Formal Verifications of Call-by-Need and Call-by-Name Evaluations with Mutual Recursion

Masayuki Mizuno
Joint work with Eijiro Sumii

Tohoku University

November 19, 2019

Our goals

- Long-term: formally verified compiler for non-strict language (e.g. Haskell)

Our goals

- Long-term: formally verified compiler for non-strict language (e.g. Haskell)
- Short-term: formal verification of correspondence between
  - Call-by-name [Plotkin 1975 etc.]: High-level specification of non-strict languages
  - Call-by-need [Wadsworth 1971 etc.]: Implementation of non-strict languages

Background 1: non-strict language

Allows users to define non-strict functions e.g.

$$\texttt{const42}\ x = \textbf{if true then } 42 \textbf{ else } x$$

$$\begin{aligned} & \texttt{const42}\ (\texttt{loop}\ ()) \\ \twoheadrightarrow\ & 42 \end{aligned}$$

$$\text{where} \quad \texttt{loop}\ () = 3 + \texttt{loop}\ ()$$

Background 2: call-by-name

Substitute arguments without evaluating them

$$\texttt{const42}\ x = \textbf{if true then } 42 \textbf{ else } x$$

$$\texttt{const42}\,(\texttt{loop}\,())$$

where $\quad \texttt{loop}\,() = 3 + \texttt{loop}\,()$

Background 2: call-by-name

Substitute arguments without evaluating them

$$\texttt{const42}\ x = \textbf{if true then}\ 42\ \textbf{else}\ x$$

$$\begin{aligned} & \texttt{const42}\ (\texttt{loop}\ ()) \\ \xrightarrow{\text{name}}\ & \textbf{if true then}\ 42\ \textbf{else}\ \texttt{loop}\ () \end{aligned}$$

where $\texttt{loop}\ () = 3 + \texttt{loop}\ ()$

Background 2: call-by-name

Substitute arguments without evaluating them

$$\texttt{const42}\ x = \textbf{if true then } 42 \textbf{ else } x$$

$$\begin{aligned} & \texttt{const42}\ (\texttt{loop}\ ()) \\ \xrightarrow{\text{name}}\ & \textbf{if true then } 42 \textbf{ else } \texttt{loop}\ () \\ \xrightarrow{\text{name}}\ & 42 \end{aligned}$$

where $\texttt{loop}\ () = 3 + \texttt{loop}\ ()$

Problem of call-by-name: redundant computations

$$\texttt{double}\ x = x + x$$

$$\texttt{double}\ (3 + 3)$$

Problem of call-by-name: redundant computations

$$\texttt{double}\ x = x + x$$

$$\begin{aligned} & \texttt{double}\ (3+3) \\ \xrightarrow{\text{name}}\ & (3+3)+(3+3) \end{aligned}$$

Problem of call-by-name: redundant computations

$$\texttt{double}\ x = x + x$$

$$\begin{aligned} & \texttt{double}\ (3+3) \\ \xrightarrow{\text{name}}\ & (3+3)+(3+3) \\ \xrightarrow{\text{name}}\ & 6+(3+3) \end{aligned}$$

Problem of call-by-name: redundant computations

$$\texttt{double}\ x = x + x$$

$$\begin{aligned} & \texttt{double}\ (3+3) \\ \xrightarrow{\text{name}}\ & (3+3)+(3+3) \\ \xrightarrow{\text{name}}\ & 6+(3+3) \\ \xrightarrow{\text{name}}\ & 6+6 \end{aligned}$$

Problem of call-by-name: redundant computations

$$\texttt{double}\ x = x + x$$

$$
\begin{aligned}
& \texttt{double}\ (3+3) \\
\xrightarrow{\text{name}}\ & (3+3)+(3+3) \\
\xrightarrow{\text{name}}\ & 6+(3+3) \\
\xrightarrow{\text{name}}\ & 6+6 \\
\xrightarrow{\text{name}}\ & 12
\end{aligned}
$$

Background 3: call-by-need

- Shares values of arguments

$$\texttt{double}\ x = x + x$$

$$\texttt{double}\ (3 + 3)$$

Background 3: call-by-need

- Shares values of arguments

$$\texttt{double}\ x = x + x$$

$$\texttt{double}\ (3+3)$$

$$\xrightarrow{\text{need}}\ x + x \quad \mathit{where} \quad x = 3 + 3$$

Background 3: call-by-need

- Shares values of arguments

$$\texttt{double}\ x = x + x$$

$$\begin{array}{lllll} & \texttt{double}\ (3+3) & & & \\ \xrightarrow{\text{need}} & x + x & \textit{where} & x & = 3 + 3 \\ \xrightarrow{\text{need}} & x + x & \textit{where} & x & = 6 \end{array}$$

