Reasoning about non-terminating programs

Keiko Nakata
Institute of Cybernetics, Tallinn
(Joint work with Tarmo U.)

1 Feburary 2009

Summary of the talk

I will introduce some operational semantics for the While language, inductively defined as well as coinductively defined, to reason about terminating as well as non-terminating program executions.

Motivation and goal

There are many interesting terminating programs.
There are interesting non-terminating programs, e.g. server programs.

Our goal is to design a logic which can talk about

- whether an execution is definitely terminating or definitely non-terminating,

- observational behaviour of non-terminating executions, e.g. a program execution alternately prints “hello” and “bye” infinitely often.

- secure information flow analysis on non-terminating executions

The While language

$$
\begin{array}{rrcl}
\textit{Integers} & i & ::= & 1 \mid 2 \mid 3 \mid \dots \\
\textit{Expressions} & a & ::= & i \mid x \mid a + a \mid a - a \mid a < a \mid \dots \\
\textit{Statements} & s & ::= & \texttt{skip} \mid x := a \mid s; s \\
 & & \mid & \texttt{if}\ a\ \texttt{then}\ s\ \texttt{else}\ s \mid \texttt{while}\ (a)\ \{s\} \\
\textit{States} & \sigma & \in & \textit{Vars} \to \textit{Int}
\end{array}
$$

$\sigma \vdash a \Downarrow v$ means a evaluates to v under σ.
E.g. $(x : 1, y : 2) \vdash x + y \Downarrow 3$

$\sigma \vdash a \Downarrow \texttt{tt}$ means a evaluates to a truth value, i.e. non-zero.
E.g. $(x : 1) \vdash x < 3 \Downarrow \texttt{tt}$

$\sigma \vdash a \Downarrow \texttt{ff}$ means a evaluates to a false value, i.e. zero.
E.g. $(x : 1) \vdash x > 3 \Downarrow \texttt{ff}$

Operational semantics (inductively defined)

$s : \sigma \to \sigma'$ means an execution of s transfers σ to σ'.

$$\frac{}{\texttt{skip} : \sigma \to \sigma} \qquad \frac{\sigma \vdash a \Downarrow v}{x := a : \sigma \to \sigma[x \mapsto v]}$$

$$\frac{s_1 : \sigma \to \sigma' \quad s_2 : \sigma' \to \sigma''}{s_1 ; s_2 : \sigma \to \sigma''}$$

$$\frac{\sigma \vdash a \Downarrow \texttt{tt} \quad s_1 : \sigma \to \sigma'}{\texttt{if}\ a\ \texttt{then}\ s_1\ \texttt{else}\ s_2 : \sigma \to \sigma'} \qquad \frac{\sigma \vdash a \Downarrow \texttt{ff} \quad s_2 : \sigma \to \sigma'}{\texttt{if}\ a\ \texttt{then}\ s_1\ \texttt{else}\ s_2 : \sigma \to \sigma'}$$

$$\frac{\sigma \vdash a \Downarrow \texttt{ff}}{\texttt{while}\ (a)\ \{s\} : \sigma \to \sigma} \qquad \frac{\sigma \vdash a \Downarrow \texttt{tt} \quad s : \sigma \to \sigma' \quad \texttt{while}\ (a)\ \{s\} : \sigma' \to \sigma''}{\texttt{while}\ (a)\ \{s\} : \sigma \to \sigma''}$$

An example

$$\texttt{while}\ (x < 3)\ \{y := y * y; x := x + 1\}$$

$$\dfrac{s' : (1,2) \to (2,4) \qquad \dfrac{s' : (2,4) \to (3,16) \quad s : (3,16) \to (3,16)}{s : (2,4) \to (3,16)}}{s : (x : 1, y : 2) \to (3,16)}$$

where s abbreviates $\texttt{while}\ (x < 3)\ \{y := y * y; x := x + 1\}$
and s' abbreviates $y := y * y; x := x + 1$.

Operational semantics (coinductively defined)

$s :: \sigma \rightarrow \sigma'$ means an execution of s transfers σ to σ'.

$$\frac{}{\texttt{skip} :: \sigma \rightarrow \sigma} \qquad \frac{\sigma \vdash a \Downarrow v}{x := a :: \sigma \rightarrow \sigma[x \mapsto v]}$$

$$\frac{s_1 :: \sigma \rightarrow \sigma' \quad s_2 :: \sigma' \rightarrow \sigma''}{s_1 ; s_2 :: \sigma \rightarrow \sigma''}$$

$$\frac{\sigma \vdash a \Downarrow \texttt{tt} \quad s_1 :: \sigma \rightarrow \sigma'}{\texttt{if}\ a\ \texttt{then}\ s_1\ \texttt{else}\ s_2 :: \sigma \rightarrow \sigma'} \qquad \frac{\sigma \vdash a \Downarrow \texttt{ff} \quad s_2 :: \sigma \rightarrow \sigma'}{\texttt{if}\ a\ \texttt{then}\ s_1\ \texttt{else}\ s_2 :: \sigma \rightarrow \sigma'}$$

$$\frac{\sigma \vdash a \Downarrow \texttt{ff}}{\texttt{while}\ (a)\ \{s\} :: \sigma \rightarrow \sigma} \qquad \frac{\begin{array}{c}\sigma \vdash a \Downarrow \texttt{tt} \\ s :: \sigma \rightarrow \sigma' \quad \texttt{while}\ (a)\ \{s\} :: \sigma' \rightarrow \sigma''\end{array}}{\texttt{while}\ (a)\ \{s\} :: \sigma \rightarrow \sigma''}$$

Inductive semantics vs coinductive semantics

The two semantics differ in derivation trees they admit.

