Termination Analysis of the Untyped λ-Calculus

Neil D. Jones (DIKU, University of Copenhagen)

Nina Bohr (IT University of Copenhagen)

Review: size-change termination of first-order functional programs

POPL 2001: for first-order general recursive functional programs.

- Assumption: *well-founded* set of data values
- Function calls are *labeled*, e.g., $1 : \texttt{f} \to \texttt{g}$

$$\texttt{f(x,y) = y + } \boxed{\texttt{1:}}\,\texttt{g(y-1,x)}$$

- **Call graph:** nodes = function names

  e.g., $\texttt{f} \xrightarrow{1} \texttt{g}$ arcs = function calls

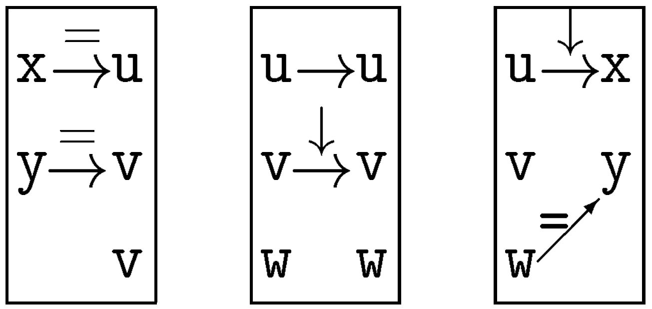
- **Size-change graphs G_c:**

  - one graph G_c for each call $c : \texttt{f} \to \texttt{g}$
  - $\texttt{f}$ parameter sizes to $\texttt{g}$ parameter sizes
  - Relations: $\texttt{f}^i \xrightarrow{\downarrow} \texttt{g}^j$ or $\texttt{f}^i \xrightarrow{=} \texttt{g}^j$ or no relation

Termination of a first-order program

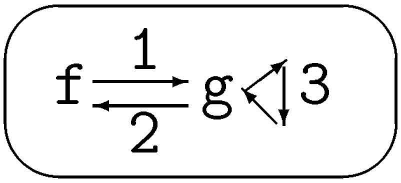
```
f(x,y) = if x=0 then y else 1: g(x,y,0)
g(u,v,w) = if w=0 then 2:f(u-1,w)
                   else 3:g(u,v-1,w+2)
```

Size-change graphs $\mathcal{G}$



G_1 call 1 G_3 call 2 G_2 call 3

Call graph



Termination: Let $cs \in (1+2+3)^\omega$ be an *infinite* call sequence following call graph.

- **Case 1:** $cs = \ldots 3^\omega$:

  Then `v` *descends infinitely*

- **Case 2:** $cs = \ldots (13^*2)^\omega$.

  Then `u` *descends infinitely*

Both are impossible since data is well-founded.

Thus program terminates: from **any** function call, **any** input.

Size-change terminology

1. A *size-change graph* $A \overset{G}{\to} B$ is
   - a *source* set A;
   - a *target* set B; and
   - a set of labeled arcs

$$G \subseteq A \times \{=, \downarrow\} \times B$$

2. $A \overset{G_1}{\to} B$ and $C \overset{G_2}{\to} D$ are *composible* if $B = C$.

Definition A set $\mathcal{G}$ of graphs *satisfies the size-change condition*

- if every infinite composible sequence

$$G_1, G_2, G_3, \ldots$$

  of graphs from $\mathcal{G}$
- has a thread of infinite descent.

Challenges in the pure λ-calculus:

- What is a *control point*: (cf. `f`, `g` above)

  Are there finitely many?

- What is the *call graph*: (cf. $\mathtt{f} \xrightarrow{1} \mathtt{g}$)

  From what and to what ?

- What is a *call sequence* ?
- What is a *value decrease* ?
- How to deal with *evaluation order*:

  (CBN, CBV, normal order) ?

Ω: a nonterminating λ-expression

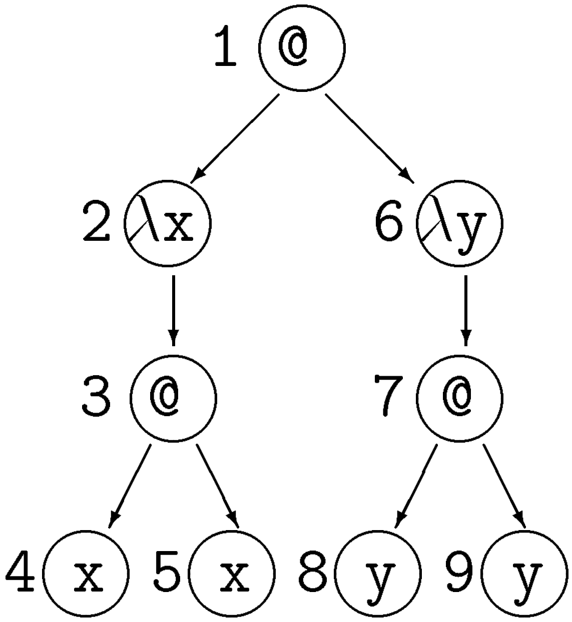
λ-expression $\Omega = (\lambda \mathtt{x}.\mathtt{xx})(\lambda \mathtt{y}.\mathtt{yy})$

Call-by-value reduction:

$$
\begin{aligned}
&(\lambda \mathtt{x}.\mathtt{xx})(\lambda \mathtt{y}.\mathtt{yy}) \to \\
&(\lambda \mathtt{y}.\mathtt{yy})(\lambda \mathtt{y}.\mathtt{yy}) \to \\
&(\lambda \mathtt{y}.\mathtt{yy})(\lambda \mathtt{y}.\mathtt{yy}) \to \ldots
\end{aligned}
$$

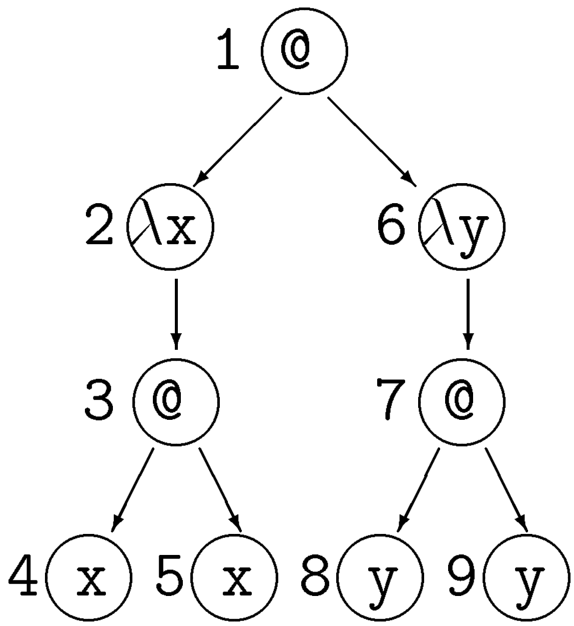
Step 1: label subexpressions of Ω

λ-expression $\Omega = (\lambda\mathtt{x}.\mathtt{xx})(\lambda\mathtt{y}.\mathtt{yy})$