Background 3: call-by-need

- Shares values of arguments

$$\texttt{double}\ x = x + x$$

$$\begin{array}{ll} & \texttt{double}\ (3+3) \\ \xrightarrow{\text{need}} & x + x \quad \mathit{where} \quad x = 3+3 \\ \xrightarrow{\text{need}} & x + x \quad \mathit{where} \quad x = 6 \\ \xrightarrow{\text{need}} & 12 \end{array}$$

Background 3: call-by-need

- Shares values of arguments

$$\texttt{double}\ x = x + x$$

$$\begin{aligned}
& \texttt{double}\ (3+3) \\
\xrightarrow{\text{need}}\ & x + x \quad \textit{where} \quad x = 3 + 3 \\
\xrightarrow{\text{need}}\ & x + x \quad \textit{where} \quad x = 6 \\
\xrightarrow{\text{need}}\ & 12
\end{aligned}$$

- Should correspond with call-by-name

Challenge: recursion (1/2)

- Implicit recursion destroys sharing

$$Y\ f = (\lambda x.\ f\ (x\ x))\ (\lambda x.\ f\ (x\ x))$$

$$\textbf{let}\ x = e_x\ \textbf{in}\ x \quad \Rightarrow \quad Y\ (\lambda x.\ e_x)$$

  - Computation of e_x inside λx is repeated

Challenge: recursion (2/2)

- Explicit recursion requires sophisticated mechanism
  - Dependency in small-step semantics [Ariola+ 97]

$$
\begin{array}{lcl}
E & ::= & \cdots \\
 & | & \mathbf{let}\ D, x = E\ \mathbf{in}\ E'[x] \\
 & | & \mathbf{let}\ x_n = E, D[x, x_n]\ \mathbf{in}\ E'[x] \\
D[x, x_n] & ::= & x = E[x_1], \cdots, x_{n-1} = E[x_n], D
\end{array}
$$

  - Fixed point in denotational semantics [Launchbury 93]

$$\mu\rho'.\ \rho \sqcup (x_1 \mapsto [\![e_1]\!]_{\rho'} \cdots x_n \mapsto [\![e_n]\!]_{\rho'})$$

Our results

- Simple formal verification (in Coq) of correspondence among call-by-need and 3 different styles of call-by-name evaluations of language with explicit mutual recursion

Our results

- Simple formal verification (in Coq) of correspondence among call-by-need and 3 different styles of call-by-name evaluations of language with explicit mutual recursion
  - Call-by-need natural semantics

Our results

- Simple formal verification (in Coq) of correspondence among call-by-need and 3 different styles of call-by-name evaluations of language with explicit mutual recursion
  - Call-by-need natural semantics
    - heap-based [Launchbury 93]

Our results

- Simple formal verification (in Coq) of correspondence among call-by-need and 3 different styles of call-by-name evaluations of language with explicit mutual recursion
  - Call-by-need natural semantics
    - heap-based [Launchbury 93]
  - Call-by-name natural semantics

Our results

- Simple formal verification (in Coq) of correspondence among call-by-need and 3 different styles of call-by-name evaluations of language with explicit mutual recursion
  - Call-by-need natural semantics
    - heap-based [Launchbury 93]
  - Call-by-name natural semantics
    - heap-based [Launchbury 93]

Our results

- Simple formal verification (in Coq) of correspondence among call-by-need and 3 different styles of call-by-name evaluations of language with explicit mutual recursion
  - Call-by-need natural semantics
    - heap-based [Launchbury 93]
  - Call-by-name natural semantics
    - heap-based [Launchbury 93]
    - closure-based (cf. [Launchbury 93])

