Coinductive big-step semantics and Hoare logics for nontermination

Tarmo Uustalu, Inst of Cybernetics, Tallinn
joint work with Keiko Nakata

COST Rich Models Toolkit meeting,
Madrid, 17–18 October 2013
Motivation

- Standard big-step semantics and Hoare-style program logics do not account for nonterminating program behaviors.
- But we want to reason about nonterminating behaviors (esp in the context of reactive programming).
- Solution: Devise semantics operating on coinductively defined semantic entities (traces, resumptions) and logics for reasoning about them.
- Original inspiration: Leroy’s work on coinductive big-step semantics in CompCert.
Constructivity

- An angle: We do programming language theory in a constructive setting (type theory). We are happy to apply non-constructive principles, where necessary, but want to notice when we do.
- This is the setting of proof assistants/dependently typed programming languages like Coq/Agda.
- It is nice to have semantics executable.
- Constructive proofs are (extract to) programs.
- In a constructive setting, indications of suboptimal designs sometimes surface earlier than in a classical setting.
This talk

- I show a trace-based big-step semantics and Hoare logic for the simple imperative language While, featuring nontermination from loops.
- Could add recursive procedures or consider a (higher-order) functional language.
- Important extensions: interactive input-output, shared-variable concurrency.
- Here all intermediate states are tracked, generally may want to single out a class of observable event, identify weakly bisimilar traces.
Big-step semantics for nontermination
Syntax of While, states

- Statements are defined inductively by the grammar:
  \[ s ::= x := e \mid \text{skip} \mid s_0; s_1 \mid \text{if } e \text{ then } s_t \text{ else } s_f \mid \text{while } e \text{ do } s_t \]

- States \( \sigma \) are assignments of integers to variables names.
Trace-based big-step semantics

- We describe both converging and diverging behaviors by one single evaluation relation defined coinductively.
- The leading idea is to avoid any need to decide if a computation converges or diverges.
- This requires (at least) appreciating that computations take time.
- Cf. Leroy, Grall: two separate evaluation relations, or one single evaluation relation, but no productivity.
- We will record all intermediate states, corresponding to the idea of making all intermediate states observable.
Traces

- We define traces coinductively (as non-empty possibly infinite lists of states) by

\[
\begin{align*}
\sigma : state & \quad \langle \sigma \rangle : \text{trace} \\
\sigma : state & \quad \tau : trace \\
\sigma : state & \quad \tau : trace \\
\end{align*}
\]

- (Strong) bisimilarity is defined coinductively by

\[
\begin{align*}
\langle \sigma \rangle \sim \langle \sigma \rangle \\
\tau \sim \tau_* \\
\sigma :: \tau \sim \sigma :: \tau_*
\end{align*}
\]

- We want to consider bisimilar traces as equal, so require that all predicates and functions on traces are stable under bisimilarity.

- Classically, bisimilarity is nothing but equality. Constructively, one has to be more careful...
Big-step semantics

- Evaluation relates an (initial) state to a trace and is defined coinductively:

\[
\begin{align*}
(x := e, \sigma) &\Rightarrow \sigma :: \langle \sigma[x \mapsto \llbracket e \rrbracket \sigma] \rangle \\
(skip, \sigma) &\Rightarrow \langle \sigma \rangle \\
(s_0, \sigma) &\Rightarrow \tau \quad (s_1, \tau) \xrightarrow{\ast} \tau' \\
(s_0; s_1, \sigma) &\Rightarrow \tau'
\end{align*}
\]

- \(\sigma \models e\) \(\quad (s_t, \sigma :: \langle \sigma \rangle) \xrightarrow{\ast} \tau\) \(\quad (s_f, \sigma :: \langle \sigma \rangle) \not\models e\) 

\[
\begin{align*}
(if \ e \ then \ s_t \ else \ s_f, \sigma) &\Rightarrow \tau \\
(if \ e \ then \ s_t \ else \ s_f, \sigma) &\Rightarrow \tau \\
(while \ e \ do \ s_t, \tau) &\Rightarrow \tau' \\
(while \ e \ do \ s_t, \sigma) &\Rightarrow \tau' \\
(while \ e \ do \ s_t, \sigma) &\Rightarrow \sigma :: \langle \sigma \rangle
\end{align*}
\]
Big-step semantics ctd

- Extended evaluation relates an (already accumulated) trace to a (total) trace, is also defined coinductively:

\[
\begin{align*}
(s, \sigma) \Rightarrow & \quad \tau \\
(s, \langle \sigma \rangle) & \Rightarrow^* \tau \\
(s, \tau) & \Rightarrow^* \tau' \\
(s, \sigma :: \tau) & \Rightarrow^* \sigma :: \tau'
\end{align*}
\]

(coinductive prefix closure of evaluation)

- Design choice: evaluation of an expression to assign/updating of a variable and evaluation of a guard constitute take unit time.

