Types and Effects: from Monads to Differential Nets

Part I: Regions, Stratification, Monads

Paolo Tranquilli

`paolo.tranquilli@ens-lyon.fr`

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon

Séminaire LCR
LIPN, 01/02/2010

Outline

1. The λ-calculus with regions
    - Syntax and evaluation
    - Typing and stratification
2. Towards ordinary λ-calculus via monads
    - The result
    - Localized state monads
    - The translation
3. Conclusion

Outline

Types and effects

- Types and effects systems statically analyze side effects (e.g. memory, I/O, exceptions, messages, continuations,. . .) by enriching types.
- Usual notation: $A \xrightarrow{e} B$ types procedures having effects e.
- Memory access: locations are divided in regions (r, s,. . .), effects are sets of regions.
- $A \xrightarrow{e} B$: reads, writes, allocates or frees locations in e (finer distinctions possible).

  J. M. Lucassen and D. K. Gifford.
  Polymorphic effect systems.
  In *POPL '88*, pages 47–57, New York, NY, USA, 1988. ACM.
- Analysis can be used to parallelize evaluation, or make safe garbage collection.

The syntax of Λ_{reg}

A call-by-value calculus with two basic memory access ops (`set` and `get`) and a memory allocation/deallocation op (ν).

Functions are values:

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

Terms can also be memory management operations:

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

Call-by-value order enforced via evaluation contexts:

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

The syntax of Λ_{reg}

A call-by-value calculus with two basic memory access ops (`set` and `get`) and a memory allocation/deallocation op (ν).

Functions are values:

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

Terms can also be memory management operations:

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

Call-by-value order enforced via evaluation contexts:

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

Evaluation

- Intuition: νr's allocate, represent and garbage collect memory.
- PR(E) (private regions of E) are r's for which E's hole is in the scope of a νr.

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

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

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

Evaluation

- Intuition: νr's allocate, represent and garbage collect memory.
- PR(E) (private regions of E) are r's for which E's hole is in the scope of a νr.

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

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

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

Evaluation

- Intuition: νr's allocate, represent and garbage collect memory.
- PR(E) (private regions of E) are r's for which E's hole is in the scope of a νr.

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

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

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

Regular beta-reduction

Evaluation

- Intuition: νr's allocate, represent and garbage collect memory.
- PR(E) (private regions of E) are r's for which E's hole is in the scope of a νr.

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

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

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

Memory access operations, setting and getting:
"closer" assignment for r is considered

Evaluation

- Intuition: νr's allocate, represent and garbage collect memory.
- PR(E) (private regions of E) are r's for which E's hole is in the scope of a νr.

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

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

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

Memory management:
space allocated by ν is garbage collected at end of evaluation

An example

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

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

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

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

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

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

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

An example

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

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

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

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

An example

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

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

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

An example

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

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

```
$\mathtt{pow} := \lambda n, m.$
    $\nu r \Leftarrow \underline{1}.$
    $m$
        $(\lambda d.\, \mathtt{set}(r, \mathtt{mult}\, n\, \mathtt{get}(r)))\, \langle\rangle\, ;$
    $\mathtt{get}(r)$
```

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

An example

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

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

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

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

An example

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

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

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

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

An example

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

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

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

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

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

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

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

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

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

An example

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

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

```
pow $:= \lambda n, m.$
    $\nu r \Leftarrow \underline{1}.$
    $m$
        $(\lambda d.\,\mathtt{set}(r, \mathtt{mult}\; n\; \mathtt{get}(r)))\,\langle\rangle\,;$
    $\mathtt{get}(r)$
```

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

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

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

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

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

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

An example

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

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

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

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

An example

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

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

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

An example

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

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

```
pow $:= \lambda n, m.$
    $\nu r \Leftarrow \underline{1}.$
    $m$
        $(\lambda d.\,\mathtt{set}(r, \mathtt{mult}\ n\ \mathtt{get}(r)))\ \langle\rangle$ ;
    $\mathtt{get}(r)$
```

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

An example

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

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

```
pow $:= \lambda n, m.$
    $\nu r{\Leftarrow}\underline{1}.$
    $m$
        $(\lambda d.\,\mathtt{set}(r, \mathtt{mult}\; n\; \mathtt{get}(r)))\,\langle\rangle\;;$
    $\mathtt{get}(r)$
```

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

An example

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

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

```
pow := $\lambda n, m.$
    $\nu r \Leftarrow \underline{1}.$
    $m$
        $(\lambda d.\, \mathtt{set}(r, \mathtt{mult}\, n\, \mathtt{get}(r)))\, \langle\rangle$ ;
    $\mathtt{get}(r)$
```

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

An example

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

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

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

Another example

- $\texttt{true} := \lambda x, y.x$, $\texttt{false} := \lambda x, y.y$.

$$
\begin{aligned}
&\nu r \Leftarrow \texttt{true}.((\nu r \Leftarrow \texttt{false}.\lambda d\,\texttt{get}(r))\,\langle\rangle\,U_1 U_2)\\
&\to \nu r \Leftarrow \texttt{true}.((\lambda d.\,\texttt{get}(r))\,\langle\rangle\,U_1 U_2)\\
&\to \nu r \Leftarrow \texttt{true}.\texttt{get}(r) U_1 U_2)\\
&\to \nu r \Leftarrow \texttt{true}.\texttt{true}\,U_1 U_2\\
&\xrightarrow{*} \nu r \Leftarrow \texttt{true}.U_1\\
&\to U_1
\end{aligned}
$$

- Notice νr does not bind: internal $\texttt{get}(r)$ gets outer value of r.
- Regions are stacks (similaraties with Forth, PostScript or TeX).