- Inductive semantics demands complete derivation trees.

$\Rightarrow$ We can only construct finite trees, which represent finite executions.

- Coinductive semantics needs derivation trees that can be built on demand, or lazily.

$\Rightarrow$ We can construct potentially infinitely growing trees lazily, thus coinductive semantics can express both finite and infinite executions.

Ref. We can program with infinite lists, i.e. streams, in Haskell, but not in OCaml (unless we use the lazy/force operators).

Coinductive semantics

The coinductive semantics characterizes both terminating and non-terminating execution.

It permits all derivation trees that the inductive semantics permits and more.

An example (1)

$$\texttt{while}\ (\texttt{true})\ \{x := 1\}$$

$$\dfrac{x := 1 :: (1) \to (1) \qquad \dfrac{x := 1 :: (1) \to (1) \qquad \dfrac{\cdots}{s :: (x : 1) \to (1)}}{s :: (1) \to (1)}}{s :: (x : 1) \to (1)}$$

where s abbreviates $\texttt{while}\ (\texttt{true})\ \{x := 1\}$.

Reminder:

$$\dfrac{\sigma \vdash a \Downarrow \texttt{tt} \qquad s :: \sigma \to \sigma' \quad \texttt{while}\ (a)\ \{s\} :: \sigma' \to \sigma''}{\texttt{while}\ (a)\ \{s\} :: \sigma \to \sigma''}$$

An example of diverging execution (2)

$$\texttt{while}\ (\texttt{true})\ \{x := x + 1\}$$

$$\dfrac{x := 1 :: (1) \to (2) \qquad \dfrac{x := 1 :: (2) \to (3) \qquad \dfrac{\dfrac{x := 1 :: (i) \to (i+1) \quad s :: (i+1) \to (?)}{\cdots}}{s :: (3) \to (?)}}{s :: (2) \to (?)}}{s :: (x : 1) \to (?)}$$

where s abbreviates $\texttt{while}\ (\texttt{true})\ \{x := x + 1\}$.

Reminder:

$$\dfrac{\sigma \vdash a \Downarrow \texttt{tt} \qquad s :: \sigma \to \sigma' \quad \texttt{while}\ (a)\ \{s\} :: \sigma' \to \sigma''}{\texttt{while}\ (a)\ \{s\} :: \sigma \to \sigma''}$$

Non-determinism

We can deduce, for any integer i,
$\texttt{while}\ (\texttt{true})\ \{x := x + 1\} :: (x : 1) \to (i)$

The post state is deterministic for terminating executions,
but is non-deterministic for non-terminating executions.

A program may reach any state after an infinite execution.

Because the program cannot reach such a state!

The coinductive semantics might not be informative enough to talk about interesting properties of non-terminating executions.

Operational semantics with traces

We extend our operational semantics with traces, or sequences of states.

$$\begin{array}{lccl} \mathit{States} & \sigma & \in & \mathit{Vars} \rightarrow \mathit{Int} \\ \mathit{Traces} & \tau & ::= & () \mid \sigma :: \tau \end{array}$$

$$x := 1; y := x + 1; x := x + y$$

$$x := 1; y := x + 1; x := x + y :: (0,0) \rightarrow [(0,0); (1,0); (1,2); (3,2)]$$

Operational semantics with traces (1)

inductively defined

$s : \sigma \longrightarrow \tau$ means execution of s starting at σ goes through τ.

$s : \tau \twoheadrightarrow \tau'$ means s transfers τ to τ', i.e. τ followed by execution of s yields τ'.

Operational semantics with traces (2)

inductively defined

$$\frac{s : \sigma \to \tau}{s : [\sigma] \twoheadrightarrow \tau} \qquad \frac{s : \tau \twoheadrightarrow \tau'}{s : \sigma :: \tau \twoheadrightarrow \sigma :: \tau'}$$

$$\frac{}{\texttt{skip} : \sigma \to [\sigma]} \qquad \frac{\sigma \vdash a \Downarrow v}{x := a : \sigma \to [\sigma; (\sigma[x \mapsto v])]}$$

$$\frac{s_1 : \sigma \to \tau \quad s_2 : \tau \twoheadrightarrow \tau'}{s_1 ; s_2 : \sigma \to \tau'}$$