Our results

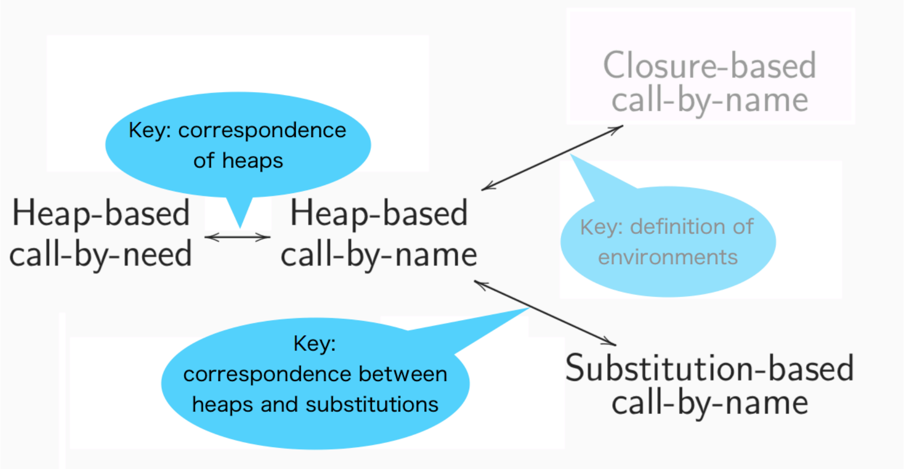
- Simple formal verification (in Coq) of correspondence among call-by-need and 3 different styles of call-by-name evaluations of language with explicit mutual recursion
  - Call-by-need natural semantics
    - heap-based [Launchbury 93]
  - Call-by-name natural semantics
    - heap-based [Launchbury 93]
    - closure-based (cf. [Launchbury 93])
    - substitution-based [Church 36]

Our results

- Simple formal verification (in Coq) of correspondence among call-by-need and 3 different styles of call-by-name evaluations of language with explicit mutual recursion
  - Call-by-need natural semantics
    - heap-based [Launchbury 93]
  - Call-by-name natural semantics
    - heap-based [Launchbury 93]
    - closure-based (cf. [Launchbury 93])
    - substitution-based [Church 36]
- Cf. correspondence with call-by-name denotational semantics [Breitner 18]

Proof outline

- Consists of 3 correspondences



Outline

1. Proof of the correspondences
2. Formalization in Coq
3. Conclusion

Outline

1. Proof of the correspondences
2. Formalization in Coq
3. Conclusion

Our target language

- λ-calculus with mutually recursive bindings
  - Using de Bruijn indices

$$
\begin{array}{llll}
x,\ l & \in & \mathit{Nat} & \\
H,\ \overline{e} & \in & \mathit{Terms} & \\
v & \in & \mathit{Value} & ::= \ \mathbf{abs}\ e \\
e & \in & \mathit{Term} & ::= \ \mathbf{var}\ x \mid \mathbf{loc}\ l \mid \mathbf{abs}\ e \\
 & & & \ \ \mid \ \ \mathbf{app}\ e_1\ e_2 \mid \mathbf{let}\ \overline{e}\ \mathbf{in}\ e
\end{array}
$$

Correspondence between heap-based call-by-need and call-by-name evaluations

Closure-based
call-by-name

Heap-based
call-by-need

Heap-based
call-by-name

Substitution-based
call-by-name

Call-by-need semantics [Launchbury 93]

$$\boxed{\langle H_1 \rangle e \Downarrow_{\mathtt{d}} \langle H_2 \rangle v}$$

$$\frac{\langle H_1 \rangle e_1 \Downarrow_{\mathtt{d}} \langle H_2 \rangle \mathbf{abs}\ e_0 \qquad \langle H_2,\ e_2 \rangle e_0[0 \mapsto \mathbf{loc}\ |H_2|] \Downarrow_{\mathtt{d}} \langle H_3 \rangle v}{\langle H_1 \rangle \mathbf{app}\ e_1\ e_2 \Downarrow_{\mathtt{d}} \langle H_3 \rangle v}$$

$$\frac{\langle H_1 \rangle H_1.l \Downarrow_{\mathtt{d}} \langle H_2 \rangle v}{\langle H_1 \rangle \mathbf{loc}\ l \Downarrow_{\mathtt{d}} \langle H_2[l \mapsto v] \rangle v}$$

Call-by-name semantics [Launchbury 93]

$$\boxed{\langle H_1 \rangle e \Downarrow_{\mathtt{m}} \langle H_2 \rangle v}$$

$$\frac{\langle H_1 \rangle e_1 \Downarrow_{\mathtt{m}} \langle H_2 \rangle \mathbf{abs}\ e_0 \qquad \langle H_2,\ e_2 \rangle e_0[0 \mapsto \mathbf{loc}\ |H_2|] \Downarrow_{\mathtt{m}} \langle H_3 \rangle v}{\langle H_1 \rangle \mathbf{app}\ e_1\ e_2 \Downarrow_{\mathtt{m}} \langle H_3 \rangle v}$$

$$\frac{\langle H_1 \rangle H_1.l \Downarrow_{\mathtt{m}} \langle H_2 \rangle v}{\langle H_1 \rangle \mathbf{loc}\ l \Downarrow_{\mathtt{m}} \langle H_2 \rangle v}$$