Another example

- $\mathtt{true} := \lambda x, y.x$, $\mathtt{false} := \lambda x, y.y$.

$$\begin{aligned}
&\nu r \Leftarrow \mathtt{true}.((\nu r \Leftarrow \mathtt{false}.\lambda d\,\mathtt{get}(r))\,\langle\rangle\,U_1 U_2)\\
&\quad\to \nu r \Leftarrow \mathtt{true}.((\lambda d.\,\mathtt{get}(r))\,\langle\rangle\,U_1 U_2)\\
&\quad\to \nu r \Leftarrow \mathtt{true}.\mathtt{get}(r) U_1 U_2)\\
&\quad\to \nu r \Leftarrow \mathtt{true}.\mathtt{true}\,U_1 U_2\\
&\quad\overset{*}{\to} \nu r \Leftarrow \mathtt{true}.U_1\\
&\quad\to U_1
\end{aligned}$$

- Notice νr does not bind: internal $\mathtt{get}(r)$ gets outer value of r.
- Regions are stacks (similaraties with Forth, PostScript or TeX).

Another example

- $\mathtt{true} := \lambda x, y.x$, $\mathtt{false} := \lambda x, y.y$.

$$\begin{aligned}
&\nu r \Leftarrow \mathtt{true}.\big((\nu r \Leftarrow \mathtt{false}.\lambda d\, \mathtt{get}(r))\, \langle\rangle\, U_1\, U_2\big)\\
&\quad \to \nu r \Leftarrow \mathtt{true}.\big((\lambda d.\, \mathtt{get}(r))\, \langle\rangle\, U_1\, U_2\big)\\
&\quad \to \nu r \Leftarrow \mathtt{true}.\mathtt{get}(r) U_1\, U_2\big)\\
&\quad \to \nu r \Leftarrow \mathtt{true}.\mathtt{true}\, U_1\, U_2\\
&\quad \xrightarrow{*} \nu r \Leftarrow \mathtt{true}.U_1\\
&\quad \to U_1
\end{aligned}$$

- Notice νr does not bind: internal $\mathtt{get}(r)$ gets outer value of r.
- Regions are stacks (similaraties with Forth, PostScript or TEX).

Another example

- $\mathtt{true} := \lambda x, y.x$, $\mathtt{false} := \lambda x, y.y$.

$$
\begin{aligned}
&\nu r \Leftarrow \mathtt{true}.((\nu r \Leftarrow \mathtt{false}.\lambda d\, \mathtt{get}(r))\, \langle\rangle\, U_1\, U_2) \\
&\quad \to \nu r \Leftarrow \mathtt{true}.((\lambda d.\, \mathtt{get}(r))\, \langle\rangle\, U_1\, U_2) \\
&\quad \to \nu r \Leftarrow \mathtt{true}.\mathtt{get}(r) U_1\, U_2) \\
&\quad \to \nu r \Leftarrow \mathtt{true}.\mathtt{true}\, U_1\, U_2 \\
&\quad \xrightarrow{*} \nu r \Leftarrow \mathtt{true}.U_1 \\
&\quad \to U_1
\end{aligned}
$$

- Notice νr does not bind: internal $\mathtt{get}(r)$ gets outer value of r.
- Regions are stacks (similaraties with Forth, PostScript or TEX).

Another example

- $\mathtt{true} := \lambda x, y.x$, $\mathtt{false} := \lambda x, y.y$.

$$
\begin{aligned}
&\nu r \Leftarrow \mathtt{true}.((\nu r \Leftarrow \mathtt{false}.\lambda d\, \mathtt{get}(r))\, \langle\rangle\, U_1 U_2) \\
&\quad \to \nu r \Leftarrow \mathtt{true}.((\lambda d.\, \mathtt{get}(r))\, \langle\rangle\, U_1 U_2) \\
&\quad \to \nu r \Leftarrow \mathtt{true}.\mathtt{get}(r) U_1 U_2) \\
&\quad \to \nu r \Leftarrow \mathtt{true}.\mathtt{true}\, U_1 U_2 \\
&\quad \xrightarrow{*} \nu r \Leftarrow \mathtt{true}.U_1 \\
&\quad \to U_1
\end{aligned}
$$

- Notice νr does not bind: internal $\mathtt{get}(r)$ gets outer value of r.
- Regions are stacks (similaraties with Forth, PostScript or TEX).

Another example

- $\texttt{true} := \lambda x, y.x$, $\texttt{false} := \lambda x, y.y$.

$$
\begin{aligned}
&\nu r \Leftarrow \texttt{true}.((\nu r \Leftarrow \texttt{false}.\lambda d\, \texttt{get}(r))\, \langle\rangle\, U_1 U_2) \\
&\quad \to \nu r \Leftarrow \texttt{true}.((\lambda d.\, \texttt{get}(r))\, \langle\rangle\, U_1 U_2) \\
&\quad \to \nu r \Leftarrow \texttt{true}.\texttt{get}(r) U_1 U_2) \\
&\quad \to \nu r \Leftarrow \texttt{true}.\texttt{true}\, U_1 U_2 \\
&\quad \xrightarrow{*} \nu r \Leftarrow \texttt{true}.U_1 \\
&\quad \to U_1
\end{aligned}
$$

- Notice νr does not bind: internal $\texttt{get}(r)$ gets outer value of r.
- Regions are stacks (similaraties with Forth, PostScript or TEX).

An equivalent syntax with explicit stores

- Terms: $\ldots \mid \epsilon r.M$ (place marker to garbage collect).
- Evaluation contexts: $\ldots \mid \nu r{\Leftarrow}E.M \mid \epsilon r.E$.
- Stores: words over $r{\Leftarrow}V$, "almost" commutative (stack nature)

$$r{\Leftarrow}U, s{\Leftarrow}V \equiv s{\Leftarrow}V, r{\Leftarrow}U \quad \text{if } r \neq s.$$

- Reduction over pairs M, S:

$$\begin{aligned} E[(\lambda x.M)V], S &\to E[M\{V/x\}], S \\ E[\mathtt{set}(r,U)], r{\Leftarrow}V, S &\to E[\langle\rangle], r{\Leftarrow}U, S \\ E[\mathtt{get}(r)], r{\Leftarrow}V, S &\to E[V], r{\Leftarrow}V, S \\ E[\nu r{\Leftarrow}V.M], S &\to E[\epsilon r.M], r{\Leftarrow}V, S \\ E[\epsilon r.U], r{\Leftarrow}V, S &\to E[U], S \end{aligned}$$

