Weak evaluation strategies in the λ-calculus

labandalambda

May 10, 2018

In this talk we will define three basic weak evaluation strategies: *call-by-name*, *call-by-value*, and *call-by-need*, by means of three formal presentations: *small-step evaluation*, *big-step evaluation* and an *abstract machine*. We will show how to prove that the three presentations are equivalent in the case of the call-by-name strategy. For the other two strategies similar techniques may be used (although the proofs are much more complex, especially in the call-by-need case).

Contents

1 Call-by-name

1.1 Small-step evaluation

Definition 1 (Small-step call-by-name evaluation). Terms and evaluation contexts are given by:

$$\begin{array}{lccl} \text{Terms} & t & ::= & x \mid \lambda x.t \mid t\,t \\ \text{Weak head contexts} & \mathtt{H} & ::= & \Box \mid \mathtt{H}\,t \end{array}$$

The binary relation of small-step call-by-name evaluation is defined as follows:

$$\mathtt{H}\langle(\lambda x.t)\,s\rangle \rightarrow_{\mathsf{name}} \mathtt{H}\langle t\{x := s\}\rangle$$

Remark 2 (Determinism). If $t \rightarrow_{\mathsf{name}} t_1$ and $t \rightarrow_{\mathsf{name}} t_2$ then $t_1 = t_2$.

Exercise 3. Evaluate $(\lambda x.xx)(I\,I)$ using small-step call-by-name evaluation.

1.2 Big-step evaluation

Definition 4 (Big-step call-by-name evaluation). Call-by-name environments and closures are given by the following abstract syntax:

$$\begin{array}{lccl} \text{Environments} & e & ::= & \bullet \mid [x \mapsto c] : e \\ \text{Closures} & c & ::= & (t, e) \end{array}$$

Environment concatenation is written $e_1 : e_2$. We also write $\mathrm{dom}\, e$ for the set of variables defined in e, and $e(x)$ for the value of x in e.

The *big-step call-by-name evaluation* judgment $t \Downarrow^e c$, relates a source term t, an environment e mapping variables to closures, and the resulting closure c. Derivable judgments are inductively defined as follows:

$$\frac{e(x) = (t, e') \qquad t \Downarrow^{e'} c}{x \Downarrow^e c} \qquad \frac{}{\lambda x.t \Downarrow^e (\lambda x.t, e)} \qquad \frac{t \Downarrow^e (\lambda x.t', e') \qquad t' \Downarrow^{[x \mapsto (s,e)]:e'} c}{t\, s \Downarrow^e c}$$

Remark 5. If $t \Downarrow^e c$ holds, then c is of the form $(\lambda x.s, e)$.

Exercise 6. Evaluate $(\lambda x.xx)(I\, I)$ using big-step call-by-name evaluation.

Algorithm 7. A big-step call-by-name evaluator in Haskell:

```
type Id      = String
data Term    = Var Id | Lam Id Term | App Term Term
type Env     = [(Id, Closure)]
data Closure = C Term Env

eval :: Term -> Env -> Closure
eval (Var x)   e = let Just (C t e') = lookup x e in
                   eval t e'
eval (Lam x t) e = C (Lam x t) e
eval (App t s) e = let C (Lam x t') e' = eval t e in
                   eval t' ((x, C s e) : e')
```

1.3 Abstract machine

Definition 8 (Krivine Abstract Machine — KAM)**.** Syntax:

$$\begin{array}{lcll} \text{States} & S & ::= & \langle t \mid e \mid \pi \rangle \\ \text{Stacks} & \pi & ::= & \bullet \mid c : \pi \end{array}$$

The transition relation is defined as $\mapsto \stackrel{\text{def}}{=} \mapsto_{\texttt{app}} \cup \mapsto_{\texttt{lam}} \cup \mapsto_{\texttt{var}}$, where:

$$\begin{array}{rcll} \langle t\, s \mid e \mid \pi \rangle & \mapsto_{\texttt{app}} & \langle t \mid e \mid (s, e) : \pi \rangle & \\ \langle \lambda x.t \mid e \mid c : \pi \rangle & \mapsto_{\texttt{lam}} & \langle t \mid [x \mapsto c] : e \mid \pi \rangle & \\ \langle x \mid e \mid \pi \rangle & \mapsto_{\texttt{var}} & \langle t \mid e' \mid \pi \rangle & \text{if } e(x) = (t, e') \end{array}$$

Stack concatenation is denoted by $\pi_1 : \pi_2$.

