Types and effects seen through linear logic

Paolo Tranquilli

paolo.tranquilli@ens-lyon.fr

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon

PICS-CONCERTO meeting, 10/06/2010

Outline

1. Lambda-calculus with regions
   - Memory
   - Types and effects

2. Translating into Proof Nets
   - The target
   - The translation

Intro

- Side effects:
  all that is not functional, i.e. not the mere input-output relation.
- e.g. memory, I/O, exceptions, messages, continuations,...
- Memory is challenging for static analysis: behaviour depends on runtime external behaviour.

```
function f(x)
  return x + 1;
```

```
function g(x)
  y := y + 1;
  return x + y;
```

Types and effects

- One approach is types and effects systems.
- One adds an abstract description of side effects to usual types: e.g. $A \xrightarrow{e} B$ types procedures having effects e.
- Memory access: locations are divided in regions (r, s, . . .), effects are sets of regions.
- $A \xrightarrow{\{r_1,\ldots,r_k\}} B$: reads, writes, allocates or frees locations in r_i (finer distinctions possible).
- Analysis can be used to parallelize evaluation, or make safe garbage collection.

J. M. Lucassen and D. K. Gifford.
Polymorphic effect systems.
In *POPL '88*, pages 47–57, New York, NY, USA, 1988. ACM.

A toy language

We will work on Λ_{reg}, a call-by-value calculus with two basic memory access ops (`set` and `get`), where regions *are* locations.

$$\texttt{set}(r, M) \qquad \texttt{get}(r).$$

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

The syntax of Λ_{reg}

Functions are values:

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

Terms can also be memory ops:

$$M, N ::= V \mid MN \mid \mathtt{set}(r, M) \mid \mathtt{get}(r)$$

Call-by-value order enforced via evaluation contexts:

$$E, F ::= [\,] \mid EM \mid VE \mid \mathtt{set}(r, E)$$

Memory represented by stores:

$$S, T ::= r_1 \Leftarrow V_1, \ldots, r_k \Leftarrow V_k$$

Evaluation

Call-by-value beta-reduction:

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

Reading from memory:

$$E[\mathtt{get}(r)], r \Leftarrow V, S \to E[V], r \Leftarrow V, S$$

Writing to memory:

$$E[\mathtt{set}(r, V)], r \Leftarrow U, S \to E[\langle\rangle], r \Leftarrow V, S$$

(notice we do not do allocation and garbage collection here)

An example

```
function pow(n, m)
    r := 1;
    for i := 1 to m
        r := n * r;
    return r;
```

$$\begin{aligned}&\mathtt{pow} := \lambda n, m.\\&\quad \mathtt{set}(r, \underline{1});\\&\quad m\\&\qquad (\lambda d.\, \mathtt{set}(r, \mathtt{mult}\; n\; \mathtt{get}(r)))\,\langle\rangle\,;\\&\quad \mathtt{get}(r)\end{aligned}$$

(notation: $M; N := (\lambda d.N)M$)

$$\mathtt{pow}\,\underline{3}\,\underline{2},\ r \Leftarrow \underline{0}$$

$$\to \mathtt{set}(r, \underline{1}); \underline{2}(\lambda d.\, \mathtt{set}(r, \mathtt{mult}\, \underline{3}\, \mathtt{get}(r))\,\langle\rangle\,; \mathtt{get}(r),\ r \Leftarrow \underline{0}$$

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

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

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

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

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

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: r_i contains values of type A_i.
- Typing judgments $R; \Gamma \vdash M : A, e$: means M accesses e.