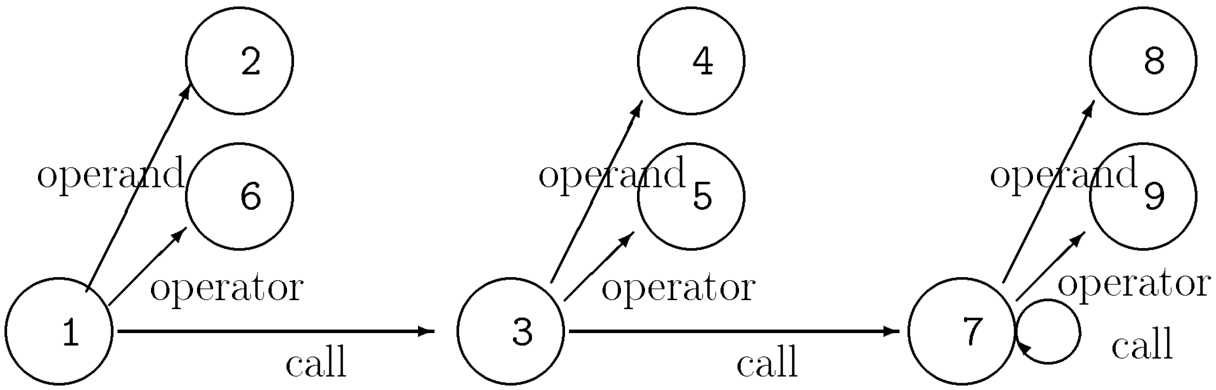


Step 2: build control-flow graph

λ-expression $\Omega = (\lambda\mathtt{x.xx})(\lambda\mathtt{y.yy})$

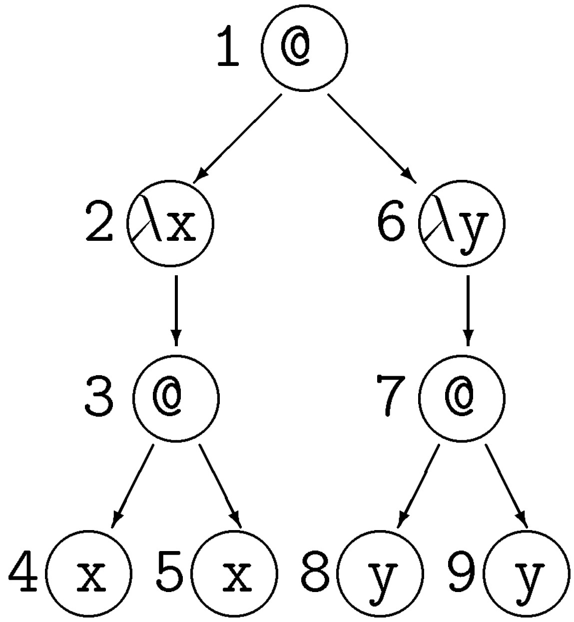


Control-flow graph

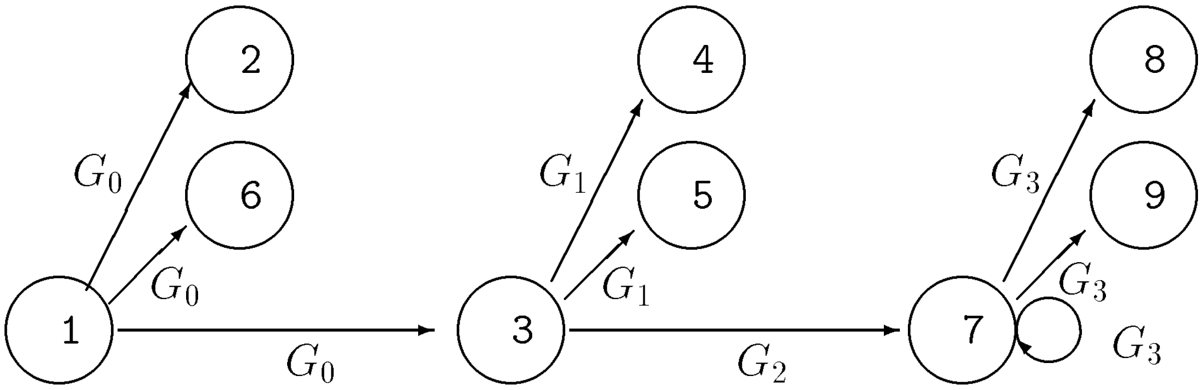


Step 3: add size-change graphs as labels

λ-expression $\Omega = (\lambda \mathtt{x}.\mathtt{xx})(\lambda \mathtt{y}.\mathtt{yy})$



Size-change graphs: G_0, G_1, G_2, G_3



$G_0 = \emptyset$, $G_1 = \{\mathtt{x} \xrightarrow{=} \mathtt{x}\}$, $G_2 = \{\mathtt{x} \xrightarrow{=} \mathtt{y}\}$, $G_3 = \{\mathtt{y} \xrightarrow{=} \mathtt{y}\}$

Observation: The size-change condition *fails* since G_0, G_2, G_3^ω has no thread of infinite descent.

Call-by-value evaluation by rewriting

Standard (e.g., Plotkin 1975) with a deterministic evaluation order.

A **program** is a closed term.

$$\frac{}{\lambda x\,.\,e \Downarrow \lambda x\,.\,e}$$

$$\frac{\mathtt{e}_1 \Downarrow \lambda\mathtt{x}.\mathtt{e}_0 \qquad \mathtt{e}_2 \Downarrow v_2 \qquad \mathtt{e}_0[\mathtt{x} := v_2] \Downarrow v}{\mathtt{e}_1@\mathtt{e}_2 \Downarrow v}$$

Note: Rewriting and β-reduction can lead to

- many intermediate expressions,
- sometines ∞ many

Problems for static analysis

$\triangleright$ How to **identify nontermination ?**

- How **finitely to describe**
  the entire "computation space" ?
- What is a **looping computation ?**
  (so "size-change condition" can be applied)
- What is a **value decrease ?**
- **How to construct**
  size-change graphs from a program `P` ?
- **What can call what** in application $\mathtt{e}_1 @ \mathtt{e}_2$?
  (need to decide it statically)

How to identify nontermination

Problem:
A deduction of $\mathsf{e} \Downarrow v$ is a *finite* object; it tells us *nothing* about nontermination.

Solution: Add to the semantics a $\boxed{\textbf{call relation}}$ $\mathsf{e} \to \mathsf{e}'$.

Rationale: $\mathsf{e} \to \mathsf{e}'$ holds if some rule has form

$$\boxed{\dfrac{\ldots\ \mathsf{e}' \Downarrow u \ldots}{\mathsf{e} \Downarrow v}}$$

i.e., $\mathsf{e} \to \mathsf{e}'$ if

- in order to find v such that $\mathsf{e} \Downarrow v$
- one must first find u such that $\mathsf{e}' \Downarrow u$

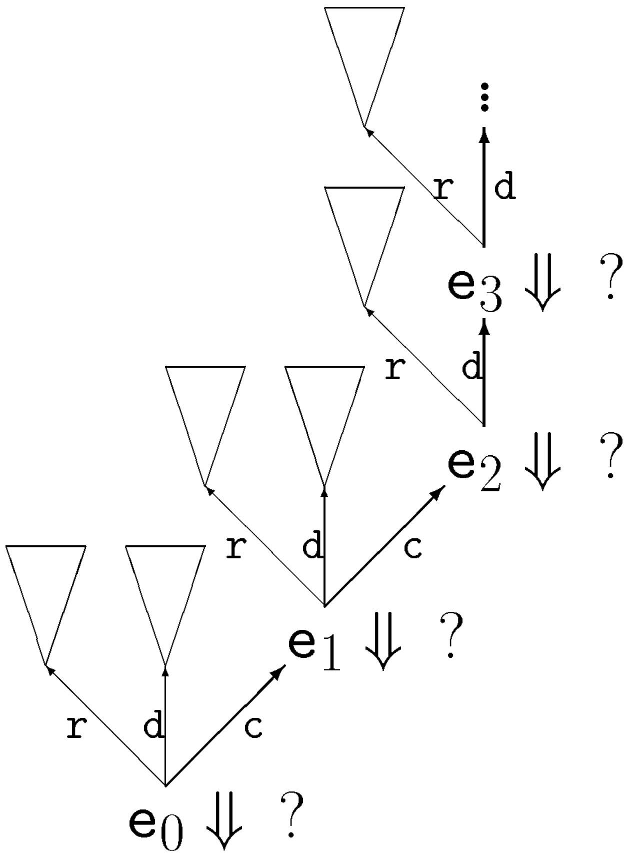
NIS: Nontermination Is Sequential

Lemma $\mathtt{P} \Downarrow v$ for some v if and only if there exists *no infinite call sequence*

$$\mathtt{P} = \mathtt{e}_0 \to \mathtt{e}_1 \to \mathtt{e}_2 \to \ldots$$

Proof A straightforward induction. ▭

A picture of ***NIS*** behavior



Code: $\mathtt{r}$ = "Operato**r**", $\mathtt{d}$ = "Operan**d**", $\mathtt{c}$ = "**c**all".

