



λ_S : A Lambda Calculus with Side-effects

Arvind
delivered by Jacob Schwartz
Laboratory for Computer Science
M.I.T.

October 30, 2002

<http://www.csg.lcs.mit.edu/6.827>

M-Structures and Barriers

- Some problems cannot be expressed functionally
 - Input / Output
 - Gensym: Generate unique identifiers
 - Gathering statistics
 - Graph algorithms
 - Non-deterministic algorithms
- Once side-effects are introduced, barriers are needed to control the execution of some operations
- The λ_S calculus
 - λ_C + side-effects and barriers



The λ_B Calculus : λ_C + Barriers

- Even adding barriers to a purely functional calculus (without side-effects) is significant
 - Observability of Termination
- Using λ_B as a stepping stone to λ_S allows us to analyze the semantic effects of barriers separate from side-effects, simplifying the analysis
 - $\lambda_S = \lambda_B + \text{side-effects}$



Outline

- Background
- The λ_C calculus: λ + letrecs
- Observable values
- The λ_B calculus: λ_C + barriers
- Garbage collection
- The λ_S calculus: λ_B + side-effects



λ + Let : A way to model sharing

Instead of the normal β -rule

$$(\lambda x.e) e_a \Rightarrow e [e_a/x]$$

use the following β_{let} rule

$$(\lambda x.e) e_a \Rightarrow \{ \text{let } t = e_a \text{ in } e[t/x] \}$$

where t is a new variable

and only allow the substitution of *values*
and *variables* to preserve sharing



Previous work on Sharing

Differences are mainly regarding

- where variables can be instantiated
- the source language

λ
or $\lambda + let$
or $\lambda + letrec$

- Graph reduction and lazy evaluation
Wadsworth (71), Launchbury (POPL93)
- Environments and Explicit Substitution
Abadi, Cardelli, Curien & Levy (POPL 92, JFP)
- Letrecs but no reductions inside λ -abstractions
Ariola, Felleisen, Wadler, ... (POPL 95)
- Letrecs
Ariola et al. (96)



λ_C Syntax

$$E ::= x \mid \lambda x.E \mid E E \mid \{ S \text{ in } E \}$$

$$\mid \text{Cond}(E, E, E)$$

$$\mid \text{PF}_k(E_1, \dots, E_k)$$

$$\mid \text{CN}_0 \mid \text{CN}_k(E_1, \dots, E_k) \mid \underline{\text{CN}}_k(\text{SE}_1, \dots, \text{SE}_k)$$

$$\text{PF}_1 ::= \text{negate} \mid \text{not} \mid \dots \mid \text{Prj}_1 \mid \text{Prj}_2 \mid \dots$$

...

$$\text{CN}_0 ::= \text{Number} \mid \text{Boolean}$$

$$\text{CN}_2 ::= \text{Cons} \mid \dots$$

$$S ::= \varepsilon \mid x = E \mid S; S$$

Not in initial expressions



λ_C Syntax

Values

$$V ::= \lambda x.E \mid \text{CN}_0 \mid \underline{\text{CN}}_k(\text{SE}_1, \dots, \text{SE}_k)$$

Simple expressions

$$\text{SE} ::= x \mid V$$


Equivalence Rules

- α -renaming

$$\lambda x.e \equiv \lambda x'.(e[x'/x])$$

$$\{x=e; S \text{ in } e_0\} \equiv \{x'=e; S \text{ in } e_0\}[x'/x]$$

- Properties of ";"

$$\varepsilon ; S \equiv S$$

$$S_1 ; S_2 \equiv S_2 ; S_1$$

$$S_1 ; (S_2 ; S_3) \equiv (S_1 ; S_2) ; S_3$$



λ_{let} Instantiation Rules

a is a Simple Expression;

[x] is a free occurrence of x in C[x] or SC[x]