An equivalent syntax with explicit stores

- Terms: $\ldots \mid \epsilon r.M$ (place marker to garbage collect).
- Evaluation contexts: $\ldots \mid \nu r{\Leftarrow}E.M \mid \epsilon r.E$.
- Stores: words over $r{\Leftarrow}V$, "almost" commutative (stack nature)

$$r{\Leftarrow}U, s{\Leftarrow}V \equiv s{\Leftarrow}V, r{\Leftarrow}U \quad \text{if } r \neq s.$$

- Reduction over pairs M, S:

$$\begin{aligned}
E[(\lambda x.M)V], S &\to E[M\{V/x\}], S\\
E[\mathtt{set}(r, U)], r{\Leftarrow}V, S &\to E[\langle\rangle], r{\Leftarrow}U, S\\
E[\mathtt{get}(r)], r{\Leftarrow}V, S &\to E[V], r{\Leftarrow}V, S\\
E[\nu r{\Leftarrow}V.M], S &\to E[\epsilon r.M], r{\Leftarrow}V, S\\
E[\epsilon r.U], r{\Leftarrow}V, S &\to E[U], S
\end{aligned}$$

Outline

Types and effects

- Types: $A ::= 1 \mid A \xrightarrow{e} B$.
- $R = r_1 : A_1, \ldots, r_k : A_k$ is a region context.
- Typing judgments $R; \Gamma \vdash M : A, e$: e are the active effects.
- Types and effects assure type and memory safety (e.g. no runtime type mismatch and no "segmentation fault"), but not termination.

Rules for types and effects

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

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

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

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

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

(stronger subtyping left out...)

Rules for types and effects

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

Regular axioms, no effects

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

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

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

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

(stronger subtyping left out...)

Rules for types and effects

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

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

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

Effects annotate arrow type and are reset

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

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

(stronger subtyping left out. . .)

Rules for types and effects

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

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

Effects are merged, annotated ones are “extracted”

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

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

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

(stronger subtyping left out. . .)

Rules for types and effects

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

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

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

Accessed regions are noted

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

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

(stronger subtyping left out. . .)

Rules for types and effects

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

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

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

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

Stack operations hide effects on region

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

(stronger subtyping left out...)

Rules for types and effects

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

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

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

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

Dummy effects can be added (weak subtyping)

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

(stronger subtyping left out...)

Rules for types and effects

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

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

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

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

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

(stronger subtyping left out. . .)

Evaluation is safe

- Programs are closed and effect-less typed terms: $R; \vdash M : A, \emptyset$.

Proposition (Subject reduction + safeness)

Every program either is a value or reduces to another program with the same type and effects.

- In particular
  - memory access never blocks (`get` and `set` are never values!),
  - garbage collection takes place (ν is never a value!).
- However, no assurance of termination. . .

Typed fixpoints

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

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

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

Typed fixpoints

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

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

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

Typed fixpoints

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

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

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

Typed fixpoints

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

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

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

Typed fixpoints

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

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

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

Typed fixpoints

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

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

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

Typed fixpoints

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

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

- 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).

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

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

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: $\cancel{r : 1 \xrightarrow{\{r\}} A \vdash}$ as $1 \xrightarrow{\{r\}} A$ not definable from $\emptyset$.
- $R \vdash$ and $R; \vdash M : A, \emptyset \implies M$ terminates (with multithreading!).

Stratification

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

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

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

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: $\cancel{r : 1 \xrightarrow{\{r\}} A \vdash}$ as $1 \xrightarrow{\{r\}} A$ not definable from $\emptyset$.
- $R \vdash$ and $R; \vdash M : A, \emptyset \implies M$ terminates (with multithreading!).

Stratification

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

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

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

starting from a stratified context,
one checks if a type can be defined with it...

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

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

- For example: $\cancel{r : 1 \xrightarrow{\{r\}} A \vdash}$ as $1 \xrightarrow{\{r\}} A$ not definable from $\emptyset$.
- $R \vdash$ and $R; \vdash M : A, \emptyset \implies M$ terminates (with multithreading!).

Stratification

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

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

…then such a type can be added to the context,
i.e. region type assignements are ordered

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 (with multithreading!).

Stratification

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

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

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

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 (with multithreading!).

Outline

The target

- Call-by-value λ-calculus with pairs:

$$\begin{aligned} V &::= x \mid \langle\rangle \mid \lambda x.M \mid \langle U, V\rangle \\ M &::= V \mid MN \mid \langle M, N\rangle \mid \pi_1 M \mid \pi_2 M \\ A &::= 1 \mid X \mid A \to B \mid A \times B \\ \Psi &::= X_1 \doteq A_1, \ldots, X_k \doteq A_k \end{aligned}$$

- … with systems of equations possibly definining recursive types (e.g. $X \doteq X \to X$)
- Systems are solvable if they have a solution with closed types (e.g. $X \doteq 1 \to 1$, $Y \doteq X \to X$)

The target

- Call-by-value λ-calculus with pairs:

$$\begin{aligned} V &::= x \mid \langle\rangle \mid \lambda x.M \mid \langle U, V\rangle \\ M &::= V \mid MN \mid \langle M, N\rangle \mid \pi_1 M \mid \pi_2 M \\ A &::= 1 \mid X \mid A \to B \mid A \times B \\ \Psi &::= X_1 \doteq A_1, \ldots, X_k \doteq A_k \end{aligned}$$

- ... with systems of equations possibly definining recursive types (e.g. $X \doteq X \to X$)
- Systems are solvable if they have a solution with closed types (e.g. $X \doteq 1 \to 1, Y \doteq X \to X$)

The results

We present a translation M° from typed Λ_{reg} programs M to (resursively) typed λ-terms with pairs.

Theorem

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

Region contexts are translated into systems of equations R°, the ones used in the above translation.

Theorem

R is stratified iff R° is solvable.

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

Corollary

If R is stratified, a typed program always terminates.

The results

We present a translation M° from typed Λ_{reg} programs M to (resursively) typed λ-terms with pairs.

Theorem

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

Region contexts are translated into systems of equations R°, the ones used in the above translation.

Theorem

R is stratified iff R° is solvable.

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

Corollary

If R is stratified, a typed program always terminates.

The results

We present a translation M° from typed Λ_{reg} programs M to (resursively) typed λ-terms with pairs.