Evaluation and call semantic rules

Evaluation: (as before)

$$\frac{}{\lambda x \,.\, e \Downarrow \lambda x \,.\, e}$$

$$\frac{\mathtt{e}_1 \Downarrow \lambda \mathtt{x}.\mathtt{e}_0 \quad \mathtt{e}_2 \Downarrow v_2 \quad \mathtt{e}_0[\mathtt{x} := v_2] \Downarrow v}{\mathtt{e}_1 @ \mathtt{e}_2 \Downarrow v}$$

Calls: Define $\mathtt{e} \to \mathtt{e}'$ by

$$\to \;=\; \underset{r}{\to} \cup \underset{d}{\to} \cup \underset{c}{\to} \quad \text{and:}$$

Call operator: $$\frac{}{\mathtt{e}_1 @ \mathtt{e}_2 \underset{r}{\to} \mathtt{e}_1}$$

Call operand: $$\frac{\mathtt{e}_1 \Downarrow v_1}{\mathtt{e}_1 @ \mathtt{e}_2 \underset{d}{\to} \mathtt{e}_2}$$

Function call: $$\frac{\mathtt{e}_1 \Downarrow \lambda \mathtt{x}.\mathtt{e}_0 \quad \mathtt{e}_2 \Downarrow v_2}{\mathtt{e}_1 @ \mathtt{e}_2 \underset{c}{\to} \mathtt{e}_0[\mathtt{x} := v_2]}$$

Overlap. Double computation of:

$$\mathtt{e}_2 \Downarrow v_2 \quad \text{and} \quad \mathtt{e}_0[\mathtt{x} := v_2]$$

A simpler call-and-evaluation semantics

Evaluation:

$$\frac{}{\lambda x\,.\,e \Downarrow \lambda x\,.\,e}$$

$$\frac{\mathsf{e}_1@\mathsf{e}_2 \underset{c}{\to} \mathsf{e}' \quad \mathsf{e}' \Downarrow v}{\mathsf{e}_1@\mathsf{e}_2 \Downarrow v} \qquad \textit{changed}$$

Calls: Define $\mathsf{e} \to \mathsf{e}'$ by

$$\to \;=\; \underset{r}{\to} \cup \underset{d}{\to} \cup \underset{c}{\to} \quad \text{and:}$$

Call operator: $$\frac{}{\mathsf{e}_1@\mathsf{e}_2 \underset{r}{\to} \mathsf{e}_1}$$

Call operand: $$\frac{\mathsf{e}_1 \Downarrow v_1}{\mathsf{e}_1@\mathsf{e}_2 \underset{d}{\to} \mathsf{e}_2}$$

Function call: $$\frac{\mathsf{e}_1 \Downarrow \lambda\mathtt{x}.\mathsf{e}_0 \qquad \mathsf{e}_2 \Downarrow v_2}{\mathsf{e}_1@\mathsf{e}_2 \underset{c}{\to} \mathsf{e}_0[\mathtt{x} := v_2]}$$

Problems for static analysis

$\surd$ How to **identify nontermination ?**

$\triangleright$ How **finitely to describe**

the entire "computation space" ?

- What is a **looping computation ?**

  (so "size-change condition" can be applied)

- What is a **value decrease ?**

- **How to construct**

  size-change graphs from a program $\mathtt{P}$?

- **What can call what** in application $\mathtt{e}_1@\mathtt{e}_2$?

  (need to decide it statically)

Next Problem:

Finite description of the entire computation space.

Environment-based semantics:
Let *State*, *Val*, *Env* be the smallest sets such that

$$\begin{array}{lcl} State & = & \{ \; \mathsf{e} : \rho \;\mid\; \mathsf{e} \in Exp, \rho \in Env, \\ & & \quad fv(\mathsf{e}) \subseteq dom(\rho) \} \\[1em] Env & = & \{ \; \rho : X \to Value \;\mid\; X \text{ finite var'bl set} \} \\[1em] Value & = & \{ \; \lambda \mathsf{x}.\mathsf{e} : \rho \;\mid\; \lambda \mathsf{x}.\mathsf{e} : \rho \in State \} \end{array}$$

A value is a *closure* $\lambda \mathsf{x}.\mathsf{e} : \rho$

Evaluation judgement is

$$s \Downarrow v$$

where $s \in State, v \in Value$.

Call-and-evaluation with environments

Evaluation:

$$\frac{}{v \Downarrow v} \text{ (If } v \in \mathit{Value}\text{)}$$

$$\frac{}{\mathtt{x}\colon \rho \Downarrow \rho(\mathtt{x})}$$

$$\frac{\mathsf{e}_1@\mathsf{e}_2 : \rho \underset{c}{\rightarrow} \mathsf{e}' : \rho' \qquad \mathsf{e}' : \rho' \Downarrow v}{\mathsf{e}_1@\mathsf{e}_2 : \rho \Downarrow v}$$

Call:

$$\frac{}{\mathsf{e}_1@\mathsf{e}_2 : \rho \underset{r}{\rightarrow} \mathsf{e}_1 : \rho}$$

$$\frac{\mathsf{e}_1 : \rho \Downarrow v_1}{\mathsf{e}_1@\mathsf{e}_2 : \rho \underset{d}{\rightarrow} \mathsf{e}_2 : \rho}$$

$$\frac{\mathsf{e}_1 : \rho \Downarrow \lambda\mathtt{x}.\mathsf{e}_0 : \rho_0 \qquad \mathsf{e}_2 : \rho \Downarrow v_2}{\mathsf{e}_1@\mathsf{e}_2 : \rho \underset{c}{\rightarrow} \mathsf{e}_0 : \rho_0[\mathtt{x} \mapsto v_2]}$$

The Subexpression Lemma

Define the *expression support* $sup(s)$ of state s by

$$sup(\mathtt{e} : \rho) = subexp(\mathtt{e}) \ \cup \bigcup_{\mathtt{x} \in dom(\rho)} sup(\rho(\mathtt{x}))$$

Lemma (Subexpression property)
If $s \Downarrow s'$ or $s \to s'$ then $sup(s) \supseteq sup(s')$.

Corollary If $\mathtt{P} : [] \Downarrow \lambda\mathtt{x}.\mathtt{e} : \rho$ then
$\lambda\mathtt{x}.\mathtt{e} \in subexp(\mathtt{P})$.

Progress: towards size-change termination analysis of a λ-program $\mathtt{P}$:

- *"Control points"* = the function applications in $\mathtt{P}$ that cause computation.
- Since $\mathtt{P}$ is fixed, there are only finitely many control points.

The NIS phenomenon with environments

Lemma $\mathtt{P} : [\,] \rhd v$ for some v if and only if there exists *no infinite sequence*

$$\mathtt{P} : [\,] = \mathtt{e}_0{:}\rho_0 \to \mathtt{e}_1{:}\rho_1 \to \mathtt{e}_2{:}\rho_2 \to \ldots$$

Observations:

- Every subexpression $\mathtt{e}_i \in \mathit{subexp}(\mathtt{P})$
  by the Subexpression lemma
- $\mathit{subexp}(\mathtt{P})$ is a finite set, so
  some subexpressions $\mathtt{e}$ <u>*occur infinitely often*</u>.

Our algorithm focuses on the *data-flow*:

- the **size relations**

- between consecutive environments ρ_i and ρ_{i+1}
  in the sequence.

Problems for static analysis

$\surd$ How to **identify nontermination ?**

$\surd$ How **finitely to describe**

the entire "computation space" ?