Typing rules

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

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

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

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

Typing rules

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

Regular axioms, no effects

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

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

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

Typing rules

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

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

Effects annotate arrow type and are reset

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

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

Typing rules

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

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

Effects are merged, annotated ones are "extracted"

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

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

Typing rules

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

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

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

Accessed regions are noted

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

Typing rules

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

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

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

Dummy effects can be added

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

Typed fixpoints

$$r : 1 \xrightarrow{\{r\}} A; \vdash Y = \lambda f.\, \mathtt{set}(r \Leftarrow \lambda x.f(\mathtt{get}(r)x)); \mathtt{get}(r)\,\langle\rangle \; : (A \to A) \to A$$

$$\begin{aligned} YF, r \Leftarrow I &\to \mathtt{set}(r, \lambda x.F(\mathtt{get}(r)x); \mathtt{get}(r)\,\langle\rangle\,, r \Leftarrow I \\ &\to \mathtt{get}(r)\,\langle\rangle\,, r \Leftarrow \lambda x.F(\mathtt{get}(r)x) \\ &\to (\lambda x.F(\mathtt{get}(r)x))\,\langle\rangle\,, r \Leftarrow \lambda x.F(\mathtt{get}(r)x) \\ &\to F(\mathtt{get}(r)\,\langle\rangle), r \Leftarrow \lambda x.F(\mathtt{get}(r)x) \end{aligned}$$

- In particular, $Y(\lambda z.z)$ loops.
- Typing avoids self-application, but not self-reference.

Typed fixpoints

$$r : 1 \xrightarrow{\{r\}} A; \vdash Y = \lambda f.\, \mathtt{set}(r \Leftarrow \lambda x. f(\mathtt{get}(r) x)); \mathtt{get}(r) \langle\rangle \; : (A \to A) \to A$$

$$\begin{aligned} YF, r \Leftarrow I &\to \mathtt{set}(r, \lambda x. F(\mathtt{get}(r) x); \mathtt{get}(r) \langle\rangle, r \Leftarrow I \\ &\to \mathtt{get}(r) \langle\rangle, r \Leftarrow \lambda x. F(\mathtt{get}(r) x) \\ &\to (\lambda x. F(\mathtt{get}(r) x)) \langle\rangle, r \Leftarrow \lambda x. F(\mathtt{get}(r) x) \\ &\to F(\mathtt{get}(r) \langle\rangle), r \Leftarrow \lambda x. F(\mathtt{get}(r) x) \end{aligned}$$

- In particular, $Y(\lambda z.z)$ loops.
- Typing avoids self-application, but not self-reference.

Typed fixpoints

$$r : 1 \xrightarrow{\{r\}} A; \vdash Y = \lambda f.\, \mathtt{set}(r \Leftarrow \lambda x.f(\mathtt{get}(r)x)); \mathtt{get}(r)\,\langle\rangle \;:\; (A \to A) \to A$$

$$\begin{aligned} YF, r \Leftarrow I &\to \mathtt{set}(r, \lambda x.F(\mathtt{get}(r)x); \mathtt{get}(r)\,\langle\rangle\,, r \Leftarrow I \\ &\to \mathtt{get}(r)\,\langle\rangle\,, r \Leftarrow \lambda x.F(\mathtt{get}(r)x) \\ &\to (\lambda x.F(\mathtt{get}(r)x))\,\langle\rangle\,, r \Leftarrow \lambda x.F(\mathtt{get}(r)x) \\ &\to F(\mathtt{get}(r)\,\langle\rangle), r \Leftarrow \lambda x.F(\mathtt{get}(r)x) \end{aligned}$$

- In particular, $Y(\lambda z.z)$ loops.
- Typing avoids self-application, but not self-reference.

Typed fixpoints

$$r : 1 \xrightarrow{\{r\}} A; \vdash Y = \lambda f.\, \mathtt{set}(r \Leftarrow \lambda x.f(\mathtt{get}(r)x)); \mathtt{get}(r)\,\langle\rangle \;:\; (A \to A) \to A$$

$$\begin{aligned} YF, r \Leftarrow I &\to \mathtt{set}(r, \lambda x.F(\mathtt{get}(r)x); \mathtt{get}(r)\,\langle\rangle\,, r \Leftarrow I \\ &\to \mathtt{get}(r)\,\langle\rangle\,, r \Leftarrow \lambda x.F(\mathtt{get}(r)x) \\ &\to (\lambda x.F(\mathtt{get}(r)x))\,\langle\rangle\,, r \Leftarrow \lambda x.F(\mathtt{get}(r)x) \\ &\to F(\mathtt{get}(r)\,\langle\rangle), r \Leftarrow \lambda x.F(\mathtt{get}(r)x) \end{aligned}$$

- In particular, $Y(\lambda z.z)$ loops.
- Typing avoids self-application, but not self-reference.