- Instantiation Rule 1

$$\{x = a ; S \text{ in } C[x]\} \Rightarrow \{x = a ; S \text{ in } C'[a]\}$$

- Instantiation Rule 2

$$(x = a ; SC[x]) \Rightarrow (x = a ; SC'[a])$$

- Instantiation Rule 3

$$x = C[x] \Rightarrow x = C'[C[x]]$$

where C[x] is simple



λ_C Rules

- *Cond-rules*

$$\text{Cond}(\text{True}, e_1, e_2) \Rightarrow e_1$$

$$\text{Cond}(\text{False}, e_1, e_2) \Rightarrow e_2$$
- *Constructors*

$$\text{CN}_k(e_1, \dots, e_k) \Rightarrow \{t_1 = e_1 ; \dots ; t_k = e_k \text{ in } \underline{\text{CN}}_k(t_1, \dots, t_k)\}$$
- *δ -rules*

$$\text{PF}_k(v_1, \dots, v_k) \Rightarrow \text{pf}_k(v_1, \dots, v_k)$$

$$\text{Prj}_i(\underline{\text{CN}}_k(x_1, \dots, x_i, \dots, x_k)) \Rightarrow x_i$$



Need for Lifting Rules

$$\{ f = \{ S_1 \text{ in } \lambda x. e_1 \};$$

$$y = f a ;$$

$$\text{in}$$

$$(\{ S_2 \text{ in } \lambda x. e_2 \} e_3) \}$$

How do we juxtapose

or

$$(\lambda x. e_1) a$$

$$(\lambda x. e_2) e_3 \quad ?$$


λ_C Block Flattening and Lifting Rules

- *Block Flatten*

$$x = \{ S \text{ in } e \} \Rightarrow (x = e' ; S')$$

- *Lifting rules*

$$\{ S_1 \text{ in } \{ S_2 \text{ in } e \} \} \Rightarrow \{ S_1 ; S'_2 \text{ in } e' \}$$

$$\{ S \text{ in } e \} e_2 \Rightarrow \{ S' \text{ in } e' e_2 \}$$

$$\text{Cond}(\{ S \text{ in } e \}, e_1, e_2) \Rightarrow \{ S' \text{ in } \text{Cond}(e', e_1, e_2) \}$$

$$\text{PF}_k(e_1, \dots, \{ S \text{ in } e \}, \dots, e_k) \Rightarrow \{ S' \text{ in } \text{PF}_k(e_1, \dots, e', \dots, e_k) \}$$

$\{ S' \text{ in } e' \}$ is the α -renaming of $\{ S \text{ in } e \}$ to avoid name conflicts



Non-confluence

$$\text{odd} = \lambda n. \text{Cond}(n=0, \text{False}, \text{even}(n-1)) \quad \text{---- } (M)$$

$$\text{even} = \lambda n. \text{Cond}(n=0, \text{True}, \text{odd}(n-1))$$

substitute for even (n-1) in M

$$\text{odd} = \lambda n. \text{Cond}(n=0, \text{False}, \text{Cond}(n-1 = 0, \text{True}, \text{odd}((n-1)-1))) \quad \text{---- } (M_1)$$

$$\text{even} = \lambda n. \text{Cond}(n=0, \text{True}, \text{odd}(n-1))$$

substitute for odd (n-1) in M

$$\text{odd} = \lambda n. \text{Cond}(n=0, \text{False}, \text{even}(n-1)) \quad \text{---- } (M_2)$$

$$\text{even} = \lambda n. \text{Cond}(n=0, \text{True}, \text{Cond}(n-1 = 0, \text{False}, \text{even}((n-1)-1)))$$

M₁ and M₂ cannot be reduced to the same expression!
Ariola & Klop (LICS 94)



Printable Values

Printable values are trees and can be infinite

We will compute the printable value of a term in 2 steps:

Info: $E \rightarrow T_p$ (trees)
 Print: $E \rightarrow \{T_p\}$
 (downward closed sets of trees)

where

$T_p ::= \perp \mid \lambda \mid CN_0 \mid CN_k(T_{p1}, \dots, T_{pk})$

$\perp \leq t$ (*bottom*)
 $t \leq t$ (*reflexive*)
 $CN_k(v_1, \dots, v_i, \dots, v_k) \leq CN_k(v_1, \dots, v'_i, \dots, v_k)$
 if $v_i \leq v'_i$



Info Procedure

Info : $E \rightarrow T_p$

Info [{ S in E }] = Info [E]
 Info [$\lambda x.E$] = λ
 Info [CN_0] = CN_0
 Info [$CN_k(a_1, \dots, a_k)$] = $CN_k(\text{Info}[a_1], \dots, \text{Info}[a_k])$
 Info [E] = Ω
otherwise

Proposition Reduction is monotonic wrt Info:
 If $e \rightarrow e_1$ then $\text{Info}[e] \leq \text{Info}[e_1]$.