Remark 9 (Determinism). If $S \mapsto S_1$ and $S \mapsto S_2$ then $S_1 = S_2$.

Exercise 10. Evaluate $(\lambda x.xx)(I\, I)$ in the KAM.

Algorithm 11. An implementation of the KAM in Haskell:

```
type Id      = String
data State   = S Term Stack Env
data Term    = Var Id | Lam Id Term | App Term Term
type Env     = [(Id, Closure)]
data Closure = C Term Env
```

```haskell
type Stack   = [Closure]

exec1 :: State -> Maybe State
exec1 (S (App t s) p e)       = Just (S t (C s e : p) e)
exec1 (S (Lam x t) (c : p) e) = Just (S t p ((x, c) : e))
exec1 (S (Var x) p e)         = let Just (C t e') = lookup x e
                                 in Just (S t p e')
exec1 _                       = Nothing

exec :: State -> State
exec s = case exec1 s of
           Nothing -> s
           Just s' -> exec s'
```

Definition 12. The notion of being *closed* is defined inductively as follows:

- A term t is closed in an environment e if $\mathsf{fv}(t) \subseteq \mathrm{dom}\, e$. A term t is closed (*a secas*) if $\mathsf{fv}(t) = \varnothing$.
- A closure (t, e) is closed if the term t is closed in e and, moreover, e is closed.
- An environment $[x_1 \mapsto c_1] : \ldots : [x_n \mapsto c_n]$ is closed if c_i is closed for all $i = 1..n$.
- A stack $c_1 : \ldots : c_n$ is closed if c_i is closed for all $i = 1..n$.
- A state $\langle t \mid e \mid \pi \rangle$ is closed if the closure (t, e) is closed and the stack π is closed.

Lemma 13 (KAM invariant)**.** *If $S \mapsto S'$ and S is closed then S' is closed. Remark that if t is a closed term, then $\langle t \mid \bullet \mid \bullet \rangle$ trivially fulfills the invariant.*

Proof. By case analysis on the transitions. □

1.4 Equivalence: big-step evaluation vs. abstract machine

Lemma 14 (Stack weakening)**.** *If $\langle t \mid e \mid \pi_1 \rangle \mapsto \langle t' \mid e' \mid \pi_1' \rangle$ then $\langle t \mid e \mid \pi_1 : \pi_2 \rangle \mapsto \langle t' \mid e' \mid \pi_1' : \pi_2 \rangle$.*

Proof. By case analysis on the transitions of the KAM. □

Proposition 15 (From big-step to the KAM)**.** *If $t \Downarrow^e (s, e')$ then $\langle t \mid e \mid \bullet \rangle \mapsto^* \langle s \mid e' \mid \bullet \rangle$.*

Proof. By induction on the derivation that $t \Downarrow^e (s, e')$ using Lem. **??**. □

As an intermediate step to prove the converse of Prop. **??**, we need the following generalization of the judgment $t \Downarrow^e c$:

Definition 16 (Generalized evaluation)**.** Big-step call-by-name evaluation is generalized for an arbitrary stack π as follows:

$$\frac{t \Downarrow^e c}{t \Downarrow^e_\pi c} \qquad \frac{t \Downarrow^e (\lambda x.t', e') \qquad t' \Downarrow^{[x \mapsto c_1] : e'}_\pi c_2}{t \Downarrow^e_{c_1 : \pi} c_2}$$

Lemma 17 (Properties of generalized evaluation)**.** *The judgment $t \Downarrow^e_\pi c$ has the following properties:*

1. *If* $e(x) = (t, e')$ *and* $t \Downarrow_{\pi}^{e'} c$ *then* $x \Downarrow_{\pi}^{e} c$.

2. *If* $t \Downarrow^{e} (\lambda x.t', e')$ *and* $t' \Downarrow_{\pi}^{[x\mapsto(s,e)]:e'} c$ *then* $t\,s \Downarrow_{\pi}^{e} c$.

Proof. By induction on π. ☐

Proposition 18 (From KAM to big-step)**.** *If* $S = \langle t \mid e \mid \pi\rangle \mapsto^{*} \langle s \mid e' \mid \pi'\rangle = S'$ *and* S *fulfills the KAM invariant and* S' *is in* $\mapsto$*-normal form, then* π' *is empty and* $t \Downarrow_{\pi}^{e} (s, e')$.