$\triangleright$ What is a **looping computation ?**

(so "size-change condition" can be applied)

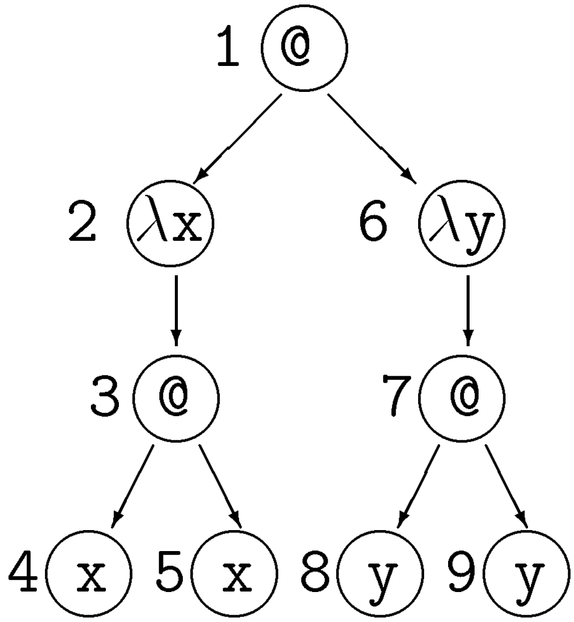
- What is a **value decrease ?**
- **How to construct**

  size-change graphs from a program $\mathtt{P}$?
- **What can call what** in application $\mathtt{e}_1@\mathtt{e}_2$?

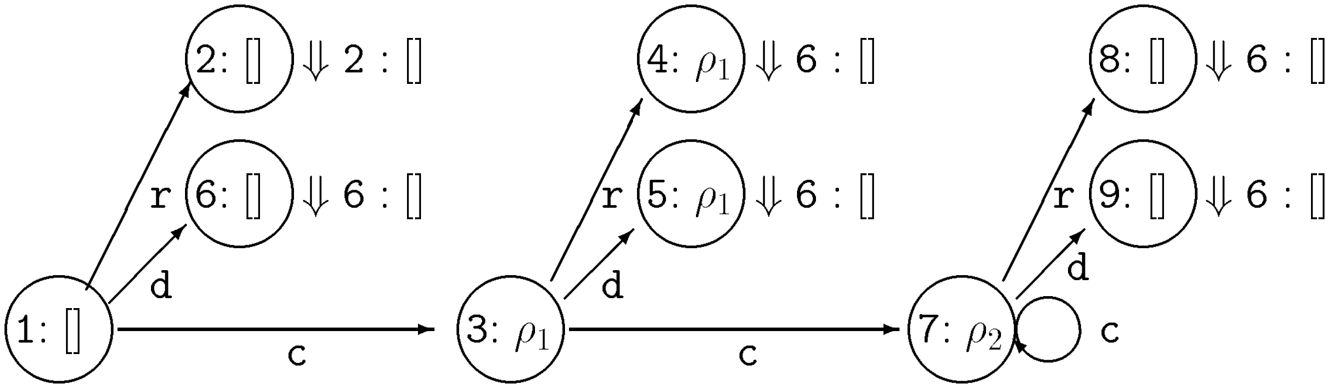
  (need to decide it statically)

A nonterminating program

λ-expression Ω



The "calls" relation $\rightarrow$



where $\rho_1 = [\mathtt{x} \mapsto 6 : []]$ and $\rho_2 = [\mathtt{y} \mapsto 6 : []]$

This repeats without environment growth.

Also possible: loops containing environments that grow without bound.

Problems for static analysis

$\surd$ How to **identify nontermination ?**

$\surd$ How **finitely to describe**

the entire "computation space" ?

$\surd$ What is a **looping computation ?**

(so "size-change condition" can be applied)

$\triangleright$ What is a **value decrease ?**

- **How to construct**

  size-change graphs from a program $\mathtt{P}$?

- **What can call what** in application $\mathtt{e}_1@\mathtt{e}_2$?

  (need to decide it statically)

Structure of states and environments

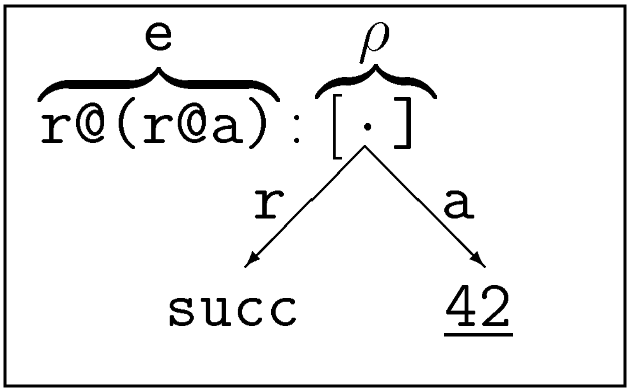
$$\texttt{e} \ ::= \ \texttt{x} \mid \texttt{e @ e} \mid \lambda\texttt{x.e}$$

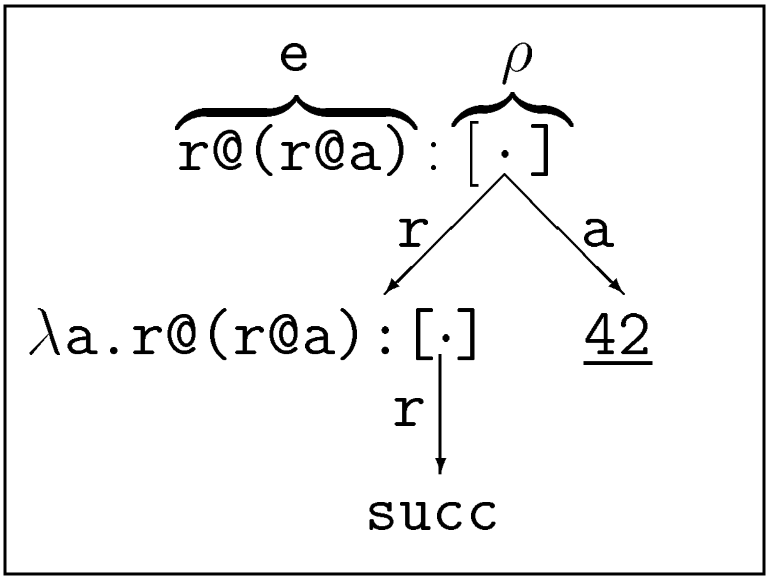
s ::= $\texttt{e}:\rho$ **State**

v ::= $\lambda\texttt{x.e}:\rho$ **Value**

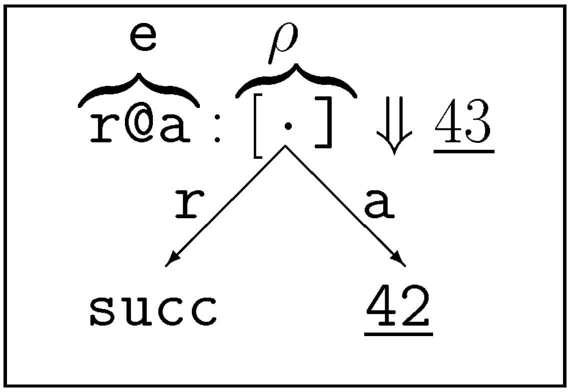
$\rho : X \to \mathit{Value}$ where $X \subseteq \{\texttt{x}_1, \texttt{x}_2, \ldots\}$

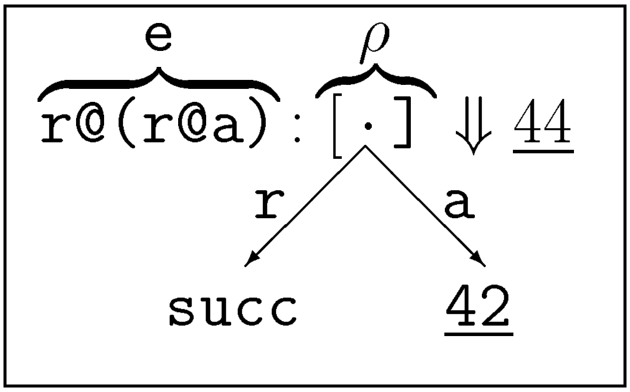
An environment ρ is a finite tree. Two examples of states:

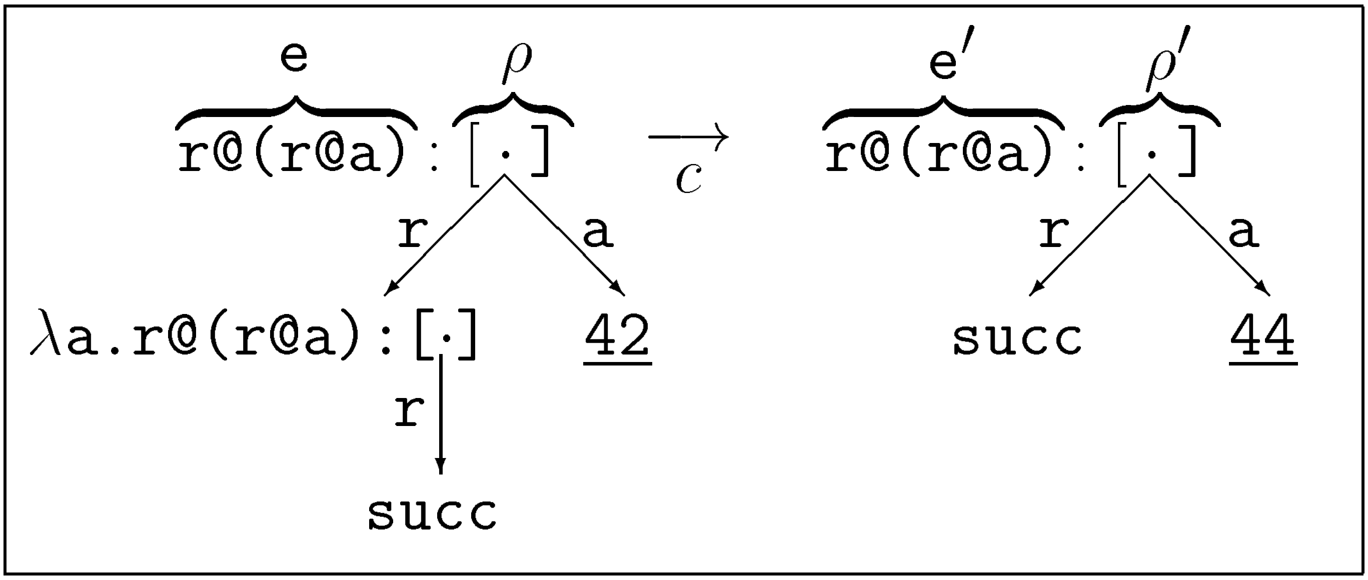


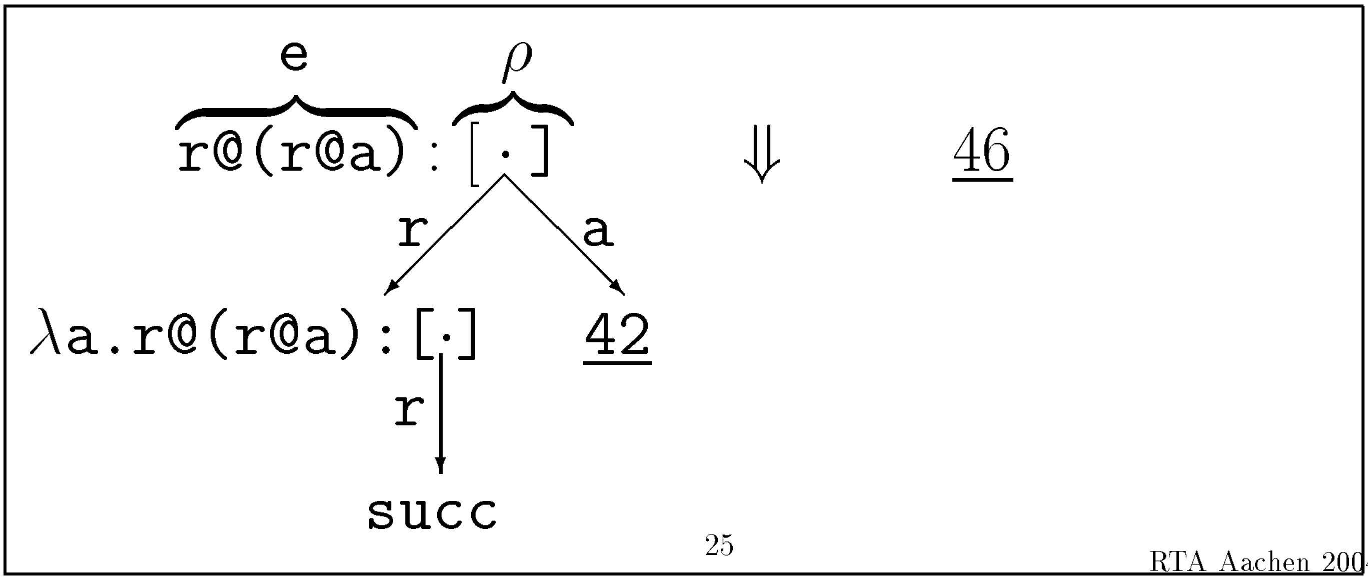


Some example evaluations and calls





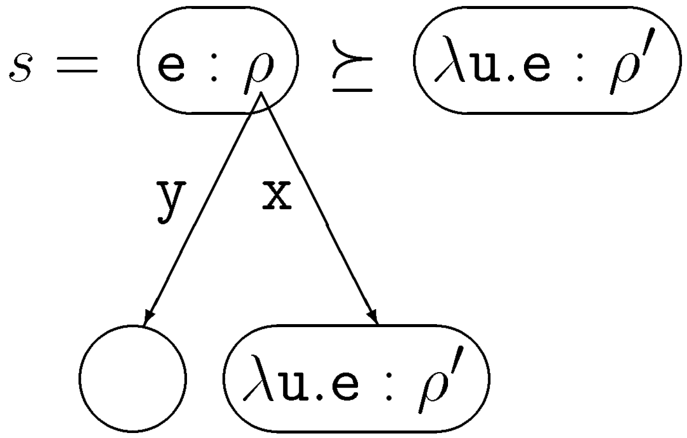




Size changes in λ-calculus values

Definition: Two parts to size relation $\succeq$ on states:
a *subtree* size relation:

$s \succeq \rho(\mathtt{x})$ if $s = \mathsf{e} : \rho$ and $\mathtt{x} \in dom(\rho)$



and a *subexpression* size relation:

$\mathsf{e} : \rho \succeq \mathsf{e}' : \rho'$ if $subexp(\mathsf{e}) \ni \mathsf{e}'$ and
$\rho(\mathtt{x}) = \rho'(\mathtt{x})$ for all $\mathtt{x} \in fv(\mathsf{e}')$.

plus the usual:

- $\succeq$ is reflexive and transitive
- $s \succ s'$ holds if $s \succeq s'$ and $s \neq s'$.

Lemma The state size relation $\succeq$ is well-founded.

Size change graphs for envir. semantics

Definition Suppose $s \to s'$ or $s \Downarrow s'$, and

$$s' = \mathsf{e} : \rho, s = \mathsf{e}' : \rho'$$

A size-change graph G for this call or evaluation has

source set $A = (fv(\mathsf{e}) \cup \{\bullet\})$; and

target set $B = (fv(\mathsf{e}') \cup \{\bullet\})$.