Typed fixpoints

$$r : 1 \xrightarrow{\{r\}} A; \vdash Y = \lambda f.\, \mathtt{set}(r \Leftarrow \lambda x.f(\mathtt{get}(r)x)); \mathtt{get}(r)\,\langle\rangle\ : (A \to A) \to A$$

$$\begin{aligned} YF, r \Leftarrow I &\to \mathtt{set}(r, \lambda x.F(\mathtt{get}(r)x); \mathtt{get}(r)\,\langle\rangle\,, r \Leftarrow I \\ &\to \mathtt{get}(r)\,\langle\rangle\,, r \Leftarrow \lambda x.F(\mathtt{get}(r)x) \\ &\to (\lambda x.F(\mathtt{get}(r)x))\,\langle\rangle\,, r \Leftarrow \lambda x.F(\mathtt{get}(r)x) \\ &\to F(\mathtt{get}(r)\,\langle\rangle), r \Leftarrow \lambda x.F(\mathtt{get}(r)x) \end{aligned}$$

- In particular, $Y(\lambda z.z)$ loops.
- Typing avoids self-application, but not self-reference.

Typed fixpoints

$$r : 1 \stackrel{\{r\}}{\to} A; \vdash Y = \lambda f.\, \mathtt{set}(r \Leftarrow \lambda x.f(\mathtt{get}(r)x)); \mathtt{get}(r)\,\langle\rangle \;:\; (A \to A) \to A$$

$$\begin{aligned} YF, r \Leftarrow I &\to \mathtt{set}(r, \lambda x.F(\mathtt{get}(r)x); \mathtt{get}(r)\,\langle\rangle\,, r \Leftarrow I \\ &\to \mathtt{get}(r)\,\langle\rangle\,, r \Leftarrow \lambda x.F(\mathtt{get}(r)x) \\ &\to (\lambda x.F(\mathtt{get}(r)x))\,\langle\rangle\,, r \Leftarrow \lambda x.F(\mathtt{get}(r)x) \\ &\to F(\mathtt{get}(r)\,\langle\rangle), r \Leftarrow \lambda x.F(\mathtt{get}(r)x) \end{aligned}$$

- In particular, $Y(\lambda z.z)$ loops.
- Typing avoids self-application, but not self-reference.

Typed fixpoints

$$r : 1 \xrightarrow{\{r\}} A; \vdash Y = \lambda f.\,\mathtt{set}(r \Leftarrow \lambda x.f(\mathtt{get}(r)x)); \mathtt{get}(r)\,\langle\rangle \ : (A \to A) \to A$$

$$\begin{aligned} YF, r \Leftarrow I &\to \ \mathtt{set}(r, \lambda x.F(\mathtt{get}(r)x); \mathtt{get}(r)\,\langle\rangle\,, r \Leftarrow I \\ &\to \mathtt{get}(r)\,\langle\rangle\,, r \Leftarrow \lambda x.F(\mathtt{get}(r)x) \\ &\to (\lambda x.F(\mathtt{get}(r)x))\,\langle\rangle\,, r \Leftarrow \lambda x.F(\mathtt{get}(r)x) \\ &\to F(\,\mathtt{get}(r)\,\langle\rangle), r \Leftarrow \lambda x.F(\mathtt{get}(r)x) \end{aligned}$$

- In particular, $Y(\lambda z.z)$ loops.
- Typing avoids self-application, but not self-reference.

Stratification

- Intuition: stratify regions, so that "lower" regions cannot reference "higher" ones (in particular themselves).