Proposition Confluence wrt Info:
 If $e \rightarrow e_1$ and $e \rightarrow e_2$ then
 $\exists e_3$ s.t. $e_1 \rightarrow e_3$ and $\text{Info}[e_2] \leq \text{Info}[e_3]$.



Print Procedure

Print : $E \dashrightarrow \{T_P\}$

$\text{Print}[e] = \{ i \mid i \leq \text{Info}[e_1] \text{ and } e \dashrightarrow e_1 \}$.

\dashrightarrow is simple instantiation:

$\text{let } x = v ; S \text{ in } C[x] \dashrightarrow \text{let } x = v ; S \text{ in } C[v]$

Unwind the value as much as possible
Keep track of all the unwindings

Terms with infinite unwindings lead to infinite sets.



Print*: Maximum Printable Info

$\text{Print}^*[e] = \{ \bigcup_i \text{Print}[s_i] \mid s \in \text{PRS}(e) \}$

where

Definition: Reduction Sequence

$\text{RS}(e) = \{ s \mid s_0 = e, s_{i-1} \dashrightarrow s_i, 0 < i < |s| \}$

Definition: Progressive Reduction Sequence

$\text{PRS}(e) = \{ s \mid s \in \text{RS}(e), \text{ and } \exists i \forall j > i. s_j \dashrightarrow t \Rightarrow \exists k. \text{Print}[t] \leq \text{Print}[s_k] \}$

Proposition:

if $e \dashrightarrow e_1$ then $\text{Print}^*[e] = \text{Print}^*[e_1]$.
 $\text{Print}^*[e]$ has precisely one element.



λ_B Syntax

$$E ::= x \mid \lambda x.E \mid E E \mid \{ S \text{ in } E \}$$

$$\mid \text{Cond}(E, E, E)$$

$$\mid \text{PF}_k(E_1, \dots, E_k)$$

$$\mid \text{CN}_0 \mid \text{CN}_k(E_1, \dots, E_k) \mid \underline{\text{CN}}_k(x_1, \dots, x_k)$$

$$\text{PF}_1 ::= \text{negate} \mid \text{not} \mid \dots \mid \text{Prj}_1 \mid \text{Prj}_2 \mid \dots$$

$$\dots$$

$$\text{CN}_0 ::= \text{Number} \mid \text{Boolean}$$

$$S ::= \varepsilon \mid x = E \mid S; S \mid S \gg S$$

Not in initial expressions



Barriers

$$\{ (y = 1+7 \gg \gg z = 3) \text{ in } z \} \Rightarrow \{ (y = 8 \gg \gg z = 3) \text{ in } z \} \Rightarrow \{ y = 8; (z = 3) \text{ in } z \} \Rightarrow \{ y = 8; z = 3 \text{ in } 3 \}$$

Barriers discharge when all the bindings in the pre-region *terminate*, i.e., all expressions become *values*.



Stability and Termination

Definition: Expression e is said to be stable if
when $e \rightarrow e_1$, $\text{Print}[e] = \text{Print}[e_1]$

In general, an expression cannot be tested for stability.

Terminated Terms

$$E^T ::= V \mid \{H \text{ in } SE\}$$

$$H ::= x = V \mid H; H$$

Proposition: All terminated terms are stable.



Values and Heap Terms

Values

$$V ::= \lambda x. E \mid \text{CN}_0 \mid \underline{\text{CN}}_k(x_1, \dots, x_k)$$

Simple expressions

$$SE ::= x \mid V$$

Terminated Terms

$$E^T ::= V \mid \{H \text{ in } SE\}$$

$$H ::= x = V \mid H; H$$


Barrier Rules

- *Barrier discharge*
 $(\varepsilon \gg \gg S) \Rightarrow S$
- *Barrier equivalence*
 $((H ; S_1) \gg \gg S_2) \equiv (H ; (S_1 \gg \gg S_2))$

$(H \gg \gg S) \Rightarrow (H ; S)$ (*derivable*)



λ_C Versus λ_B

In λ_B termination of a term is observable.
 Thus,

$$5 \neq \{x = \perp \text{ in } 5\}$$

Consider the context:

$$\{ (y = [\blacksquare]) \\ \gg \gg \\ z = 3) \\ \text{in} \\ z \}$$

\Rightarrow equality in λ_C does not imply equality in λ_B

However, barriers can only make a term less defined.