Theorem

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

Region contexts are translated into systems of equations R°, the ones used in the above translation.

Theorem

R is stratified iff R° is solvable.

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

Corollary

If R is stratified, a typed program always terminates.

Outline

1. The λ-calculus with regions
   - Syntax and evaluation
   - Typing and stratification
2. Towards ordinary λ-calculus via monads
   - The result
   - Localized state monads
   - The translation
3. Conclusion

Monads

- In category theory, monads are endofunctors T with natural transformations

  $$\nu_A : A \to TA, \qquad \mu_A : TTA \to TA$$

  with some commuting diagrams.
- In ccc's: strong monads have $s_{A,B} : TA \times B \to T(A \times B)$.
- Strong monads can be used to elegantly encapsulate side effects in a pure type system (as done in Haskell).

Eugenio Moggi.
Notions of computation and monads.
Information and Computation, 93(1):55–92, July 1991.

Values and computations

- Base idea: if A is the type of values, TA is the type of computations with side effects yielding values of type A.
- Depending on the monad, particular ops will be available, but all have the following.
- Unit$\rightsquigarrow \dfrac{\Gamma \vdash M : A}{\Gamma \vdash [M] : TA}$ $\qquad \Gamma \xrightarrow{M} A \xrightarrow{\nu} TA$
- Multiplication+functor+strength$\rightsquigarrow \dfrac{\Gamma \vdash M : TA \qquad x : A, \Gamma \vdash N : TB}{\Gamma \vdash \mathtt{let}\ x\ \mathtt{be}\ M\ \mathtt{in}\ N : TB}$

$$\Gamma \xrightarrow{\Delta} \Gamma \times \Gamma \xrightarrow{M \times 1} TA \times \Gamma \xrightarrow{s} T(A \times \Gamma) \xrightarrow{TN} TTB \xrightarrow{\mu} TB$$

Values and computations

- Base idea: if A is the type of values, TA is the type of computations with side effects yielding values of type A.
- Depending on the monad, particular ops will be available, but all have the following.
- Unit⇝ $\dfrac{\Gamma \vdash M : A}{\Gamma \vdash [M] : TA}$ $\quad \Gamma \xrightarrow{M} A \xrightarrow{\nu} TA$
- M

Values can be seen as computations
they are injected with no side effects.

$: TB$

TB

$$\Gamma \xrightarrow{\Delta} \Gamma \times \Gamma \xrightarrow{M \times 1} TA \times \Gamma \xrightarrow{s} T(A \times \Gamma) \xrightarrow{TN} TTB \xrightarrow{\mu} TB$$

Values and computations

- Base idea: if A is the type of values, TA is the type of computations with side effects yielding values of type A.
- Depending on the monad, particular ops will be available, but all have the following.
- Unit$\rightsquigarrow \dfrac{\Gamma \vdash M : A}{\Gamma \vdash [M] : TA}$ $\quad \Gamma \xrightarrow{M} A \xrightarrow{\nu} TA$
- Multiplication+functor+strength$\rightsquigarrow \dfrac{\Gamma \vdash M : TA \qquad x : A, \Gamma \vdash N : TB}{\Gamma \vdash \mathtt{let}\ x\ \mathtt{be}\ M\ \mathtt{in}\ N : TB}$

$$\Gamma \xrightarrow{\Delta} \Gamma \times \Gamma \xrightarrow{M \times 1} TA \times \Gamma \xrightarrow{s} T(A \times \Gamma) \xrightarrow{TN} TTB \xrightarrow{\mu} TB$$

Computations are composed via `let`, extracting value for x and mixing effects (recipe left to implementation/multiplication)

State monads

- State monads $TA = S \to (S \times A)$ encapsulate memory access (i.e. a computation takes a state and gives one back along with the value).
- If states are expressible in λ-calculus, state monads can be internalized.
- Injection and let:

$$[M] = \lambda s.\langle s, M\rangle \qquad \mathtt{let}\ x\ \mathtt{be}\ M\ \mathtt{in}\ N = \lambda s.(\lambda\langle t, x\rangle.Nt)(Ms)$$

($\lambda\langle x, y\rangle.M$ short for $\lambda p.(\lambda x, y.M)(\pi_1 p)(\pi_2 p)$)

- Typical operations available:
  - $\mathtt{g} : TS$, reading the state ($\mathtt{g} = \lambda s.\langle s, s\rangle$);
  - $\mathtt{s} : S \to T1$, writing the state ($\mathtt{s} = \lambda s.\lambda t.\langle s, \langle\rangle\rangle$);
  - $\mathtt{run} : S \to TA \to A$, running a computation with some initial state ($\mathtt{run} = \lambda s, c.\pi_2(cs)$).

State monads

- State monads $TA = S \to (S \times A)$ encapsulate memory access (i.e. a computation takes a state and gives one back along with the value).
- If states are expressible in λ-calculus, state monads can be internalized.
- Injection and let:

$$[M] = \lambda s.\,\langle s, M\rangle \qquad \texttt{let}\ x\ \texttt{be}\ M\ \texttt{in}\ N = \lambda s.(\lambda\langle t, x\rangle.Nt)(Ms)$$

($\lambda\langle x, y\rangle.M$ short for $\lambda p.(\lambda x, y.M)(\pi_1 p)(\pi_2 p)$)

- Typical operations available:
  - $\mathtt{g} : TS$, reading the state ($\mathtt{g} = \lambda s.\langle s, s\rangle$);
  - $\mathtt{s} : S \to T1$, writing the state ($\mathtt{s} = \lambda s.\lambda t.\langle s, \langle\rangle\rangle$);
  - $\mathtt{run} : S \to TA \to A$, running a computation with some initial state ($\mathtt{run} = \lambda s, c.\pi_2(cs)$).