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

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

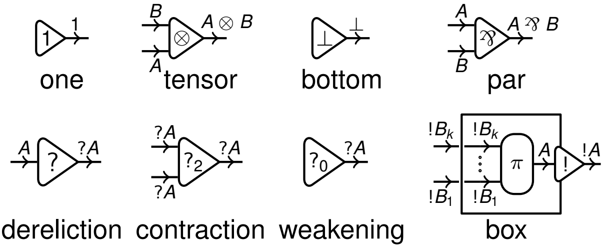
Gérard Boudol.
Fair cooperative multithreading.
In *CONCUR*, volume 4703 of *LNCS*, pages 272–286. Springer, 2007.

Roberto M. Amadio.
On stratified regions.
In *APLAS*, volume 5904 of *LNCS*, pages 210–225. Springer, 2009.

- For example: ~~$r : 1 \xrightarrow{\{r\}} A \vdash$~~ as $1 \xrightarrow{\{r\}} A$ not definable from $\emptyset$.
- $R \vdash$ and $R; \vdash M : A, \emptyset \implies M$ terminates.

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:



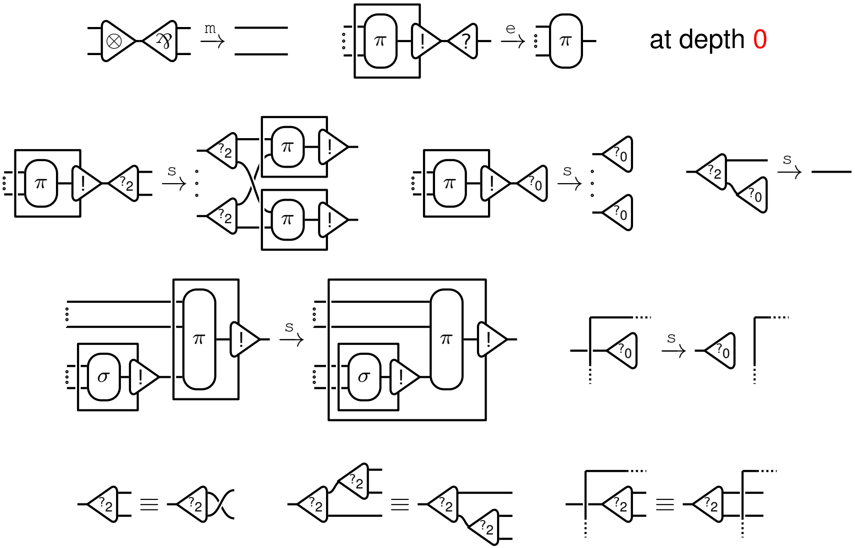
- Proof nets formed matching wires and enforcing a correctness criterion.

The target

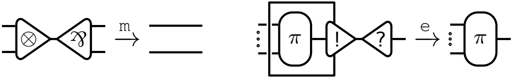
- Proof nets are the parallel representation of linear logic proofs.
- Types: $X \mid X^\perp \mid 1 \mid \perp \mid A \otimes B \mid A \parr B \mid !A \mid ?A$ with duality $A^\perp$, linear arrow $A \multimap B = A^\perp \parr B$, systems of equations $X_i \doteq A_i$.
- Cells:

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

Surface reduction

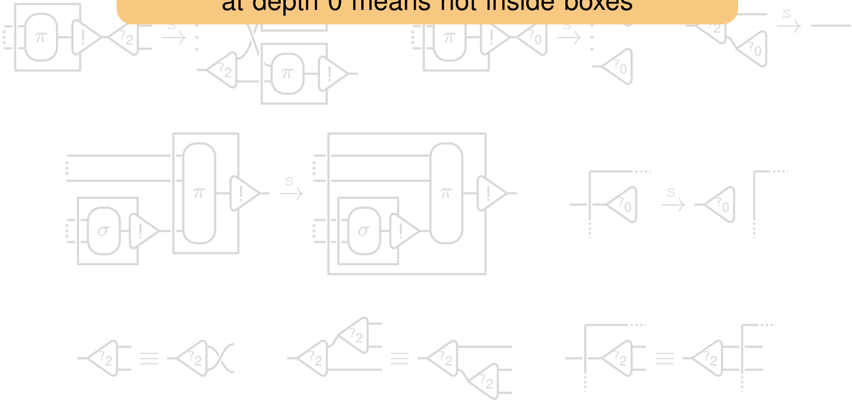


Surface reduction

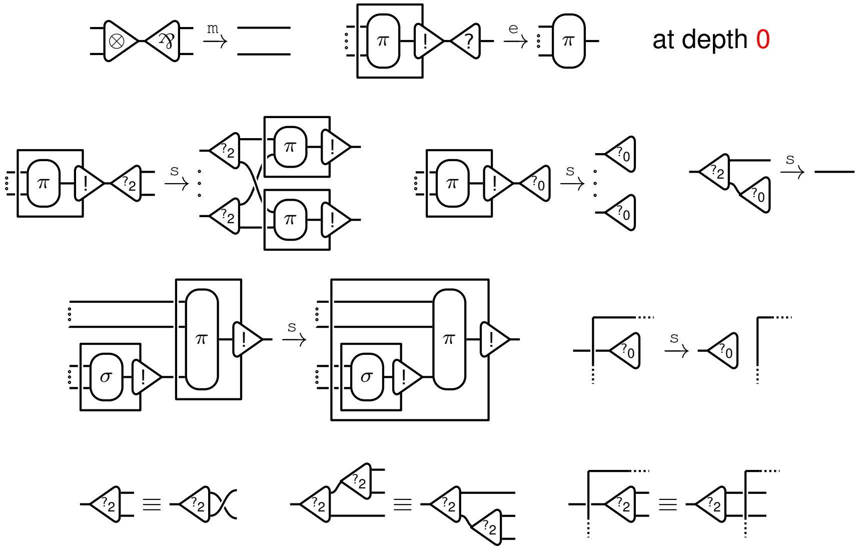


at depth 0

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



Surface reduction



The results

We present a translation $(M, S)^\bullet$ from typed Λ_{reg} programs to resursively typed proof nets, generalizing Girard's call-by-value translation (i.e. $(A \to B)^\bullet = !(A^\bullet \multimap B^\bullet)$).

Theorem

If $M, S \to N, T$ then $(M, S)^\bullet \stackrel{\mathtt{e}}{\to}\stackrel{\mathtt{m*}}{\longrightarrow}\stackrel{\mathtt{s*}}{\longrightarrow} (N, T)^\bullet$.

Theorem

$(M, S)^\bullet$ normalizes by surface reduction to π iff $\pi = (V, T)^\bullet$ and $M, S \stackrel{*}{\to} V, T$.

Recursive equations come from a translation of region contexts $R^\bullet$.

Theorem

R is stratified iff $R^\bullet$ is solvable (i.e. no real recursive types!)

The results

We present a translation $(M, S)^\bullet$ from typed Λ_{reg} programs to resursively typed proof nets, generalizing Girard's call-by-value translation (i.e. $(A \to B)^\bullet = !(A^\bullet \multimap B^\bullet)$).

Theorem

If $M, S \to N, T$ then $(M, S)^\bullet \xrightarrow{e}\xrightarrow{m*}\xrightarrow{s*} (N, T)^\bullet$.

Theorem

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

Recursive equations come from a translation of region contexts $R^\bullet$.

Theorem

R is stratified iff $R^\bullet$ is solvable (i.e. no real recursive types!)

The results

We present a translation $(M, S)^\bullet$ from typed Λ_{reg} programs to resursively typed proof nets, generalizing Girard's call-by-value translation (i.e. $(A \to B)^\bullet = !(A^\bullet \multimap B^\bullet)$).

Theorem

If $M, S \to N, T$ then $(M, S)^\bullet \xrightarrow{e}\xrightarrow{m*}\xrightarrow{s*} (N, T)^\bullet$.