Proof. By induction on the number of transitions in $S \mapsto^{*} S'$ and case analysis on the shape of t, relying on Lem. **??**. ☐

1.5 Equivalence: small-step evaluation vs. abstract machine

Definition 19 (KAM decoding)**.**

$$\begin{array}{rcl}
\underline{\langle t \mid e \mid \pi\rangle} & \stackrel{\text{def}}{=} & \underline{\pi}\langle \underline{t^{e}}\rangle \\
\underline{\bullet} & \stackrel{\text{def}}{=} & \Box \\
\underline{c : \pi} & \stackrel{\text{def}}{=} & \underline{\pi}\langle \Box\,\underline{c}\rangle \\
\underline{t^{\bullet}} & \stackrel{\text{def}}{=} & t \\
\underline{t^{[x\mapsto c]:e}} & \stackrel{\text{def}}{=} & \underline{t\{x := \underline{c}\}^{e}}
\end{array}$$

Lemma 20 (Properties of the decoding)**.** *The KAM decoding has the following properties:*

1. $\underline{(\lambda x.t)^{e}} = \lambda x.\underline{t^{e}}$

2. $\underline{(t\,s)^{e}} = \underline{t^{e}}\,\underline{s^{e}}$

3. *If* e *and* e' *are closed environments equal up to a permutation then* $\underline{t^{e}} = \underline{t^{e'}}$.

4. *If* t *is a closed term, then* $\underline{t^{e}} = t$.

Proof. By induction on e. ☐

Proposition 21 (KAM correctness)**.** *If* $S \mapsto S'$ *and the states fulfill the KAM invariant then* $\underline{S} \to_{\mathsf{name}}^{*} \underline{S'}$. *Furthermore:*

- *If* $S \mapsto_{\mathtt{app}} S'$ *then* $\underline{S} = \underline{S'}$.
- *If* $S \mapsto_{\mathtt{lam}} S'$ *then* $\underline{S} \to_{\mathsf{name}} \underline{S'}$.
- *If* $S \mapsto_{\mathtt{var}} S'$ *then* $\underline{S} = \underline{S'}$.

Proof. By case analysis on the transitions of the KAM, using Lem. **??**. ☐

Proposition 22 (KAM completeness)**.** *If* $t \to_{\mathsf{name}} t'$ *and* S *is a state fulfilling the KAM invariant such that* $\underline{S} = t$, *then there exists a state* S' *such that* $S \mapsto^{*} S'$ *and* $\underline{S'} = t'$.

Proof. Observe that the $\mapsto_{\mathtt{var}}$ transition strictly decreases the size of the environment, and the $\mapsto_{\mathtt{app}}$ transition preserves the size of the environment while strictly decreasing the size of the term. Hence $\mapsto_{\mathtt{app},\mathtt{var}}$ is terminating.

Normalize S with respect to $\mapsto_{\mathtt{app},\mathtt{var}}$ transitions, obtaining $S \mapsto^{*} S_1$. By correctness (Prop. **??**), $t = \underline{S} = \underline{S_1}$. Note that the term of S_1 is an abstraction, so $S_1 = \langle \lambda x.s \mid \pi \mid e\rangle$. If the stack π is empty, then

$t = \underline{S_1} = \underline{(\lambda x.s)^e}$ is in $\to_{\mathsf{name}}$-normal form contradicting the fact that $t \to_{\mathsf{name}} t'$. So the stack is non-empty, $\pi = c : \pi'$ and we have:

$$S \mapsto^*_{\mathtt{app,var}} S_1 = \langle \lambda x.s \mid c : \pi' \mid e \rangle \mapsto_{\mathtt{lam}} \langle s \mid \pi' \mid [x \mapsto c] : e \rangle = S'$$

By correctness (Prop. **??**), $t = \underline{S} = \underline{S_1} \to_{\mathsf{name}} \underline{S'}$. So by determinism of both $\mapsto$ and $\to_{\mathsf{name}}$ we conclude that $\underline{S'} = t'$, as required. □

2 Call-by-value

2.1 Small-step evaluation

Definition 23 (Small-step call-by-value evaluation)**.** Terms and evaluation contexts are given by:

Terms	t	$::=$	$x \mid \lambda x.t \mid t\,t$
Values	v	$::=$	$\lambda x.t$
Weak by-value contexts	$\mathtt{V}$	$::=$	$\Box \mid \mathtt{V}\,t \mid v\,\mathtt{V}$

The binary relation of small-step call-by-value evaluation is defined as follows:

$$\mathtt{V}\langle(\lambda x.t)\,v\rangle \to_{\mathsf{value}} \mathtt{V}\langle t\{x := v\}\rangle$$

Exercise 24. Evaluate $(\lambda x.xx)(I\,I)$ using small-step call-by-value evaluation.