State monads

- State monads $TA = S \to (S \times A)$ encapsulate memory access (i.e. a computation takes a state and gives one back along with the value).
- If states are expressible in λ-calculus, state monads can be internalized.
- Injection and let:

$$[M] = \lambda s.\,\langle s, M\rangle \qquad \mathtt{let}\; x\; \mathtt{be}\; M\; \mathtt{in}\; N = \lambda s.(\lambda\langle t, x\rangle.Nt)(Ms)$$

($\lambda\langle x, y\rangle.M$ short for $\lambda p.(\lambda x, y.M)(\pi_1 p)(\pi_2 p)$)

- Typical operations available:
  - $\mathtt{g} : TS$, reading the state ($\mathtt{g} = \lambda s.\langle s, s\rangle$);
  - $\mathtt{s} : S \to T1$, writing the state ($\mathtt{s} = \lambda s.\lambda t.\langle s, \langle\rangle\rangle$);
  - $\mathtt{run} : S \to TA \to A$, running a computation with some initial state ($\mathtt{run} = \lambda s, c.\pi_2(cs)$).

Localized monads

- For every region r we give a type variable X_r.
- For every set of regions e we give the type $P_e = \prod_{r \in e} X_r$ (canonical order on regions needed).
- The state monad T_e localized at e is $T_e A = P_e \to (P_e \times A)$.
- Usual implementation, with the types we will use:

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash [M] : T_\emptyset A} \qquad \frac{\Gamma \vdash M : T_e A \qquad x : A, \Gamma \vdash N : T_e B}{\Gamma \vdash \mathtt{let}\ x\ \mathtt{be}\ M\ \mathtt{in}\ N : T_e B}$$

$$\vdash \mathtt{g} : T_{\{r\}} X_r \qquad \vdash \mathtt{s} : X_r \to T_{\{r\}} 1 \qquad \vdash \mathtt{run} : 1 \to T_\emptyset A \to A.$$

Mixing monads

- There is a λ-term $\mathtt{coer}_{e,f} : T_e A \to T_{e \cup f} A$ adding dummy effects.

$$\mathtt{coer}_{e,f} = \lambda c.\lambda s.(\lambda\langle t, v\rangle.\langle s|_{f \setminus e} + t, v\rangle)(Ms|_e)$$

$$\left(\begin{array}{l} s|_e\text{: projections and pairings to restrict state to } e \\ s + t\text{: on disjoint states joins them together} \end{array} \right)$$

- Then computations with different effects can be mixed:

$$\dfrac{\dfrac{\Gamma \vdash M : T_e A}{\Gamma \vdash \mathtt{coer}_{e,f}\, M : T_{e \cup f} A} \qquad \dfrac{x : A, \Gamma \vdash N : T_f B}{x : A, \Gamma \vdash \mathtt{coer}_{f,e}\, N : T_{e \cup f} B}}{\Gamma \vdash \mathtt{let}\ x\ \mathtt{be}\ (\mathtt{coer}_{e,f}\, M)\ \mathtt{in}\ (\mathtt{coer}_{f,e}\, N) : T_{e \cup f} B}$$

Mixing monads

- There is a λ-term $\mathtt{coer}_{e,f} : T_e A \to T_{e \cup f} A$ adding dummy effects.

$$\mathtt{coer}_{e,f} = \lambda c.\lambda s.(\lambda\langle t, v\rangle.\langle s|_{f \setminus e} + t, v\rangle)(Ms|_e)$$

$$\left(\begin{array}{l} s|_e\text{: projections and pairings to restrict state to } e \\ s + t\text{: on disjoint states joins them together} \end{array}\right)$$

- Then computations with different effects can be mixed:

$$\frac{\Gamma \vdash M : T_e A \qquad x : A, \Gamma \vdash N : T_f B}{\Gamma \vdash \mathtt{let}_{e,f}\, x\, \mathtt{be}\, M\, \mathtt{in}\, N : T_{e \cup f} B}$$

Allocation and deallocation

The particular way in which νr works represents in fact a sort of partial run:

$$\begin{aligned} n_r^e &: X_r \to T_e A \to T_{e\setminus\{r\}} A, \\ n_r^e &= \lambda v, c.\lambda s.(\lambda\langle t, u\rangle.\langle t|_{e\setminus\{r\}}, u\rangle)(c(s+v)) \end{aligned}$$

n_r^e runs c adding value v and extracting “sub-computation”.

Outline

1. The λ-calculus with regions
   - Syntax and evaluation
   - Typing and stratification
2. Towards ordinary λ-calculus via monads
   - The result
   - Localized state monads
   - The translation
3. Conclusion

Translation of types

Types:

$$1^\circ = 1 \qquad (A \xrightarrow{e} B)^\circ = A^\circ \to T_e B^\circ$$

(arrows annotated with e become arrows of the Kleisli category for T_e)

Region contexts:

$$(r_1 : A_1, \dots, r_k : A_k)^\circ = (X_{r_1} \doteq A_1^\circ, \dots, X_{r_k} \doteq A_k^\circ)$$

For example, the auto-referential region context $r : 1 \xrightarrow{\{r\}} A$ gives the recursive type $X_r \doteq 1 \to X_r \to (X_r \times A^\circ)$.

Theorem

R is stratified iff R° is solvable (i.e. no true recursion).

Translation of types

Types:

$$1^\circ = 1 \qquad (A \xrightarrow{e} B)^\circ = A^\circ \to T_e B^\circ$$

(arrows annotated with e become arrows of the Kleisli category for T_e)

Region contexts:

$$(r_1 : A_1, \ldots, r_k : A_k)^\circ = (X_{r_1} \doteq A_1^\circ, \ldots, X_{r_k} \doteq A_k^\circ)$$

For example, the auto-referential region context $r : 1 \xrightarrow{\{r\}} A$ gives the recursive type $X_r \doteq 1 \to X_r \to (X_r \times A^\circ)$.

Theorem

R is stratified iff R° is solvable (i.e. no true recursion).