One of our main theorem

Theorem (soundness of $\Downarrow_{\mathtt{d}}$)

If $\langle H_1 \rangle e_1 \Downarrow_{\mathtt{d}} \langle H_1' \rangle v_1$ with $H_1 \leq_R H_2$ and $e_1 \sim_R e_2$, then $\langle H_2 \rangle e_2 \Downarrow_{\mathtt{m}} \langle H_2' \rangle v_2$ with $H_1' \leq_{R'} H_2'$ and $v_1 \sim_{R'} v_2$ for some $R' \supseteq R$, H_2', and v_2

If call-by-need evaluation converges,
call-by-name evaluation of corresponding heap and term also converges and gives a corresponding value

Intuition of correspondence

$$e \;=\; (\mathbf{let}\ x = (\mathbf{let}\ y = 1 + 2, z = z\ \mathbf{in}\ y)\ \mathbf{in}\ x + x)$$

Intuition of correspondence

$$e \;=\; (\mathbf{let}\ x = (\mathbf{let}\ y = 1 + 2, z = z\ \mathbf{in}\ y)\ \mathbf{in}\ x + x)$$

$$\langle\rangle e \Downarrow_{\mathtt{d}} \langle l_1 \mapsto 3, l_2 \mapsto 3, l_3 \mapsto l_3\rangle 6$$

Intuition of correspondence

$$e \;=\; (\textbf{let}\ x = (\textbf{let}\ y = 1 + 2, z = z\ \textbf{in}\ y)\ \textbf{in}\ x + x)$$

$$\langle\rangle e \Downarrow_{\mathtt{d}} \langle l_1 \mapsto 3, l_2 \mapsto 3, l_3 \mapsto l_3\rangle 6$$

$$\langle\rangle e \Downarrow_{\mathtt{m}} \langle l_1 \mapsto (\textbf{let}\ y = 1 + 2, z = z\ \textbf{in}\ y), \\ l_2 \mapsto 1 + 2, l_3 \mapsto l_3, l_2' \mapsto 1 + 2, l_3' \mapsto l_3'\rangle 6$$

Intuition of correspondence

$$e \ = \ (\textbf{let}\ x = (\textbf{let}\ y = 1 + 2, z = z\ \textbf{in}\ y)\ \textbf{in}\ x + x)$$

$$\langle\rangle e \Downarrow_{\mathtt{d}} \langle l_1 \mapsto 3, l_2 \mapsto 3, l_3 \mapsto l_3\rangle 6$$

$$\langle\rangle e \Downarrow_{\mathtt{m}} \langle l_1 \mapsto (\textbf{let}\ y = 1 + 2, z = z\ \textbf{in}\ y), \quad l_2 \mapsto 1 + 2, l_3 \mapsto l_3, l_2' \mapsto 1 + 2, l_3' \mapsto l_3'\rangle 6$$

- One-to-many correspondnece between call-by-need and call-by-name locations

Intuition of correspondence

$$e \ = \ (\textbf{let}\ x = (\textbf{let}\ y = 1+2, z = z\ \textbf{in}\ y)\ \textbf{in}\ x + x)$$

$$\langle\rangle e \Downarrow_{\mathtt{d}} \langle l_1 \mapsto 3, l_2 \mapsto 3, l_3 \mapsto l_3\rangle 6$$

$$\langle\rangle e \Downarrow_{\mathtt{m}} \langle l_1 \mapsto (\textbf{let}\ y = 1+2, z = z\ \textbf{in}\ y), \quad l_2 \mapsto 1+2, l_3 \mapsto l_3, l_2' \mapsto 1+2, l_3' \mapsto l_3'\rangle 6$$

- One-to-many correspondnece between call-by-need and call-by-name locations
- Contents of the corresponding locations
  - are the same (modulo corresponing locations), or

Intuition of correspondence

$$e = (\textbf{let } x = (\textbf{let } y = 1 + 2, z = z \textbf{ in } y) \textbf{ in } x + x)$$

$$\langle\rangle e \Downarrow_{\mathtt{d}} \langle l_1 \mapsto 3, l_2 \mapsto 3, l_3 \mapsto l_3 \rangle 6$$

$$\langle\rangle e \Downarrow_{\mathtt{m}} \langle l_1 \mapsto (\textbf{let } y = 1 + 2, z = z \textbf{ in } y), \\ l_2 \mapsto 1 + 2, l_3 \mapsto l_3, l'_2 \mapsto 1 + 2, l'_3 \mapsto l'_3 \rangle 6$$