Theorem

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

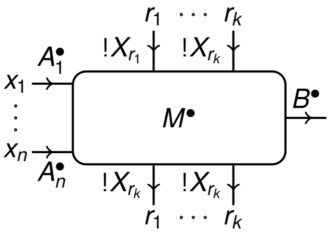
Recursive equations come from a translation of region contexts $R^\bullet$.

Theorem

R is stratified iff $R^\bullet$ is solvable (i.e. no real recursive types!)

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

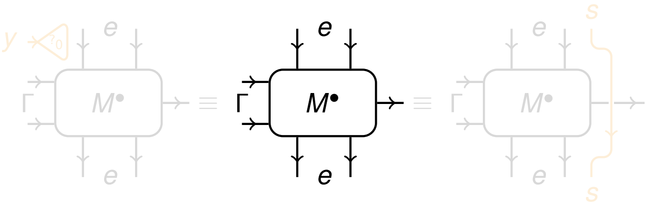


(we will show the translation of types and effects along the way)

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

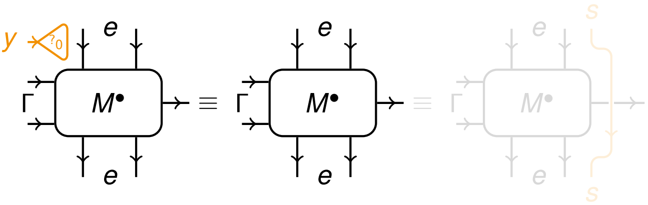
Dummy variables and dummy effects

We consider translations up to dummy variables and dummy effects.



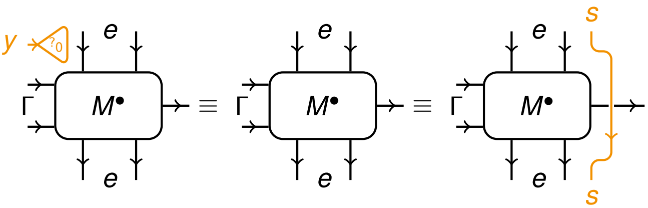
Dummy variables and dummy effects

We consider translations up to dummy variables and dummy effects.



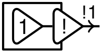
Dummy variables and dummy effects

We consider translations up to dummy variables and dummy effects.



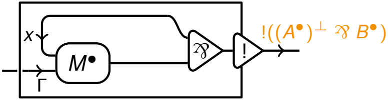
The translation: variable and unit

$x^\bullet =$ 

$\langle\rangle^\bullet =$ 

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

The translation: abstraction

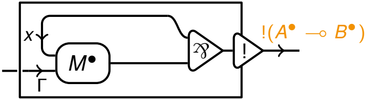
$(\lambda x.M)^\bullet =$ 

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

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

(anybody sees anything familiar? Some tidbits on state monads later...)

The translation: abstraction

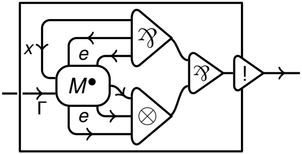
$(\lambda x.M)^\bullet =$ 

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

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

(anybody sees anything familiar? Some tidbits on state monads later. . .)

The translation: abstraction

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

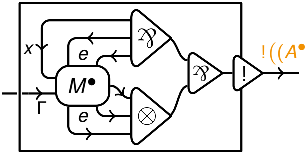
Usual call-by-value translation extended by encapsulating the effects.

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

(anybody sees anything familiar? Some tidbits on state monads later...)