Translation of terms

$$R; \overrightarrow{x_i : A_i} \vdash M : A, e \quad \longmapsto \quad \overrightarrow{x_i : A_i^\circ} \vdash M^\circ : T_e A^\circ \text{ modulo } R^\circ .$$

Identifying a term with its type inference:

$$\langle\rangle^* = \langle\rangle, \quad x^* = x, \quad (\lambda x.M)^* = \lambda x.M^\circ, \quad V^\circ = [V^*]$$

$$(MN)^\circ = \mathtt{let}\ f\ \mathtt{be}\ M^\circ\ \mathtt{in}\ \mathtt{let}\ a\ \mathtt{be}\ N^\circ\ \mathtt{in}\ fa$$

$$(\nu r \Leftarrow M.N)^\circ = \mathtt{let}\ v\ \mathtt{be}\ M^\circ\ \mathtt{in}\ \mathtt{n}\ vN^\circ$$

$$(\mathtt{set}(r, M))^\circ = \mathtt{let}\ v\ \mathtt{be}\ M^\circ\ \mathtt{in}\ \mathtt{s}\ v, \qquad (\mathtt{get}(r))^\circ = \mathtt{g}$$

Dummy effects are added with `coer`.

Translation of terms

$$R; \overrightarrow{x_i : A_i} \vdash M : A, e \quad \longmapsto \quad \overrightarrow{x_i : A_i^\circ} \vdash M^\circ : T_e A^\circ \text{ modulo } R^\circ .$$

Identifying a term with its type inference:

$$\langle\rangle^* = \langle\rangle, \quad x^* = x, \quad (\lambda x.M)^* = \lambda x.M^\circ, \quad V^\circ = [V^*]$$

$$(MN)^\circ = \mathtt{let}\ f\ \text{be}\ M^\circ\ \mathtt{in}\ \mathtt{let}\ a\ \text{be}\ N^\circ\ \mathtt{in}\ fa$$

$$(\nu r{\Leftarrow}M.N)^\circ = \mathtt{let}\ v\ \text{be}\ M^\circ\ \mathtt{in}\ \mathtt{n}\ vN^\circ$$

$$(\mathtt{set}(r, M))^\circ = \mathtt{let}\ v\ \text{be}\ M^\circ\ \mathtt{in}\ \mathtt{s}\ v, \qquad (\mathtt{get}(r))^\circ = \mathtt{g}$$

Dummy effects are added with `coer`.

Translation of terms

$$R; \overrightarrow{x_i : A_i} \vdash M : A, e \quad \longmapsto \quad \overrightarrow{x_i : A_i^\circ} \vdash M^\circ : T_e A^\circ \text{ modulo } R^\circ.$$

Identifying a term with its type inference:

$$\langle\rangle^* = \langle\rangle\,, \quad x^* = x, \quad (\lambda x.M)^* = \lambda x.M^\circ, \quad V^\circ = [V^*]$$

Values are translated as monad-free values
When used as computations they are just injected

$$(\nu r \Leftarrow M.N) = \mathtt{let}\ v\ \mathtt{be}\ M\ \mathtt{in\ n}\ vN$$

$$(\mathtt{set}(r, M))^\circ = \mathtt{let}\ v\ \mathtt{be}\ M^\circ\ \mathtt{in\ s}\ v, \qquad (\mathtt{get}(r))^\circ = \mathtt{g}$$

Dummy effects are added with `coer`.

Translation of terms

$$R; \overrightarrow{x_i : A_i} \vdash M : A, e \quad \longmapsto \quad \overrightarrow{x_i : A_i^\circ} \vdash M^\circ : T_e A^\circ \text{ modulo } R^\circ.$$

Identifying a term with its type inference:

$$\langle\rangle^* = \langle\rangle\,, \quad x^* = x, \quad (\lambda x.M)^* = \lambda x.M^\circ, \quad V^\circ = [V^*]$$

$$(MN)^\circ = \mathtt{let}\ f\ \mathtt{be}\ M^\circ\ \mathtt{in}\ \mathtt{let}\ a\ \mathtt{be}\ N^\circ\ \mathtt{in}\ fa$$

`let` is used to merge effects in an otherwise simple application
(we hided explicit effects for $M{:}A \xrightarrow{e_3} B, e_1$ and $N{:}A, e_2$)

$$(\mathtt{set}(r, M))^\circ = \mathtt{let}\ v\ \mathtt{be}\ M^\circ\ \mathtt{in}\ s\,v, \qquad (\mathtt{get}(r))^\circ = g$$

Dummy effects are added with `coer`.

Translation of terms

$$R; \overrightarrow{x_i : A_i} \vdash M : A, e \quad \longmapsto \quad \overrightarrow{x_i : A_i^\circ} \vdash M^\circ : T_e A^\circ \text{ modulo } R^\circ.$$

Identifying a term with its type inference:

$$\langle\rangle^* = \langle\rangle, \quad x^* = x, \quad (\lambda x.M)^* = \lambda x.M^\circ, \quad V^\circ = [V^*]$$

$$(MN)^\circ = \mathtt{let}_{e_1, e_2 \cup e_3}\, f\, \mathtt{be}\, M^\circ\, \mathtt{in}\, \mathtt{let}_{e_2, e_3}\, a\, \mathtt{be}\, N^\circ\, \mathtt{in}\, fa$$

`let` is used to merge effects in an otherwise simple application (we hided explicit effects for $M{:}A \xrightarrow{e_3} B, e_1$ and $N{:}A, e_2$)

Dummy effects are added with `coer`.

Translation of terms

$$R; \overrightarrow{x_i : A_i} \vdash M : A, e \quad \longmapsto \quad \overrightarrow{x_i : A_i^\circ} \vdash M^\circ : T_e A^\circ \text{ modulo } R^\circ.$$

Identifying a term with its type inference:

$$\langle\rangle^* = \langle\rangle\,, \quad x^* = x, \quad (\lambda x.M)^* = \lambda x.M^\circ, \quad V^\circ = [V^*]$$

$$(MN)^\circ = \mathtt{let}_{e_1, e_2 \cup e_3}\, f\ \mathtt{be}\ M^\circ\ \mathtt{in}\ \mathtt{let}_{e_2, e_3}\, a\ \mathtt{be}\ N^\circ\ \mathtt{in}\ fa$$

$$(\nu r{\Leftarrow}M.N)^\circ = \mathtt{let}\ v\ \mathtt{be}\ M^\circ\ \mathtt{in}\ \mathtt{n}\ vN^\circ$$

`n` is used to introduce regions (*M*'s effects merged with `let`)
(we hided explicit effects for $M{:}A, e_1$ and $N{:}B, e_2$)

Dummy effects are added with `coer`.