- One-to-many correspondnece between call-by-need and call-by-name locations
- Contents of the corresponding locations
  - are the same (modulo corresponing locations), or
  - give the same value by call-by-name re-evaluation

Definition (lazy correspondence of heaps)

$H_1 \leq_R H_2$ iff

for all $(l_1, l_2) \in R$,

either $H_1.l_1 \sim_R H_2.l_2$

$$\text{or } \exists S, H_2', v_2. \quad \langle H_2 \rangle H_2.l_2 \Downarrow_{\mathtt{m}} \langle H_2' \rangle v_2 \wedge (H_1.l_1 \sim_{(R \circ S) \cup R} v_2) \wedge (H_2 \sim_S H_2')$$

Definition (lazy correspondence of heaps)

$H_1 \leq_R H_2$ iff
for all $(l_1, l_2) \in R$,
either $H_1.l_1 \sim_R H_2.l_2$

$$\text{or } \exists S, H_2', v_2.\quad \langle H_2\rangle H_2.l_2 \Downarrow_{\mathtt{m}} \langle H_2'\rangle v_2 \land (H_1.l_1 \sim_{(R\circ S)\cup R} v_2) \land (H_2 \sim_S H_2')$$

- S is increased part of the correspondence

Definition (lazy correspondence of heaps)

$H_1 \leq_R H_2$ iff
for all $(l_1, l_2) \in R$,
either $H_1.l_1 \sim_R H_2.l_2$
or $\exists S, H_2', v_2.\quad \langle H_2 \rangle H_2.l_2 \Downarrow_{\mathtt{m}} \langle H_2' \rangle v_2 \wedge$
$(H_1.l_1 \sim_{(R \circ S) \cup R} v_2) \wedge (H_2 \sim_S H_2')$

- S is increased part of the correspondence
- Heaps increased by re-evaluation are homomorphic to original heaps
  - no coinduction required

Proof of our main theorem

Theorem (soundness of $\Downarrow_{\mathtt{d}}$)

If $\langle H_1 \rangle e_1 \Downarrow_{\mathtt{d}} \langle H_1' \rangle v_1$ with $H_1 \leq_R H_2$ and $e_1 \sim_R e_2$, then $\langle H_2 \rangle e_2 \Downarrow_{\mathtt{m}} \langle H_2' \rangle v_2$ with $H_1' \leq_{R'} H_2'$ and $v_1 \sim_{R'} v_2$ for some $R' \supseteq R$, H_2', and v_2

Proof outline.

By induction on the derivation of $\langle H_1 \rangle e_1 \Downarrow_{\mathtt{d}} \langle H_1' \rangle v_1$
The essential case is evaluation of locations:

$$\frac{\langle H_1 \rangle H_1.l \Downarrow_{\mathtt{d}} \langle H_1'' \rangle v}{\langle H_1 \rangle \mathbf{loc}\ \ l \Downarrow_{\mathtt{d}} \langle H_1''[l \mapsto v] \rangle v}$$

Proof outline.

$$\frac{\langle H_1 \rangle H_1.l \Downarrow_{\mathtt{d}} \langle H_1'' \rangle v}{\langle H_1 \rangle \mathbf{loc}\ \ l \Downarrow_{\mathtt{d}} \langle H_1''[l \mapsto v] \rangle v}$$

From $H_1 \leq_R H_2$, we have two subcases:

Proof outline.

$$\frac{\langle H_1 \rangle H_1.l \Downarrow_{\mathtt{d}} \langle H_1'' \rangle v}{\langle H_1 \rangle \mathbf{loc}\ l \Downarrow_{\mathtt{d}} \langle H_1''[l \mapsto v] \rangle v}$$

From $H_1 \leq_R H_2$, we have two subcases:

Subcase (thunk update): $H_1.l_1 \sim_R H_2.l_2$

To show $H_1''[l_1 \mapsto v_1] \leq_{R'} H_2'$,
we use the induction hypothesis and apply
location renaming

Proof outline.

$$\frac{\langle H_1\rangle H_1.l \Downarrow_{\mathtt{d}} \langle H_1''\rangle v}{\langle H_1\rangle \mathbf{loc}\ l \Downarrow_{\mathtt{d}} \langle H_1''[l \mapsto v]\rangle v}$$

From $H_1 \leq_R H_2$, we have two subcases:

Subcase (thunk update): $H_1.l_1 \sim_R H_2.l_2$

To show $H_1''[l_1 \mapsto v_1] \leq_{R'} H_2'$,
we use the induction hypothesis and apply location renaming