How to read an arc $\alpha \stackrel{r}{\to} \beta$ in size-change graph G:

- $\alpha \stackrel{=}{\to} \beta$ represents a *copied value*
- $\alpha \stackrel{\downarrow}{\to} \beta$ represents a *decreasing value*

Further,

- $\bullet$ in A represents state s
- $\mathbf{x}$ in A represents a free variable of e
- $\bullet$ in B represents state s'
- y in B represents a free variable of e'

Safety of a size change graph

Definition G is **safe** for $s \to s'$ or $s \Downarrow s'$ if

$$s = \mathsf{e} : \rho, s = \mathsf{e}' : \rho'$$

and

1. $\mathtt{x} \xrightarrow{=} \mathtt{y}$ in G implies $\rho(\mathtt{x}) = \rho'(\mathtt{y})$ and
   $\mathtt{x} \xrightarrow{\downarrow} \mathtt{y}$ in G implies $\rho(\mathtt{x}) \succ \rho'(\mathtt{y})$

2. $\bullet \xrightarrow{=} \bullet$ in G implies $s = s'$ and
   $\bullet \xrightarrow{\downarrow} \bullet$ in G implies $s \succ s'$

and following the same pattern:

3. $\mathtt{x} \xrightarrow{=} \bullet$ in G implies $\rho(\mathtt{x}) = s'$ and
   $\mathtt{x} \xrightarrow{\downarrow} \bullet$ in G implies $\rho(\mathtt{x}) \succ s'$
4. $\bullet \xrightarrow{=} \mathtt{y}$ in G implies $s = \rho'(\mathtt{y})$ and
   $\bullet \xrightarrow{\downarrow} \mathtt{y}$ in G implies $s \succ \rho'(\mathtt{y})$

The size-change condition $\Rightarrow$ termination

Theorem Program $\mathtt{P} \Downarrow$ if

1. size-change graph set $\mathcal{G}$ is safe for $\mathtt{P}$;
2. $\mathtt{P} : [] \rightarrow^* \mathtt{e} : \rho \rightarrow \mathtt{e}' : \rho'$ implies some $G \in \mathcal{G}$ is safe for $\mathtt{e} : \rho \rightarrow \mathtt{e}' : \rho'$; and
3. $\mathcal{G}$ satisfies the size-change condition.

Proof If $\mathtt{P} \not\Downarrow$, then by the NIS lemma there is an infinite call sequence

$$\mathtt{P} : [\,] = \mathtt{e}_0{:}\rho_0 \rightarrow \mathtt{e}_1{:}\rho_1 \rightarrow \mathtt{e}_2{:}\rho_2 \rightarrow \ldots$$

By 2 there are graphs

$$G_0, G_1, \ldots \in \mathcal{G}$$

such that each G_i is safe for $\mathtt{e}_i{:}\rho_i \rightarrow \mathtt{e}_{i+1}{:}\rho_{i+1}$.

By 3 (size-change condition) sequence $G_0, G_1, \ldots$ has an *infinite descending thread.*

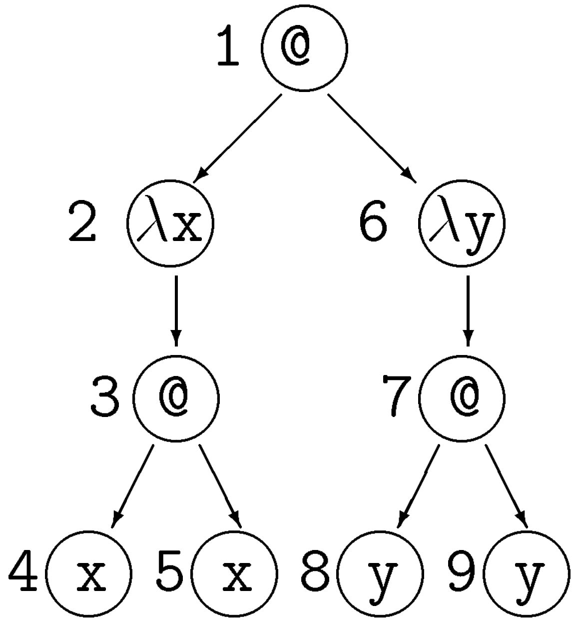
By 1 $\exists$ an *infinite* $\succeq$*-descending value sequence* within

$$\mathtt{e}_0{:}\rho_0 \rightarrow \mathtt{e}_1{:}\rho_1 \rightarrow \mathtt{e}_2{:}\rho_2 \rightarrow \ldots$$

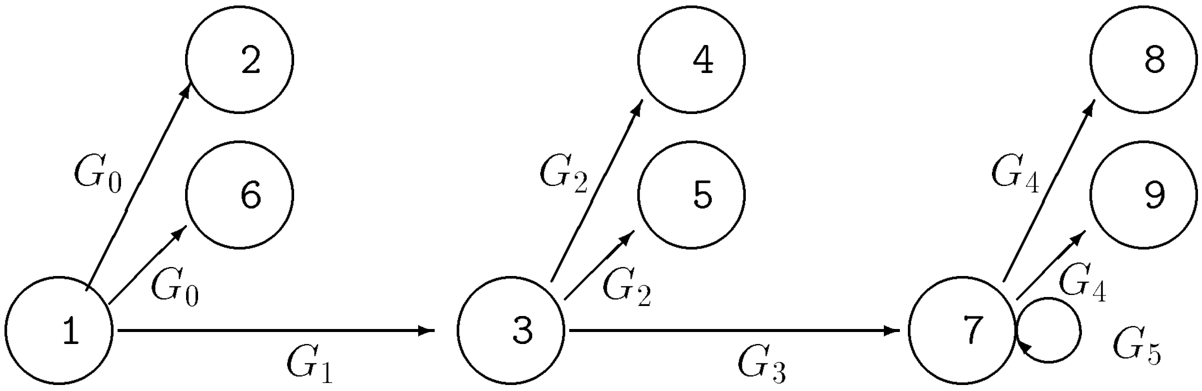
Contradiction since $\succeq$ is well-founded.

A safe set $\mathcal{G}$ of size-change graphs for Ω

λ-expression Ω



Set of size-change graphs $\mathcal{G} = \{G_0, G_1, \ldots, G_5\}$



$G_0 = \{\bullet \overset{\downarrow}{\to} \bullet\}$, $\quad G_1 = \{\bullet \overset{\downarrow}{\to} \mathtt{x}\}$, $\quad G_2 = \{\mathtt{x} \overset{=}{\to} \mathtt{x}, \bullet \overset{\downarrow}{\to} \bullet,\}$

$G_3 = \{\mathtt{x} \overset{=}{\to} \mathtt{y}\}$, $\quad G_4 = \{\mathtt{y} \overset{=}{\to} \mathtt{y}, \bullet \overset{\downarrow}{\to} \bullet,\}$, $\quad G_5 = \{\mathtt{y} \overset{=}{\to} \mathtt{y}\}$

Observation: $\mathcal{G}$ *fails* the size-change condition since G_0, G_2, G_3^{ω} has no thread of infinite descent.

Problems for static analysis

$\surd$ How to **identify nontermination ?**

$\surd$ How **finitely to describe**

the entire "computation space" ?

$\surd$ What is a **looping computation ?**

(so "size-change condition" can be applied)

$\surd$ What is a **value decrease ?**

$\triangleright$ **How to construct**

size-change graphs from a program $\mathtt{P}$?

- **What can call what** in application $\mathtt{e}_1@\mathtt{e}_2$?

  (need to decide it statically)

Semantics-based graph construction

1. Show how to **instrument** the evaluation and call semantic rules, to *build size-change graphs while evaluating or calling.*
2. Extended judgement forms:

$$\mathsf{e} : \rho \to \mathsf{e} : \rho, G \quad \text{and} \quad \mathsf{e} : \rho \Downarrow \mathsf{e} : \rho, G$$

3. Prove that these graphs are *always safe.*

Drawback for program analysis:

- Graphs are not obtained by static computations. (You have to run the program!)
- We show later how to overcome this drawback.