2.2 Big-step evaluation

Definition 25 (Big-step call-by-value evaluation)**.** Call-by-value environments and closures are given by the following abstract syntax:

Environments	e	$::=$	$\bullet \mid [x \mapsto c] : e$
Closures	c	$::=$	(v, e)

Derivability of the *big-step call-by-value evaluation* judgment $t \Downarrow^e c$ is defined as follows:

$$\frac{e(x) = c}{x \Downarrow^e c} \qquad \frac{}{\lambda x.t \Downarrow^e (\lambda x.t, e)} \qquad \frac{t \Downarrow^e (\lambda x.t', e') \qquad s \Downarrow^e c_1 \qquad t' \Downarrow^{[x \mapsto c]:e'} c_2}{t\,s \Downarrow^e c_2}$$

Exercise 26. Evaluate $(\lambda x.xx)(I\,I)$ using big-step call-by-value evaluation.

2.3 Abstract machine

Definition 27 (CEK Machine)**.** Syntax:

States	S	$::=$	$\langle t \mid e \mid \pi \rangle$
Stacks	π	$::=$	$\bullet \mid \mathbf{A}(t, e) : \pi \mid \mathbf{F}(v, e) : \pi$

The transition relation is defined as follows:

$$\begin{array}{rcll} \langle t\,s \mid e \mid \pi \rangle & \mapsto & \langle t \mid e \mid \mathbf{F}(s, e) : \pi \rangle & \\ \langle v \mid e \mid \mathbf{A}(t, e') : \pi \rangle & \mapsto & \langle t \mid e' \mid \mathbf{F}(v, e) : \pi \rangle & \\ \langle v \mid e \mid \mathbf{F}(\lambda x.t, e') : \pi \rangle & \mapsto & \langle t \mid [x \mapsto (v, e)] : e' \mid \pi \rangle & \\ \langle x \mid e \mid \pi \rangle & \mapsto & \langle v \mid e' \mid \pi \rangle & \text{if } e(x) = (v, e') \end{array}$$

Exercise 28. Evaluate $(\lambda x.xx)(I\,I)$ in the CEK.

3 Call-by-need

3.1 Small-step evaluation

In contrast to call-by-name and call-by-value, small-step call-by-need evaluation cannot be expressed directly in the λ-calculus. To be able to express call-by-need as a small-step reduction strategy, we need to extend the set of terms with *explicit substitutions*.

Definition 29 (Small-step call-by-need evaluation)**.** Terms and evaluation contexts are given by:

$$\begin{array}{llcl}
\text{Terms} & t & ::= & x \mid \lambda x.t \mid t\,t \mid t[x := t] \\
\text{Substitution contexts} & \mathtt{L} & ::= & \Box \mid \mathtt{L}[x := t] \\
\text{Values} & v & ::= & \lambda x.t \\
\text{Weak by-need contexts} & \mathtt{N} & ::= & \Box \mid \mathtt{N}\,t \mid \mathtt{N}[x := t] \mid \mathtt{N}\langle x\rangle[x := \mathtt{N}]
\end{array}$$

Substitution contexts are lists of explicit substitutions, $\mathtt{L} = \Box[x_1 := t_1]\ldots[x_n := t_n]$. We write $t\mathtt{L}$ for $t[x_1 := t_1]\ldots[x_n := t_n]$ rather than $\mathtt{L}\langle t\rangle$. The binary relation of small-step call-by-need evaluation is defined as the union $\to_{\mathsf{need}} = \to_{\mathtt{db}} \cup \to_{\mathtt{lv}} \cup \to_{\mathtt{gc}}$ of the following three relations:

$$\begin{array}{rcll}
\mathtt{N}\langle(\lambda x.t)\mathtt{L}\,s\rangle & \to_{\mathtt{db}} & \mathtt{N}\langle t[x := s]\mathtt{L}\rangle & \text{distant beta} \\
\mathtt{N}_1\langle \mathtt{N}_2\langle x\rangle[x := v\mathtt{L}]\rangle & \to_{\mathtt{lv}} & \mathtt{N}_1\langle \mathtt{N}_2\langle v\rangle[x := v]\mathtt{L}\rangle & \text{linear value substitution} \\
\mathtt{N}\langle t[x := s]\rangle & \to_{\mathtt{gc}} & \mathtt{N}\langle t\rangle \quad \text{if } x \notin \mathsf{fv}(t) & \text{garbage collection}
\end{array}$$