Subcase (re-evaluation): $\langle H_2\rangle H_2.l_2 \Downarrow_{\mathtt{m}} \langle H_2'\rangle v_2$, $H_2 \sim_S H_2'$, $H_1.l_1 \sim_{R \circ S} v_2$

Straightforward (thanks to the definition of $H_1 \leq_R H_2$)

□

Our main theorems

- Call-by-need evaluation converges
  $\Rightarrow$ call-by-name evaluation converges
- Call-by-name evaluation converges
  $\Rightarrow$ call-by-need evaluation converges
- Call-by-need evaluation diverges
  $\Rightarrow$ call-by-name evaluation diverges
- Call-by-name evaluation diverges
  $\Rightarrow$ call-by-need evaluation diverges
    - Cf. coinductive definition of divergence [Leroy 06]

Correspondence between heap-based and substitution-based call-by-name evaluations

Closure-based
call-by-name

Heap-based
call-by-need

Heap-based
call-by-name

Substitution-based
call-by-name

Heap-based call-by-name semantics

[Launchbury 93]

$$\boxed{\langle H_1 \rangle e \Downarrow_{\mathtt{m}} \langle H_2 \rangle v}$$

$$\frac{\langle H_1 \rangle e_1 \Downarrow_{\mathtt{m}} \langle H_2 \rangle \mathbf{abs}\ e_0 \qquad \langle H_2,\ e_2 \rangle e_0[0 \mapsto \mathbf{loc}\ |H_2|] \Downarrow_{\mathtt{m}} \langle H_3 \rangle v}{\langle H_1 \rangle \mathbf{app}\ e_1\ e_2 \Downarrow_{\mathtt{m}} \langle H_3 \rangle v}$$

$$\frac{\langle H_1,\ \overline{e}[\forall x \mapsto \mathbf{loc}\ (|H_1| + x)] \rangle\ e[\forall x \mapsto \mathbf{loc}\ (|H_1| + x)] \Downarrow_{\mathtt{m}} \langle H_2 \rangle v}{\langle H_1 \rangle \mathbf{let}\ \overline{e}\ \mathbf{in}\ e \Downarrow_{\mathtt{m}} \langle H_2 \rangle v}$$

Substitution-based call-by-name semantics

$$\boxed{e \Downarrow_{\mathtt{m}} v}$$

$$\frac{e_1 \Downarrow_{\mathtt{m}} \textbf{abs}\ e_0 \qquad e_0[0 \mapsto e_2] \Downarrow_{\mathtt{m}} v}{\textbf{app}\ e_1\ e_2 \Downarrow_{\mathtt{m}} v}$$

$$\frac{e[\forall x \mapsto \textbf{let}\ \overline{e}\ \textbf{in}\ \overline{e}.x] \Downarrow_{\mathtt{m}} v}{\textbf{let}\ \overline{e}\ \textbf{in}\ e \Downarrow_{\mathtt{m}} v}$$

Correspondence between heaps-terms and substituted-terms

$$e \;=\; (\mathbf{let}\ x = \lambda w.\, x\ \mathbf{in}\ (\lambda y.\, \lambda z.\, x\, y)\, \mathbf{true})$$

Correspondence between heaps-terms and substituted-terms

$e \;=\; (\textbf{let}\; x = \lambda w.\, x \;\textbf{in}\; (\lambda y.\, \lambda z.\, x\, y)\, \textbf{true})$

$\langle\rangle e \Downarrow_{\mathtt{m}} \langle l_1 \mapsto \lambda w.\, l_1,\, l_2 \mapsto \textbf{true}\rangle \lambda z.\, l_1\, l_2$

Correspondence between heaps-terms and substituted-terms

$e \;=\; (\mathbf{let}\ x = \lambda w.\, x\ \mathbf{in}\ (\lambda y.\, \lambda z.\, x\, y)\, \mathbf{true})$

$\langle\rangle e \Downarrow_{\mathtt{m}} \langle l_1 \mapsto \lambda w.\, l_1,\ l_2 \mapsto \mathbf{true}\rangle \lambda z.\, l_1\, l_2$

$e \Downarrow_{\mathtt{m}} \lambda z.\, (\mathbf{let}\ x = \lambda w.\, x\ \mathbf{in}\ \lambda w.\, x)\, \mathbf{true}$

Correspondence between heaps-terms and substituted-terms

$e \ = \ (\mathbf{let}\ x = \lambda w.\, x\ \mathbf{in}\ (\lambda y.\, \lambda z.\, x\, y)\, \mathbf{true})$