Some easy instrumented semantic rules

$$\frac{}{\lambda \mathsf{x.e} : \rho \Downarrow \lambda \mathsf{x.e} : \rho, id^{=}_{\lambda \mathsf{x.e}}}$$

$$\frac{}{\mathsf{e}_1 @ \mathsf{e}_2 : \rho \underset{r}{\rightarrow} \mathsf{e}_1 : \rho, id^{\downarrow}_{\mathsf{e}_1}}$$

$$\frac{\mathsf{e}_1 : \rho \Downarrow v_1}{\mathsf{e}_1 @ \mathsf{e}_2 : \rho \underset{d}{\rightarrow} \mathsf{e}_2 : \rho, id^{\downarrow}_{\mathsf{e}_2}}$$

Notations:

$id^{=}_{\mathsf{e}}$ stands for $\{\mathsf{x} \overset{=}{\rightarrow} \mathsf{y} \mid \mathsf{y} \in fv(\mathsf{e})\} \cup \{\bullet \overset{=}{\rightarrow} \bullet\}$

$id^{\downarrow}_{\mathsf{e}}$ stands for $\{\mathsf{x} \overset{=}{\rightarrow} \mathsf{y} \mid \mathsf{y} \in fv(\mathsf{e})\} \cup \{\bullet \overset{\downarrow}{\rightarrow} \bullet\}$

Another easy one:

$$\frac{\mathsf{e}_1 @ \mathsf{e}_2 : \rho \underset{c}{\rightarrow} \mathsf{e}' : \rho', G' \qquad \mathsf{e}' : \rho' \Downarrow v, G}{\mathsf{e}_1 @ \mathsf{e}_2 : \rho \Downarrow v, (G'; G)}$$

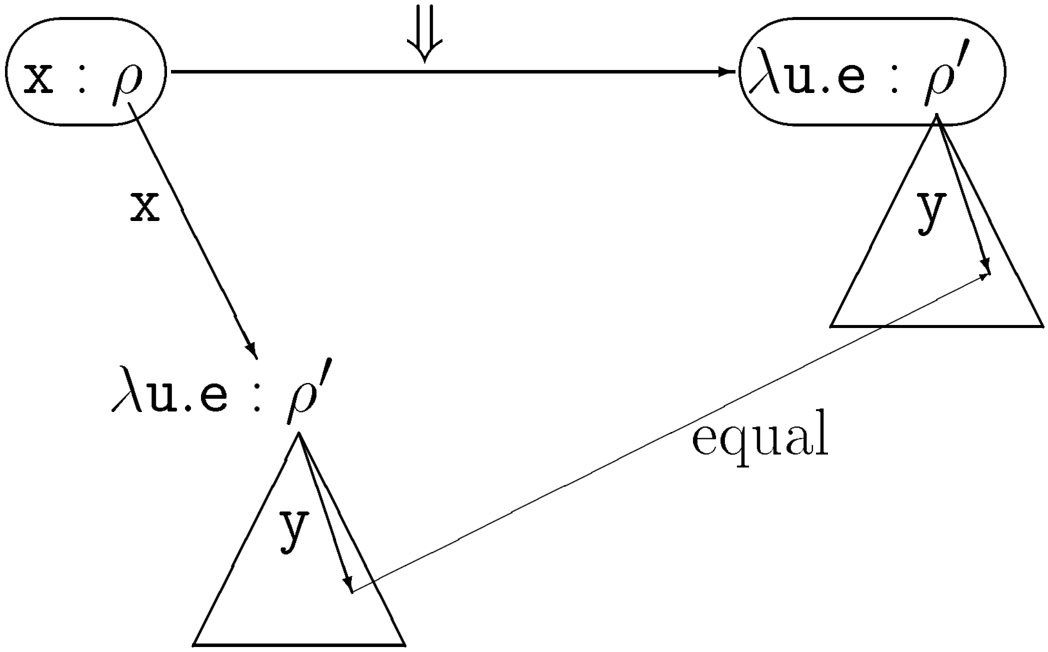
How to instrument variable lookup

$$\frac{(\text{If } \rho(\mathtt{x}) = \lambda\mathtt{x.e} : \rho')}{\mathtt{x} : \rho \Downarrow \rho(\mathtt{x}), (\{\mathtt{x} \overset{=}{\to} \bullet\} \cup \{\mathtt{x} \overset{\downarrow}{\to} \mathtt{y} \mid \mathtt{y} \in fv(\lambda\mathtt{x.e})\})}$$

Explanation:

1. Arc $\boxed{\mathtt{x} \overset{=}{\to} \bullet}$:
   The value $\rho(\mathtt{x})$ equals the value bound to $\mathtt{x}$ on the left of $\Downarrow$.

2. Arcs $\boxed{\mathtt{x} \overset{\downarrow}{\to} \mathtt{y}}$ where $\mathtt{y} \in fv(\lambda\mathtt{x.e})$:



How to instrument a function call

$$\frac{\mathtt{e}_1 : \rho \Downarrow \lambda\mathtt{x}.\mathtt{e}_0 : \rho_0, G_1 \qquad \mathtt{e}_2 : \rho \Downarrow v_2, G_2}{\mathtt{e}_1@\mathtt{e}_2 : \rho \underset{c}{\rightarrow} \mathtt{e}_0 : \rho_0[\mathtt{x} \mapsto v_2], (G_1^{-\bullet} \cup G_2^{\bullet\mapsto\mathtt{x}})}$$

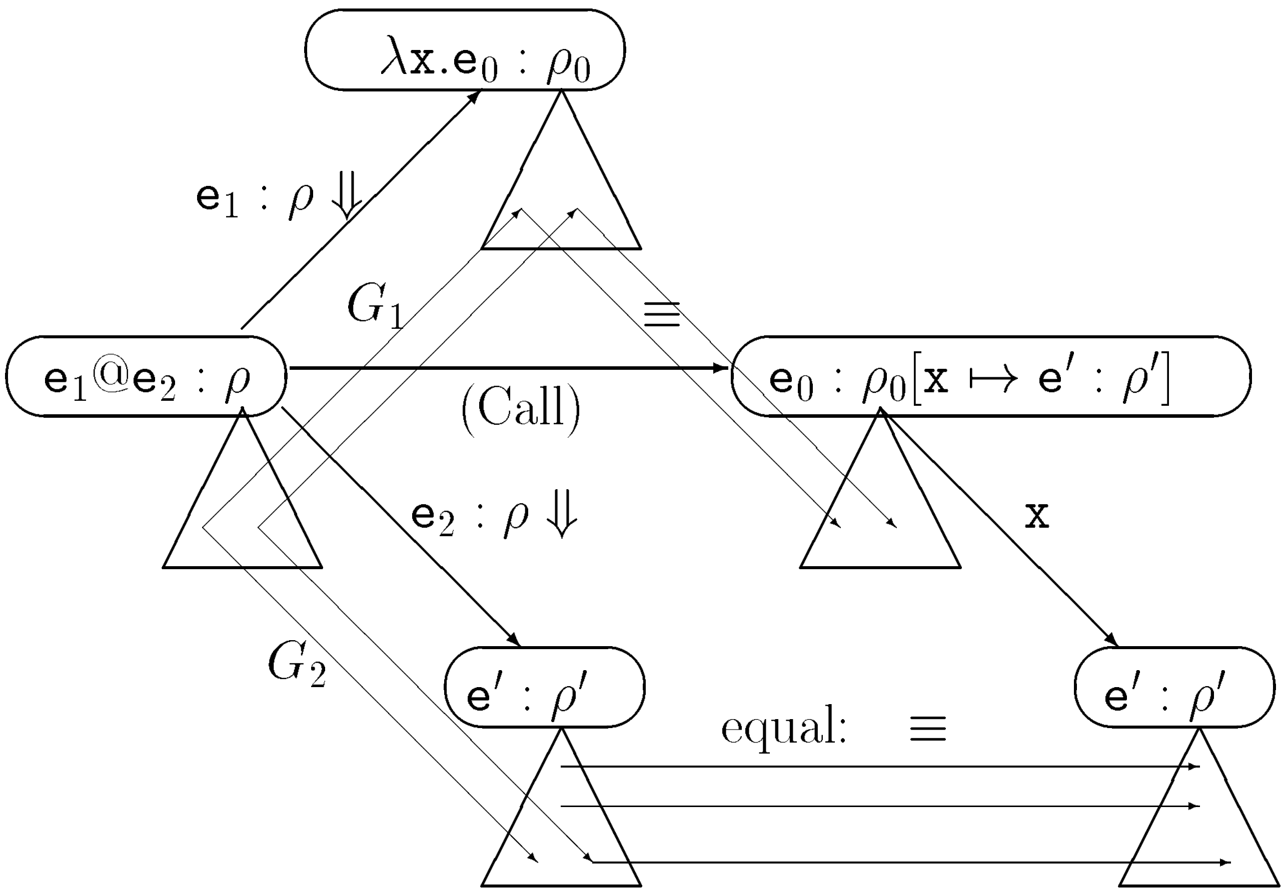
Notations:

$G_1^{-\bullet}$ stands for $\{\, \mathtt{y} \xrightarrow{r} \mathtt{z} \mid \ \mathtt{y} \xrightarrow{r} \mathtt{z} \in G_1 \} \cup$
$\{\, \bullet \overset{\downarrow}{\rightarrow} \mathtt{z} \mid \ \bullet \xrightarrow{r} \mathtt{z} \in G_1 \}$

$G_2^{\bullet\mapsto\mathtt{x}}$ stands for $\{\, \mathtt{y} \xrightarrow{r} \mathtt{x} \mid \ \mathtt{y} \xrightarrow{r} \bullet \in G_2 \,\} \cup$
$\{\, \bullet \overset{\downarrow}{\rightarrow} \mathtt{x} \mid \ \bullet \xrightarrow{r} \bullet \in G_2 \,\}$

Data flow in a function call

$$\frac{\mathsf{e}_1 : \rho \Downarrow \lambda \mathtt{x}.\mathsf{e}_0 : \rho_0, G_1 \qquad \mathsf{e}_2 : \rho \Downarrow v_2, G_2}{\mathsf{e}_1 @ \mathsf{e}_2 : \rho \;\underset{c}{\rightarrow}\; \mathsf{e}_0 : \rho_0[\mathtt{x} \mapsto v_2], (G_1^{-\bullet} \cup G_2^{\bullet \mapsto \mathtt{x}})}$$



Problems for static analysis

$\surd$ How to **identify nontermination ?**

$\surd$ How **finitely to describe**

the entire "computation space" ?

$\surd$ What is a **looping computation ?**

(so "size-change condition" can be applied)

$\surd$ What is a **value decrease ?**

$\surd$ **How to construct**

size-change graphs from a program `P` ?

$\triangleright$ **What can call what** in application $\mathtt{e}_1$@$\mathtt{e}_2$?

(need to decide it statically)

Abstract interpretation of the semantics

Approximate semantics generates a **decidable superset of all possible run-time behaviours**.

References: "closure analysis" (Jones 1981) and (Palsberg, Schwartzbach 1995), also "0-CFA" (Shivers 1990s).

1. *Remove all environments*, giving judgements $\mathsf{e} \Downarrow \mathsf{e}$ and $\mathsf{e} \to \mathsf{e}'$.
2. *Variable lookup* rule: modify it to fetch possible values of `x` by referring globally to `P`, the program being analysed.
3. Main virtue: there are only *finitely many* judgements $\mathsf{e} \to \mathsf{e}'$ and $\mathsf{e} \Downarrow \mathsf{e}'$.
4. Thus it's computable by exhaustive enumeration.

Coarse, but it works OK in practice (till now).

We have more precise analyses in reserve, using regular tree grammars.

Evaluation and call rules (approximate):

1. Remove all environments

2. Change variable lookup

$$\frac{}{\lambda \mathtt{x}.\mathtt{e} \Downarrow \lambda \mathtt{x}.\mathtt{e}}$$

$$\frac{\mathtt{e}_1 @ \mathtt{e}_2 \in \mathit{subexp}(\mathtt{P}) \qquad \mathtt{e}_1 \Downarrow \lambda \mathtt{x}.\mathtt{e}_0 \qquad \mathtt{e}_2 \Downarrow v_2}{\mathtt{x} \Downarrow v_2}$$

$$\frac{\mathtt{e}_1 @ \mathtt{e}_2 \underset{c}{\rightarrow} \mathtt{e}' \qquad \mathtt{e}' \Downarrow v}{\mathtt{e}_1 @ \mathtt{e}_2 \Downarrow v}$$

$$\frac{}{\mathtt{e}_1 @ \mathtt{e}_2 \underset{r}{\rightarrow} \mathtt{e}_1}$$

$$\frac{}{\mathtt{e}_1 @ \mathtt{e}_2 \underset{d}{\rightarrow} \mathtt{e}_2}$$

$$\frac{\mathtt{e}_1 \Downarrow \lambda \mathtt{x}.\mathtt{e}_0 \qquad \mathtt{e}_2 \Downarrow v_2}{\mathtt{e}_1 @ \mathtt{e}_2 \underset{c}{\rightarrow} \mathtt{e}_0}$$

Approximate rules with graph generation

$$\frac{}{\lambda\mathtt{x}.\mathtt{e} \Downarrow \lambda\mathtt{x}.\mathtt{e}, id^{=}_{\lambda\mathtt{x}.\mathtt{e}}}$$

$$\frac{\mathtt{e}_1@\mathtt{e}_2 \in subexp(\mathtt{P}) \qquad \mathtt{e}_1 \Downarrow \lambda\mathtt{x}.\mathtt{e}_0, \_ \qquad \mathtt{e}_2 \Downarrow v_2, \_}{\mathtt{x} \Downarrow v_2, (\{\mathtt{x} \stackrel{=}{\rightarrow} \bullet\} \cup \{\mathtt{x} \stackrel{\Downarrow}{\rightarrow} \mathtt{y} \mid \mathtt{y} \in fv(v_2)\})}$$

$$\frac{\mathtt{e}_1@\mathtt{e}_2 \underset{c}{\rightarrow} \mathtt{e}', G' \qquad \mathtt{e} \Downarrow v, G}{\mathtt{e}_1@\mathtt{e}_2 \Downarrow v, (G'; G)}$$

$$\frac{}{\mathtt{e}_1@\mathtt{e}_2 \underset{r}{\rightarrow} \mathtt{e}_1, id^{\downarrow}_{\mathtt{e}_1}}$$

$$\frac{}{\mathtt{e}_1@\mathtt{e}_2 \underset{d}{\rightarrow} \mathtt{e}_2, id^{\downarrow}_{\mathtt{e}_2}}$$

$$\frac{\mathtt{e}_1 \Downarrow \lambda\mathtt{x}.\mathtt{e}_0, G_1 \qquad \mathtt{e}_2 \Downarrow v_2, G_2}{\mathtt{e}_1@\mathtt{e}_2 \underset{c}{\rightarrow} \mathtt{e}_0, (G_1^{-\bullet} \cup G_2^{\bullet\mapsto\mathtt{x}})}$$

Current status of this work

1. Main results all proved in proceedings, e.g.,
   - Environment CBV semantics $\equiv$ Plotkin's
   - Subexpression property
   - NIS property
   - Instrumented semantics *builds safe graphs*
   - Abstract interpretation encompasses *all reachable states* in exact semantics
   - Abstract interpretation generates *all the instrumented semantics graphs*
2. A *prototype implementation* has been programmed.
3. Several test runs, e.g., Ackermann's function:
   - 2. order with Church numerals
   - 1. order with Y combinator
   - higher-order with Y combinator
   - programs with mutual recursion, parameter interchange

Some remaining questions

1. How *powerful* is the method: Does it prove natural programs terminating ?
2. How does it relate to *typing* as a means of showing termination ?
3. How *complex* is the analysis *method*, as a function of program size ?
4. How *complex* are the *functions computed* by size-change terminating programs ?

Power of the size-change method

1. It looks pretty powerful, but an in-depth study is needed.
2. It *handles* (no special extensions):
   - Mutual recursion
   - Permutation of function arguments
   - Church numerals as control (but not as data – it's hard to recognise value decrease)
   - Other numerals *as control or as data* (if $\underline{n}$ is a subexpression of $\underline{n+1}$)
   - *Higher-order primitive recursion*, for instance 2nd-order Ackermann's function
   - *General recursion*, e.g., via the Y combinator. (Untypable!)

Conjecture: Any simply-typed program (sans types) is size-change terminating

Conjecture: Any expression of Gödel's System T (sans types) is size-change terminating