Translation of terms

$$R; \overrightarrow{x_i : A_i} \vdash M : A, e \quad \longmapsto \quad \overrightarrow{x_i : A_i^\circ} \vdash M^\circ : T_e A^\circ \text{ modulo } R^\circ.$$

Identifying a term with its type inference:

$$\langle\rangle^* = \langle\rangle, \quad x^* = x, \quad (\lambda x.M)^* = \lambda x.M^\circ, \quad V^\circ = [V^*]$$

$$(MN)^\circ = \mathtt{let}_{e_1, e_2 \cup e_3}\, f \,\mathtt{be}\, M^\circ \,\mathtt{in}\, \mathtt{let}_{e_2, e_3}\, a \,\mathtt{be}\, N^\circ \,\mathtt{in}\, fa$$

$$(\nu r \Leftarrow M.N)^\circ = \mathtt{let}_{e_1, e_2 \setminus \{r\}}\, v \,\mathtt{be}\, M^\circ \,\mathtt{in}\, \mathtt{n}_r^{e_2}\, vN^\circ$$

`n` is used to introduce regions (M's effects merged with `let`)
(we hided explicit effects for $M{:}A, e_1$ and $N{:}B, e_2$)

Dummy effects are added with `coer`.

Translation of terms

$$R; \overrightarrow{x_i : A_i} \vdash M : A, e \quad \longmapsto \quad \overrightarrow{x_i : A_i^\circ} \vdash M^\circ : T_e A^\circ \text{ modulo } R^\circ.$$

Identifying a term with its type inference:

$$\langle\rangle^* = \langle\rangle, \quad x^* = x, \quad (\lambda x.M)^* = \lambda x.M^\circ, \quad V^\circ = [V^*]$$

$$(MN)^\circ = \mathtt{let}_{e_1,e_2\cup e_3} f \,\mathtt{be}\, M^\circ \,\mathtt{in}\, \mathtt{let}_{e_2,e_3} a \,\mathtt{be}\, N^\circ \,\mathtt{in}\, fa$$

$$(\nu r \Leftarrow M.N)^\circ = \mathtt{let}_{e_1,e_2\setminus\{r\}} v \,\mathtt{be}\, M^\circ \,\mathtt{in}\, \mathtt{n}_r^{e_2} vN^\circ$$

$$(\mathtt{set}(r,M))^\circ = \mathtt{let}\, v \,\mathtt{be}\, M^\circ \,\mathtt{in}\, \mathtt{s}\, v, \qquad (\mathtt{get}(r))^\circ = \mathtt{g}$$

Du

`s` and `g` used to affect regions (*M*'s effects merged with `let`)
types indicate what region is affected

Translation of terms

$$R; \overrightarrow{x_i : A_i} \vdash M : A, e \quad \longmapsto \quad \overrightarrow{x_i : A_i^\circ} \vdash M^\circ : T_e A^\circ \text{ modulo } R^\circ.$$

Identifying a term with its type inference:

$$\langle\rangle^* = \langle\rangle, \quad x^* = x, \quad (\lambda x.M)^* = \lambda x.M^\circ, \quad V^\circ = [V^*]$$

$$(MN)^\circ = \mathtt{let}_{e_1, e_2 \cup e_3} f \,\mathtt{be}\, M^\circ \,\mathtt{in}\, \mathtt{let}_{e_2, e_3} a \,\mathtt{be}\, N^\circ \,\mathtt{in}\, fa$$

$$(\nu r \Leftarrow M.N)^\circ = \mathtt{let}_{e_1, e_2 \setminus \{r\}} v \,\mathtt{be}\, M^\circ \,\mathtt{in}\, \mathtt{n}_r^{e_2} \, vN^\circ$$

$$(\mathtt{set}(r, M))^\circ = \mathtt{let}_{e, \{r\}} v \,\mathtt{be}\, M^\circ \,\mathtt{in}\, \mathtt{s}\, v, \qquad (\mathtt{get}(r))^\circ = \mathtt{g}$$

Du

`s` and `g` used to affect regions (M's effects merged with `let`)
types indicate what region is affected

Translation of terms

$$R; \overrightarrow{x_i : A_i} \vdash M : A, e \quad \longmapsto \quad \overrightarrow{x_i : A_i^\circ} \vdash M^\circ : T_e A^\circ \text{ modulo } R^\circ .$$

Identifying a term with its type inference:

$$\langle\rangle^* = \langle\rangle\,, \quad x^* = x, \quad (\lambda x.M)^* = \lambda x.M^\circ, \quad V^\circ = [V^*]$$

$$(MN)^\circ = \mathtt{let}_{e_1, e_2 \cup e_3}\, f \,\mathtt{be}\, M^\circ \,\mathtt{in}\, \mathtt{let}_{e_2, e_3}\, a \,\mathtt{be}\, N^\circ \,\mathtt{in}\, fa$$

$$(\nu r {\Leftarrow} M.N)^\circ = \mathtt{let}_{e_1, e_2 \setminus \{r\}}\, v \,\mathtt{be}\, M^\circ \,\mathtt{in}\, \mathtt{n}_r^{e_2}\, vN^\circ$$

$$(\mathtt{set}(r, M))^\circ = \mathtt{let}_{e, \{r\}}\, v \,\mathtt{be}\, M^\circ \,\mathtt{in}\, \mathtt{s}\, v, \qquad (\mathtt{get}(r))^\circ = \mathtt{g}$$

Dummy effects are added with `coer`.

The result, precised

We recall programs have no active effects,i.e.

$$R; \vdash M, \emptyset \implies \vdash M^\circ : T_\emptyset A^\circ .$$

Theorem

A program M evaluates to V iff $\mathtt{run}\,\langle\rangle\, M^\circ$ evaluates to V^*.