$\langle\rangle e \Downarrow_{\mathtt{m}} \langle l_1 \mapsto \lambda w.\, l_1,\ l_2 \mapsto \mathbf{true}\rangle \lambda z.\, l_1\, l_2$

$e \Downarrow_{\mathtt{m}} \lambda z.\, (\mathbf{let}\ x = \lambda w.\, x\ \mathbf{in}\ \lambda w.\, x)\, \mathbf{true}$

- Correspondence R between locations and substituted terms
  - $R = \{(l_1, \mathbf{let}\ x = \lambda w.\, x\ \mathbf{in}\ \lambda w.\, x), (l_2, \mathbf{true})\}$

Correspondence between heaps-terms and substituted-terms

$$e \;=\; (\textbf{let}\ x = \lambda w.\, x\ \textbf{in}\ (\lambda y.\, \lambda z.\, x\, y)\, \textbf{true})$$

$$\langle\rangle e \Downarrow_{\mathtt{m}} \langle l_1 \mapsto \lambda w.\, l_1,\ l_2 \mapsto \textbf{true}\rangle \lambda z.\, l_1\, l_2$$

$$e \Downarrow_{\mathtt{m}} \lambda z.\, (\textbf{let}\ x = \lambda w.\, x\ \textbf{in}\ \lambda w.\, x)\, \textbf{true}$$

- Correspondence R between locations and substituted terms
  - $R = \{(l_1, \textbf{let}\ x = \lambda w.\, x\ \textbf{in}\ \lambda w.\, x), (l_2, \textbf{true})\}$

$$\boxed{\langle H\rangle e_1 \sim_R e_2} \qquad \frac{(l, e_2) \in R}{\langle H\rangle \textbf{loc}\ l \sim_R e_2}$$

Proof of our main theorem

Theorem (substitution-based $\Rightarrow$ heap-based convergence)

If $e_2 \Downarrow_{\mathtt{m}} v_2$ with $let_R(H)$ and $\langle H \rangle e_1 \sim_R e_2$,
then $\langle H \rangle e_1 \Downarrow_{\mathtt{m}} \langle H' \rangle v_1$ with $let_{R'}(H')$ and $\langle H' \rangle v_1 \sim_{R'} v_2$
for some $R' \supseteq R$, H', and v_2.

Proof outline.

By induction on derivations of $e_2 \Downarrow_{\mathtt{m}} v_2$

Problem: substitution for locations becomes 0-step

- There is no rule in substitution-based semantics that corresponds to:

$$\frac{\langle H \rangle H.l \Downarrow_{\mathtt{m}} \langle H' \rangle v}{\langle H \rangle \mathbf{loc}\ l \Downarrow_{\mathtt{m}} \langle H' \rangle v}$$

Problem: substitution for locations becomes 0-step

- There is no rule in substitution-based semantics that corresponds to:

$$\frac{\langle H \rangle H.l \Downarrow_{\mathtt{m}} \langle H' \rangle v}{\langle H \rangle \mathbf{loc}\ l \Downarrow_{\mathtt{m}} \langle H' \rangle v}$$

- Induction doesn't work in some theorems:
  - substitution-based $\Rightarrow$ heap-based convergence
  - heap-based $\Rightarrow$ substitution-based divergence

Our trick: distinguish bindings for $\mathbf{let}$ and $\mathbf{app}$

- Bindings introduced by $\mathbf{let}$ always involve $\mathbf{let}$-expansions

$$\frac{e[\forall x \mapsto \mathbf{let}\ \overline{e}\ \mathbf{in}\ \overline{e}.x] \Downarrow_{\mathtt{m}} v}{\mathbf{let}\ \overline{e}\ \mathbf{in}\ e \Downarrow_{\mathtt{m}} v}$$

Our trick: distinguish bindings for **let** and **app**

- Bindings introduced by **let** always involve **let**-expansions

$$\frac{e[\forall x \mapsto \textbf{let}\ \overline{e}\ \textbf{in}\ \overline{e}.x] \Downarrow_{\mathtt{m}} v}{\textbf{let}\ \overline{e}\ \textbf{in}\ e \Downarrow_{\mathtt{m}} v}$$

- Bindings introduced by function applications are non-recursive and “finite”

⇒ We therefore distinguish:

Our trick: distinguish bindings for $\mathbf{let}$ and $\mathbf{app}$

- Bindings introduced by $\mathbf{let}$ always involve $\mathbf{let}$-expansions

$$\frac{e[\forall x \mapsto \mathbf{let}\ \overline{e}\ \mathbf{in}\ \overline{e}.x] \Downarrow_{\mathtt{m}} v}{\mathbf{let}\ \overline{e}\ \mathbf{in}\ e \Downarrow_{\mathtt{m}} v}$$

- Bindings introduced by function applications are non-recursive and "finite"

$\Rightarrow$ We therefore distinguish:

For $\mathbf{let}$: $\dfrac{(l, e_2) \in R}{\langle H\rangle \mathbf{loc}\ l \sim_R e_2}$

Our trick: distinguish bindings for $\mathbf{let}$ and $\mathbf{app}$

- Bindings introduced by $\mathbf{let}$ always involve $\mathbf{let}$-expansions

$$\frac{e[\forall x \mapsto \mathbf{let}\ \overline{e}\ \mathbf{in}\ \overline{e}.x] \Downarrow_{\mathtt{m}} v}{\mathbf{let}\ \overline{e}\ \mathbf{in}\ e \Downarrow_{\mathtt{m}} v}$$

- Bindings introduced by function applications are non-recursive and "finite"

$\Rightarrow$ We therefore distinguish:

For $\mathbf{let}$: $\dfrac{(l, e_2) \in R}{\langle H \rangle \mathbf{loc}\ l \sim_R e_2}$ For $\mathbf{app}$: $\dfrac{\langle H \rangle H.l \sim_R e_2}{\langle H \rangle \mathbf{loc}\ l \sim_R e_2}$

Proof of our main theorem

Theorem (substitution-based $\Rightarrow$ heap-based convergence)

If $e_2 \Downarrow_{\mathtt{m}} v_2$ with $let_R(H)$ and $\langle H \rangle e_1 \sim_R e_2$, then $\langle H \rangle e_1 \Downarrow_{\mathtt{m}} \langle H' \rangle v_1$ with $let_{R'}(H')$ and $\langle H' \rangle v_1 \sim_{R'} v_2$ for some $R' \supseteq R$, H', and v_2.

Proof outline.

By induction on derivations of $e_2 \Downarrow_{\mathtt{m}} v_2$ and $\langle H \rangle e_1 \sim_R e_2$ $\square$

Outline

Problem: heap-based ⇒ substitution-based divergence

Theorem

If $\langle H\rangle e_1 \Uparrow_{\mathtt{m}}$ with $let_R(H)$ and $\langle H\rangle e_1 \sim_R e_2$, then $e_2 \Uparrow_{\mathtt{m}}$

Coq's syntactic guard criterion rejects:

```
cofix heap_subst_diverge H e1
  (Hdiv : Heaps.diverge H e) :=
  (fix heap_subst_diverge_inner H e1 R e2
    (Hcorr : corr_term R H e1 e2) :=
    match Hdiv in ... with ...
      Subst.diverge_name_appL e21 e22
        (heap_subst_diverge H e11 R e21 ...)
    ... end) H e1
```

Solution: detour using small-step semantics

$$
\begin{tikzcd}
\langle H \rangle e_1 \Uuparrow_{\mathtt{m}} \arrow[r] \arrow[d, dotted] & \begin{array}{c} \forall H', e_1'.\ \langle H \rangle e_1 \rightarrow_{\mathtt{m}}^{*} \langle H' \rangle e_1' \\ \Longrightarrow \langle H' \rangle e_1' \rightarrow_{\mathtt{m}} \end{array} \arrow[d] \\
e_2 \Uuparrow_{\mathtt{m}} & \forall e_2'.\ e_2 \rightarrow_{\mathtt{m}}^{*} e_2' \Longrightarrow e_2' \rightarrow_{\mathtt{m}} \arrow[l, "\dagger"']
\end{tikzcd}
$$

† uses law of excluded middle

Outline

1. Proof of the correspondences
2. Formalization in Coq
3. Conclusion

Conclusion

- Simpler formal verification in Coq of correspondence among call-by-need and 3 different styles of call-by-name evaluations of language with explicit mutual recursion
  - Call-by-need natural semantics
    - heap-based [Launchbury 93]
  - Call-by-name natural semantics
    - heap-based [Launchbury 93]
    - closure-based (cf. [Launchbury 93])
    - substitution-based [Church 36]
- ♦ Also extended with black hole [Nakata+ 09]

Future work

- Cost analysis of call-by-need evaluations
- Verification of compilers for non-strict languages
  - Part of GHC [The Glasgow Haskell Team 92]?