs M and N accessing region r.

- $\nu r \Leftarrow V.M$ terminates.
- $\nu r \Leftarrow V.N$ terminates too.
- What about $\nu r \Leftarrow V.(M|N)$? Interleaved reduction...

$$\nu r \Leftarrow V.(M|N) \xrightarrow{+} \nu r \Leftarrow V_1.(M_1|N) \xrightarrow{+} \nu r \Leftarrow V_2.(M_1|N_1) \xrightarrow{+} \cdots \xrightarrow{+} r \Leftarrow V_{2k+1}.(M_{k+1}|N_k) \xrightarrow{+} r \Leftarrow V_{2k+2}.(M_{k+1}|N_{k+1}) \cdots$$

- What if we were able to prove that $\nu r \Leftarrow \{V_0, V_1, \ldots, V_k, \ldots\}.M$ terminates?
(i.e. $\nu r \Leftarrow \mu.\,\mathtt{get}(r) \to V$ nondeterministically for any $V \in \mu$.)

Another approach

- Let us concentrate on termination.
- Take two typed & stratified sequential programs M and N accessing region r.

- $\nu r \Leftarrow V.M$ terminates.
- $\nu r \Leftarrow V.N$ terminates too.
- What about $\nu r \Leftarrow V.(M|N)$? Interleaved reduction…

$$\nu r \Leftarrow V.(M|N) \xrightarrow{+} \nu r \Leftarrow V_1.(M_1|N) \xrightarrow{+} \nu r \Leftarrow V_2.(M_1|N_1) \xrightarrow{+} \cdots \xrightarrow{+} r \Leftarrow V_{2k+1}.(M_{k+1}|N_k) \xrightarrow{+} r \Leftarrow V_{2k+2}.(M_{k+1}|N_{k+1}) \cdots$$

- What if we were able to prove that $\nu r \Leftarrow \{V_0, V_1, \ldots, V_k, \ldots\}.M$ terminates?
(i.e. $\nu r \Leftarrow \mu.\,\mathtt{get}(r) \to V$ nondeterministically for any $V \in \mu$.)

Infinite memory cells

Let Λ_∞ be given by

- Terms: $x \mid \langle\rangle \mid \lambda x.M \mid MN \mid \mathtt{get}(r) \mid (M|N)$.
  (memory is read-only)
- Programs: M, S.
- Stores: S functions from regions to sets of closed values.
  (possibly infinite)

$$E[(\lambda x.M)V], S \to E[M\{V/x\}], S$$

$$E[\mathtt{get}(r)], S \to E[V], S \quad \text{with } V \in S(r).$$

Proving termination with Λ_∞

Take any region based calculus. All we need to prove its termination is

- a forgetful mapping $M^\downarrow$ to Λ_∞ translating all memory ops except access into silent actions.
- a mapping $\Phi^\downarrow$ from reduction chains to stores, with $\Phi^\downarrow(r)$ containing all $V^\downarrow$ for V assigned to an r-marked cell during Φ.
- a discipline (e.g. stratification) preserved by $(\,.\,)^\downarrow$ and ensuring termination in Λ_∞.

Then $M^\downarrow, \Phi^\downarrow$ simulates R (among many nondeterministic branches!).

For example:

$$(\nu r{\Leftarrow}M.N)^\downarrow = M^\downarrow; IN^\downarrow, \quad (\nu r{\Leftarrow}V.N)^\downarrow = IN^\downarrow, \quad (\mathtt{set}(r, M))^\downarrow = M^\downarrow; \langle\rangle$$

$$\Phi^\downarrow(r) = \{\, V^\downarrow \mid \nu r{\Leftarrow}V.M \text{ is subterm of } N \in \Phi \,\}$$

Proving termination with Λ_∞

Take any region based calculus. All we need to prove its termination is

- a forgetful mapping $M^\downarrow$ to Λ_∞ translating all memory ops except access into silent actions.
- a mapping $\Phi^\downarrow$ from reduction chains to stores, with $\Phi^\downarrow(r)$ containing all $V^\downarrow$ for V assigned to an r-marked cell during Φ.
- a discipline (e.g. stratification) preserved by $(\,.\,)^\downarrow$ and ensuring termination in Λ_∞.

Then $M^\downarrow, \Phi^\downarrow$ simulates R (among many nondeterministic branches!).

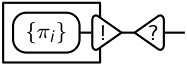
For example:

$$(\nu r {\Leftarrow} M.N)^\downarrow = M^\downarrow; IN^\downarrow, \quad (\nu r {\Leftarrow} V.N)^\downarrow = IN^\downarrow, \quad (\mathtt{set}(r, M))^\downarrow = M^\downarrow; \langle\rangle$$

$$\Phi^\downarrow(r) = \{\, V^\downarrow \mid \nu r {\Leftarrow} V.M \text{ is subterm of } N \in \Phi \,\}$$

Infinite boxes

- LL_∞: regular LL (no cocontraction, coweakening nor codereliction), where boxes contain sets of nets.



$\xrightarrow{e} \pi_i$ for any i.

- Need care (deep structural red. $\rightsquigarrow$ infinite red., so we revert to a form of the so-called quotienting “nouvelle syntaxe”).
- For the purpose of Λ_∞, we can be strict:
  - infinite boxes at depth 0 only;
  - no infinite box on auxiliary port cut (by typing).

Theorem

Surface reduction of simply typed LL_∞ *terminates.*

Termination of stratified Λ_∞

- Translation of Λ_∞ in LL_∞: a matter of simple adaptation of the one of Λ_{reg} in LL.

Theorem

M, S evaluates to V, S iff $(M, S)^\bullet$ normalizes to $(V, S)^\bullet$.

- S typed under region context R if $\mathrm{dom}(S) \subseteq \mathrm{dom}(R)$ and all $V \in S(r)$ typed by $R(r)$.
- M, S typed with stratified region R, then $(M, S)^\bullet$ is simply typed.

Theorem

If M, S is simply typed, then all its reductions terminate.

- Direct proof certainly possible, but for now I prefer playing with infinite boxes :)

Termination of stratified Λ_∞

- Translation of Λ_∞ in LL_∞: a matter of simple adaptation of the one of Λ_{reg} in LL.

Theorem

M, S evaluates to V, S iff $(M, S)^\bullet$ normalizes to $(V, S)^\bullet$.

- S typed under region context R if $\mathrm{dom}(S) \subseteq \mathrm{dom}(R)$ and all $V \in S(r)$ typed by $R(r)$.
- M, S typed with stratified region R, then $(M, S)^\bullet$ is simply typed.

Theorem

If M, S is simply typed, then all its reductions terminate.

- Direct proof certainly possible, but for now I prefer playing with infinite boxes :)

Termination of stratified Λ_∞

- Translation of Λ_∞ in LL_∞: a matter of simple adaptation of the one of Λ_{reg} in LL.

Theorem

M, S evaluates to V, S iff $(M, S)^\bullet$ normalizes to $(V, S)^\bullet$.

- S typed under region context R if $\mathrm{dom}(S) \subseteq \mathrm{dom}(R)$ and all $V \in S(r)$ typed by $R(r)$.
- M, S typed with stratified region R, then $(M, S)^\bullet$ is simply typed.

Theorem

If M, S is simply typed, then all its reductions terminate.

- Direct proof certainly possible, but for now I prefer playing with infinite boxes :)

What's next

- Study LL_∞.
- Carry over results to second order.
- Adapt to region polymorphism.
- Design a sensible stratification discipline for real world languages (ML and its dialects) ensuring termination.

Thanks

Questions?