Properties of λ_B

Proposition Barriers are associative:

$S1 \ggg (S2 \ggg S3) = (S1 \ggg S2) \ggg S3$
in all contexts.

Proposition Barriers reduce results:

Every reduction in $C[S1 \ggg S2]$ can be modeled by a reduction in $C[S1 ; S2]$.

Proposition Postregions can be postponed:

If $C1[S1 \ggg S2] \rightarrow C3[S3 \ggg S4]$ where the barrier is the same in both terms, there is a $C2$ such that:

$C1[S1 \ggg S2] \rightarrow C2[S3 \ggg S2] \rightarrow C3[S3 \ggg S4]$



Garbage Collection

A Garbage collection rule *erases* part of a term.

Definition:

A garbage collection rule, GC , is said to be *correct* if for all e , $\text{Print}^*(e) = \text{Print}^*(GC(e))$



λ_B Garbage Collection Rule

GC₀-rule
 $\{ S_G ; S \text{ in } e \} \Rightarrow \{ S \text{ in } e \}$
 if forall $x, x \in (FV(e) \cup FV(S))$ then $x \notin BV(S_G)$

GC_v-rule
 $\{ H ; S \text{ in } e \} \Rightarrow \{ S \text{ in } e \}$
 if forall $x, x \in (FV(e) \cup FV(S))$ then $x \notin BV(H)$

While both GC_0 and GC_v rules are correct for λ_{let} ,
 only the GC_v -rule is correct for λ_B .



λ_S Syntax

$E ::= x \mid \lambda x.E \mid EE \mid \{ S \text{ in } E \}$
 $\mid \text{Cond}(E, E, E)$
 $\mid PF_k(E_1, \dots, E_k)$
 $\mid CN_0 \mid CN_k(E_1, \dots, E_k) \mid \underline{CN}_k(x_1, \dots, x_k)$
 $\mid \text{allocate}()$
 $\mid o_i$ object descriptors

$PF_1 ::= \text{negate} \mid \text{not} \mid \dots \mid \text{Prj}_1 \mid \text{Prj}_2 \mid \dots \mid \text{ifetch} \mid \text{mfetch}$

\dots
 $CN_0 ::= \text{Number} \mid \text{Boolean} \mid ()$

$S ::= \varepsilon \mid x = E \mid S; S$
 $\mid S \gg S$
 $\mid \text{sstore}(E, E)$
 $\mid \text{allocator} \mid \text{empty}(o_i) \mid \text{full}(o_i, E) \mid \text{error}(o_i)$

Not in initial expressions



Values and Heap Terms

Values

$$V ::= \lambda x.E \mid \text{CN}_0 \mid \underline{\text{CN}}_k(x_1, \dots, x_k) \mid o_i$$

Simple expressions

$$\text{SE} ::= x \mid V$$

Heap Terms

$$H ::= x = V \mid H; H \mid \text{allocator} \\ \mid \text{empty}(o_i) \mid \text{full}(o_i, V)$$

Terminal Expressions

$$\text{ET} ::= V \mid \text{let } H \text{ in } \text{SE}$$


Side-effect Rules

- *Allocation rule*
 $(\text{allocator}; x = \text{allocate}()) \Rightarrow \text{allocator}; x = o; \text{empty}(o)$
 where o is a new object descriptor
- *Fetch and Take rules*
 $(x = \text{ifetch}(o); \text{full}(o, v)) \Rightarrow (x = v; \text{full}(o, v))$
 $(x = \text{mfetch}(o); \text{full}(o, v)) \Rightarrow (x = v; \text{empty}(o))$
- *Store rules*
 $(\text{sstore}(o, v); \text{empty}(o)) \Rightarrow \text{full}(o, v)$
 $(\text{sstore}(o, v); \text{full}(o, v')) \Rightarrow (\text{error}(o); \text{full}(o, v'))$
- *Lifting rules*
 $\text{sstore}(\{ S \text{ in } e \}, e_2) \Rightarrow (S; \text{sstore}(e, e_2))$
 $\text{sstore}(e_1, \{ S \text{ in } e \}) \Rightarrow (S; \text{sstore}(e_1, e))$



Nondeterministic Choice

```
choose =  $\lambda x.$  { m = allocate();  
                sstore(m, True);  
                ( y = mfetch(m)  
                  >>>  
                  sstore(m, False) );  
                ( z = mfetch(m)  
                  >>>  
                  sstore(m, True) )  
                in z }
```

choose 100 ?