Outline

1. The λ-calculus with regions
   - Syntax and evaluation
   - Typing and stratification
2. Towards ordinary λ-calculus via monads
   - The result
   - Localized state monads
   - The translation
3. Conclusion

Polymorphism

- Type polymorphism: $\ldots \mid X \mid \forall X.A$.
- Regions are still given fixed, closed types.

$$\frac{R;\Gamma \vdash M : A, e \qquad X \notin \mathrm{FV}(\Gamma)}{R;\Gamma \vdash M : \forall X.A, e} \qquad \frac{R;\Gamma \vdash M : \forall X.A, e}{R;\Gamma \vdash M : A\{B/X\}, e}$$

- Translation and results still hold.
- Region polymorphism (e.g. a function swapping values between any two regions) to be studied.
- Probably leads to dependent types: abstraction on r binds its occurrences in types.

Polymorphism

- Type polymorphism: $\ldots \mid X \mid \forall X.A$.
- Regions are still given fixed, closed types.

$$\frac{R;\Gamma \vdash M : A, e \qquad X \notin \mathrm{FV}(\Gamma)}{R;\Gamma \vdash M : \forall X.A, e} \qquad \frac{R;\Gamma \vdash M : \forall X.A, e}{R;\Gamma \vdash M : A\{B/X\}, e}$$

- Translation and results still hold.
- Region polymorphism (e.g. a function swapping values between any two regions) to be studied.
- Probably leads to dependent types: abstraction on r binds its occurrences in types.

Exceptions and positively recursive types

- The stack nature of regions, i.e. the non-binding of νr's, is similar to exception handling in Ocaml, e.g.

  `(try (fun d -> raise Exception) with Exception -> ...)(...)`

  has uncaught exception (`try`/`with` does not bind exceptions).
- Possibly $\Lambda_{\mathrm{reg}} +$ call/cc could simulate exceptions (and provide for a translation in nets...)
- Also, positively recursive types (i.e. $X \doteq A$ with X appearing in positive position in A) preserve normalization.

  N. P. Mendler.
  Inductive types and type constraints in the second-order lambda calculus.
  Ann. Pure Appl. Logic, 51(1-2):159–172, 1991.
- Could provide finer condition than stratification, but separation of read and write access is necessary (in $T_{\{r\}}A = X_r \to (X_r \times A)$, X_r is both positive and negative).

Exceptions and positively recursive types

- The stack nature of regions, i.e. the non-binding of νr's, is similar to exception handling in Ocaml, e.g.

  `(try (fun d -> raise Exception) with Exception -> ...)(...)`

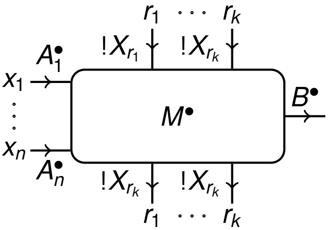
  has uncaught exception (`try`/`with` does not bind exceptions).
- Possibly $\Lambda_{\mathrm{reg}} +$ call/cc could simulate exceptions (and provide for a translation in nets. . .)
- Also, positively recursive types (i.e. $X \doteq A$ with X appearing in positive position in A) preserve normalization.

N. P. Mendler.
Inductive types and type constraints in the second-order lambda calculus.
Ann. Pure Appl. Logic, 51(1-2):159–172, 1991.

- Could provide finer condition than stratification, but separation of read and write access is necessary (in $T_{\{r\}}A = X_r \to (X_r \times A)$, X_r is both positive and negative).

In the next episode – I

We give a translation $M^\bullet$ from Λ_{reg} into LL proof nets, relying on translation into ordinary λ-calculus and the call-by-value one into LL.



Theorem

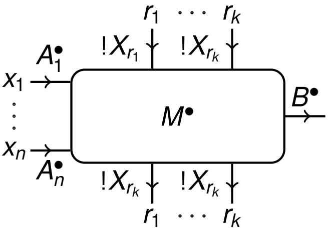
If $M \to N$ *then* $M^\bullet \xrightarrow{+} N^\bullet$.

Theorem

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

In the next episode – I

We give a translation $M^\bullet$ from Λ_{reg} into LL proof nets, relying on translation into ordinary λ-calculus and the call-by-value one into LL.



Theorem

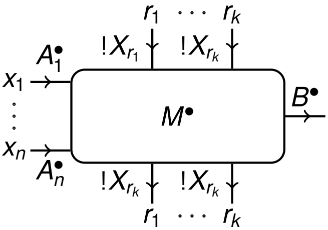
If $M \to N$ then $M^\bullet \xrightarrow{+} N^\bullet$.

Theorem

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

In the next episode – I

We give a translation $M^\bullet$ from Λ_{reg} into LL proof nets, relying on translation into ordinary λ-calculus and the call-by-value one into LL.



Theorem

If $M \to N$ *then* $M^\bullet \xrightarrow{+} N^\bullet$*.*

Theorem

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

In the next episode – II

- We will then introduce multithreading: $\ldots \mid (M|N)$
- νr's provide for inter-thread communication.
- We then show how it can also be translated using differential nets.
- We will discuss the problems with logical correctness.

Thanks

Questions?

Full subtyping

- Full subtyping can be introduced:

$$A \leq A, \qquad \frac{C \leq A \qquad e \subseteq f \qquad B \leq D}{A \xrightarrow{e} B \leq C \xrightarrow{f} D}$$

$$\frac{R; \Gamma \vdash_{\mathrm{st}} M : A, e \qquad A \leq B}{R; \Gamma \vdash_{\mathrm{st}} M : B, e}$$

- Allows for more flexibility: for example $M : (1 \xrightarrow{e} 1) \xrightarrow{e} 1$ would directly accept pure functions.

Avoiding full subtyping

- Full subtyping can be avoided at the cost of more reduntant terms.
- define $M \rightsquigarrow N$ as context closure of

$$\begin{aligned} V &\rightsquigarrow \lambda x.Vx && \text{with } x \notin V, \\ M &\rightsquigarrow (\lambda z.z)M && \text{with } M \text{ not a value.} \end{aligned}$$

i.e. a combination of η- and β-expansions “acceptable” in call-by-value.

Proposition

- *If $M \overset{*}{\rightsquigarrow} N$, then M evaluates to V iff N evaluates to U with $V \overset{*}{\rightsquigarrow} U$.*
- *If $R; \Gamma \vdash_{\mathrm{st}} M : A, e$ then $\exists N$ with $M \overset{*}{\rightsquigarrow} N$ and $R; \Gamma \vdash N : A, e$ without sybtyping.*

(back)