The three rules above are not deterministic. For example, in a term like $(I\,I)[x := t]$ the first and the third rule may apply. One can show that the system without the last rule is deterministic, and the garbage collection rule can be postponed:

Lemma 30 (Postponement of garbage-collection)**.** *If $t \to_{\mathtt{gc}}\to_{\mathtt{db,lv}} s$ then $t \to_{\mathtt{db,lv}}\to^*_{\mathtt{gc}} s$.*

Proof. By case analysis on all the possibilities in which a $\to_{\mathtt{gc}}$ step is followed by a $\to_{\mathtt{db,lv}}$ step. □

Exercise 31. Evaluate $(\lambda x.xx)(I\,I)$ using small-step call-by-need evaluation.

3.2 Big-step evaluation

Definition 32 (Big-step call-by-need evaluation)**.** Let $\mathcal{L} = \{\ell_1, \ell_2, \ldots\}$ be a denumerable set of *memory locations*. Call-by-need environments and closures are given by the following abstract syntax:

$$\begin{array}{llcl}
\text{Environments} & e & ::= & \bullet \mid [x \mapsto \ell] : e \\
\text{Memories} & \mu & ::= & \bullet \mid [\ell \mapsto \mathbf{T}(t, e)] : \mu \mid [\ell \mapsto \mathbf{V}(v, e)] : \mu \\
\text{Closures} & c & ::= & (v, e)
\end{array}$$

Derivability of the *big-step call-by-need evaluation* judgment $t \mathbin{@} \mu_1 \Downarrow^e c \mathbin{@} \mu_2$ is defined as follows:

$$\frac{\ell = e(x) \qquad \mu(\ell) = \mathbf{V}(v, e')}{x \mathbin{@} \mu \Downarrow^e (v, e') \mathbin{@} \mu} \qquad \frac{}{\lambda x.t \mathbin{@} \mu \Downarrow^e (\lambda x.t, e) \mathbin{@} \mu}$$

$$\frac{\ell = e(x) \qquad \mu_1(\ell) = \mathbf{T}(t, e') \qquad t \mathbin{@} \mu_1 \Downarrow^{e'} (v, e'') \mathbin{@} \mu_2}{x \mathbin{@} \mu_1 \Downarrow^e (v, e'') \mathbin{@} [\ell \mapsto \mathbf{V}(v, e'')] : \mu_2}$$

$$\frac{t \mathbin{@} \mu_1 \Downarrow^e (\lambda x.t', e') \mathbin{@} \mu_2 \qquad \ell \text{ fresh} \qquad t' \mathbin{@} [\ell \mapsto \mathbf{T}(s, e)] : \mu_2 \Downarrow^{[x \mapsto \ell] : e'} c \mathbin{@} \mu_3}{ts \mathbin{@} \mu_1 \Downarrow^e c \mathbin{@} \mu_3}$$

Exercise 33. Evaluate $(\lambda x.xx)(I\,I)$ using big-step call-by-need evaluation.

3.3 Abstract machine

The following machine is based on Sestoft's:

Definition 34 (Milner by-need Asbtract Machine)**.** Syntax:

$$\begin{array}{llcl} \text{States} & S & ::= & \langle t \mid \pi \mid D \mid E \rangle \\ \text{Stacks} & \pi & ::= & \bullet \mid t : \pi \\ \text{Dumps} & D & ::= & (x, \pi) : D \\ \text{Global environments} & E & ::= & \bullet \mid [x \mapsto t] : E \end{array}$$

The transition relation is defined as follows:

$$\begin{array}{rcll} \langle ts \mid \pi \mid D \mid E \rangle & \mapsto & \langle t \mid s : \pi \mid D \mid E \rangle & \\ \langle \lambda x.t \mid s : \pi \mid D \mid E \rangle & \mapsto & \langle t \mid \pi \mid D \mid [x \mapsto s] : E \rangle & \\ \langle x \mid \pi \mid D \mid E \rangle & \mapsto & \langle t \mid \bullet \mid (x, \pi) : D \mid E \rangle & \text{if } E(x) = t \\ \langle v \mid \bullet \mid (x, \pi) : D \mid E \rangle & \mapsto & \langle v^{\alpha} \mid \pi \mid D \mid [x \mapsto v] : E \rangle & \end{array}$$

Exercise 35. Evaluate $(\lambda x.xx)(I\,I)$ in Milner by-need Abstract Machine.