The translation: abstraction

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



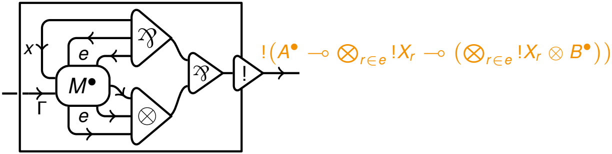
$!((A^\bullet)^\perp \parr \bigparr_{r\in e} ?X_r^\perp \parr (\bigotimes_{r\in e} !X_r \otimes B^\bullet))$

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

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

(anybody sees anything familiar? Some tidbits on state monads later…)

The translation: abstraction

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

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

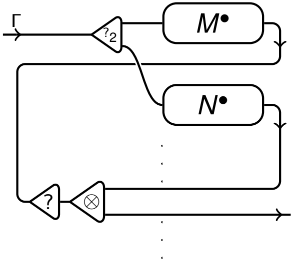
Types: $e^\bullet = \bigotimes_{r \in e} !X_r, \quad (A \xrightarrow{e} B)^\bullet = !(A^\bullet \multimap e^\bullet \multimap (e^\bullet \otimes B^\bullet)).$

(anybody sees anything familiar? Some tidbits on state monads later. . .)

The translation: application

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

$(MN)^{\bullet} =$

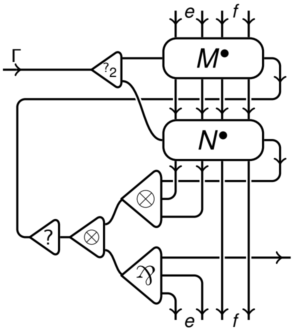


Usual translation extended by extracting effects and linking in evaluation order.

The translation: application

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

$(MN)^\bullet =$



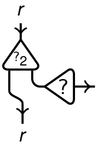
Usual translation extended by extracting effects and linking in evaluation order.

The translation of memory operations: get(r) and set(r, M).

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

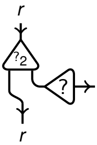
$(\texttt{set}(r, M))^\bullet =$

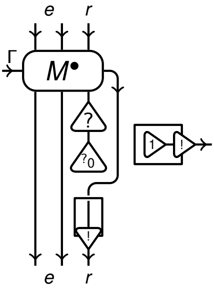
The translation of memory operations: get(r) and set(r, M).

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

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

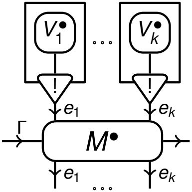
The translation of memory operations: $\texttt{get}(r)$ and $\texttt{set}(r, M)$.

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

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

The translation: stores

$$(M, r_1 \Leftarrow V_1, \ldots, r_k \Leftarrow V_k)^\bullet =$$



(for proofnet geeks like me:
adding garbage collection by weakenings $\rightsquigarrow$ switching connectedness)

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. $(M, S)^\bullet$ simply typed!).

Theorem

If $M, S \to N, T$ then $(M, S)^\bullet \xrightarrow{\mathtt{e}}\xrightarrow{\mathtt{m*}}\xrightarrow{\mathtt{s*}} (N, T)^\bullet$.

Theorem

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

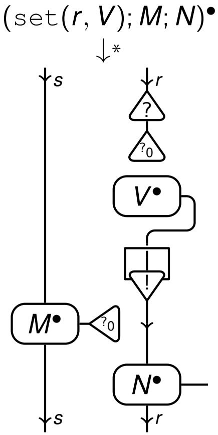
Proof nets as parallel evaluators

- 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.



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

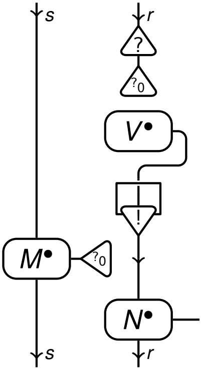
Theorem

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

ensures sequential semantics is preserved.

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

$\downarrow^*$



What was left out of this talk

- In fact the translation can be carried out completely in λ-calculus, using localized state monads $T_e A = S_e \to (S_e \times A)$ with $S_e = \prod_{r \in e} X_r$ and corresponding return and bind operators. (though the graphical parallel evaluation intuition is lost)
- Multithreading and differential nets (from state passing style to lock passing style)
- Allocation/garbage collection operations (stuff like $\nu r \Leftarrow V.M$)

Thanks!

Questions?