$$\frac{\sigma \vdash a \Downarrow \texttt{tt} \quad s_1 : \sigma :: [\sigma] \twoheadrightarrow \tau}{\texttt{if}\ a\ \texttt{then}\ s_1\ \texttt{else}\ s_2 : \sigma \to \tau'} \qquad \frac{\sigma \vdash a \Downarrow \texttt{ff} \quad s_2 : \sigma :: [\sigma] \twoheadrightarrow \tau}{\texttt{if}\ a\ \texttt{then}\ s_1\ \texttt{else}\ s_2 : \sigma \to \tau}$$

$$\frac{\sigma \vdash a \Downarrow \texttt{ff}}{\texttt{while}\ (a)\ \{s\} : \sigma \to \sigma :: [\sigma]} \qquad \frac{\sigma \vdash a \Downarrow \texttt{tt} \quad s : \sigma :: [\sigma] \twoheadrightarrow \tau \quad \texttt{while}\ (a)\ \{s\} : \tau \twoheadrightarrow \tau'}{\texttt{while}\ (a)\ \{s\} : \sigma \to \tau'}$$

An example

$$x := 1; y := x + 1; x := x + y$$

$$\dfrac{y := x+1 : (1,0) \rightarrow [(1,0);(1,2)] \quad \dfrac{x := x+y : (1,2) \twoheadrightarrow [(1,2);(3,2)]}{x := x+y : [(1,0);(1,2)] \twoheadrightarrow [(1,0);(1,2);(3,2)]}}{s' : (1,0) \rightarrow [(1,0);(1,2);(3,2)]}$$

$$\dfrac{x := 1 : (0,0) \rightarrow [(0,0);(1,0)] \quad \dfrac{s' : (1,0) \rightarrow [(1,0);(1,2);(3,2)]}{s' : [(0,0);(1,0)] \twoheadrightarrow [(0,0);(1,0);(1,2);(3,2)]}}{s : (x:0, y:0) \rightarrow [(0,0);(1,0);(1,2);(3,2)]}$$

where s abbreviates $x := 1; y := x + 1; x := x + y$
and s' abbreviates $y := x + 1; x := x + y$

Operational semantics with traces (coinductively defined)

$$\frac{s :: \sigma \to \tau}{s :: [\sigma] \twoheadrightarrow \tau} \qquad \frac{s :: \tau \twoheadrightarrow \tau'}{s :: \sigma :: \tau \twoheadrightarrow \sigma :: \tau'}$$

$$\frac{}{\texttt{skip} :: \sigma \to [\sigma]} \qquad \frac{\sigma \vdash a \Downarrow v}{x := a :: \sigma \to [\sigma; (\sigma[x \mapsto v])]}$$

$$\frac{s_1 :: \sigma \to \tau \quad s_2 :: \tau \twoheadrightarrow \tau'}{s_1 ; s_2 :: \sigma \to \tau'}$$

$$\frac{\sigma \vdash a \Downarrow \texttt{tt} \quad s_1 :: \sigma :: [\sigma] \twoheadrightarrow \tau}{\texttt{if}\ a\ \texttt{then}\ s_1\ \texttt{else}\ s_2 :: \sigma \to \tau'} \qquad \frac{\sigma \vdash a \Downarrow \texttt{ff} \quad s_2 :: \sigma :: [\sigma] \twoheadrightarrow \tau}{\texttt{if}\ a\ \texttt{then}\ s_1\ \texttt{else}\ s_2 :: \sigma \to \tau}$$

$$\frac{\sigma \vdash a \Downarrow \texttt{ff}}{\texttt{while}\ (a)\ \{s\} :: \sigma \to \sigma :: [\sigma]} \qquad \frac{\begin{array}{c}\sigma \vdash a \Downarrow \texttt{tt} \\ s :: \sigma :: [\sigma] \twoheadrightarrow \tau \quad \texttt{while}\ (a)\ \{s\} :: \tau \twoheadrightarrow \tau'\end{array}}{\texttt{while}\ (a)\ \{s\} :: \sigma \to \tau'}$$

Examples

The coinductive semantics permits all derivation trees that the inductive semantics permits and more.

$\texttt{while}\ (\texttt{true})\ \{x := 1\}$

$\texttt{while}\ (\texttt{true})\ \{x := 1\} :: (x : 1) \rightarrow [(1); (1); (1); (1); \ldots]$

$\texttt{while}\ (\texttt{true})\ \{x := x + 1\}$

$\texttt{while}\ (\texttt{true})\ \{x := x + 1\} :: (x : 1) \rightarrow [(1); (1); (2); (2); (3); (3); \ldots]$

Determinism

For terminating executions traces are deterministic.

For non-terminating executions traces are deterministic up to the bisimulation relation,
i.e. traces are deterministic up to finite observations.

Design issues

- `skip` preserves traces, or `skip` is an identity of sequential composition.

for all s, $\texttt{skip}; s :: \tau \twoheadrightarrow \tau'$ if and only if $s :: \tau \twoheadrightarrow \tau'$

- Checking conditionals increases traces by one.

$\texttt{while}\ (\texttt{true})\ \{\texttt{skip}\} :: (1) \rightarrow [(1); (1); (1); \ldots]$

On going work

We are designing an axiomatic semantics for the coinductive operational semantics with traces.

Thank you for your attention.