- Consideration: every loop always progresses, e.g., we have

\[(\text{while true do skip, } \sigma) \Rightarrow \sigma :: \sigma :: \ldots .\]

As a minimum, evaluating a guard to true must take unit time.

- Our choice of what takes unit time gives agreement up to bisimilarity with the natural small-step semantics.
Big-step semantics ctd

- Evaluation is stable under bisimilarity:
  - If \((s, \sigma) \Rightarrow \tau\) and \(\tau \sim \tau_*\), then \((s, \sigma) \Rightarrow \tau_*\).
    (Proof by coinduction.)

- Evaluation is deterministic (up to bisimilarity):
  - If \((s, \sigma) \Rightarrow \tau\) and \((s, \sigma) \Rightarrow \tau_*\), then \(\tau \sim \tau_*\).
    (Proof by coinduction.)
Big-step semantics, functional-style

- How to prove that evaluation is total, i.e., that, for any \( s, \sigma \), there exists \( \tau \) such that \((s, \sigma) \Rightarrow \tau\)?
- Constructively, we must explicitly produce a witnessing \( \tau \) from \( s, \sigma \).
- This means defining evaluation and extended evaluations as functions.

**Evaluation:**

\[
\begin{align*}
[x := e] \sigma &= \sigma :: \langle \sigma[x \mapsto [e] \sigma] \rangle \\
\text{[skip]} \sigma &= \langle \sigma \rangle \\
[s_0; s_1] \sigma &= [s_1]^* ([s_0] \sigma) \\
[\text{if } e \text{ then } s_t \text{ else } s_f] \sigma &= [s_t]^* ([s_f]^* (\sigma :: \langle \sigma \rangle)) \quad \sigma \models e \\
&= [s_f]^* (\sigma :: \langle \sigma \rangle) \quad \sigma \nmodels e \\
[\text{while } e \text{ do } s_t] \sigma &= [\text{while } e \text{ do } s_t]^* ([s_t]^* (\sigma :: \langle \sigma \rangle)) \quad \sigma \models e \\
&= \sigma :: \langle \sigma \rangle \quad \sigma \nmodels e
\end{align*}
\]

(almost structurally recursive, but not the clauses for while)
Big-step semantics, functional-style ctd

- Extension:

\[
\begin{align*}
  k^* (\langle \sigma \rangle) &= k \sigma \\
  k^* (\sigma :: \tau) &= \sigma :: (k^* \tau)
\end{align*}
\]

(guided corecursion)

- Evaluation is total:

  \[ (s, \sigma) \Rightarrow [s] \sigma. \]

  (By coinduction.)
Small-step semantics

- Single-step reduction is defined in the standard fashion inductively:

\[ (x := e, \sigma) \rightarrow (\text{skip}, \sigma[x \mapsto \llbracket e \rrbracket \sigma]) \]

\[ (s_0, \sigma) \rightarrow \sigma' \quad (s_0; s_1, \sigma) \rightarrow (s_1, \sigma') \quad (s_0; s_1, \sigma) \rightarrow (s'_0; s_1, \sigma') \]

\[ \sigma \models e \quad \sigma \not\models e \]

\[ (\text{if } e \text{ then } s_t \text{ else } s_f, \sigma) \rightarrow (s_t, \sigma) \quad (\text{if } e \text{ then } s_t \text{ else } s_f, \sigma) \rightarrow (s_f, \sigma) \]

\[ \sigma \models e \]

\[ (\text{while } e \text{ do } s_t, \sigma) \rightarrow (s_t; \text{while } e \text{ do } s_t, \sigma) \]

\[ \sigma \not\models e \]

\[ (\text{while } e \text{ do } s_t, \sigma) \rightarrow (\text{skip}, \sigma) \]
Small-step semantics ctd

- Maximal multi-step reduction is defined coinductively by:

  \[
  (s, \sigma) \rightarrow \sigma' \\
  (s, \sigma) \rightsquigarrow \langle \sigma' \rangle \\
  (s, \sigma) \rightarrow (s', \sigma') \\
  (s', \sigma') \rightsquigarrow \tau \\
  (s, \sigma) \rightsquigarrow \sigma :: \tau
  \]

- Similarly to evaluation, maximal multi-step reduction is stable under bisimilarity and deterministic up to bisimilarity.
Big-step vs small-step semantics

- Big-step semantics is sound wrt. small-step semantics:
  - If \((s, \sigma) \Rightarrow \tau\), then \((s, \sigma) \Rightarrow^* \tau\).
    (Proof by coinduction.)

- It is also complete:
  - If \((s, \sigma) \Rightarrow^* \tau\), then \((s, \sigma) \Rightarrow \tau\).
  - If \((s, \tau) \Rightarrow^* \tau'\), then \((s, \tau) \Rightarrow^* \tau'\).
    (Proof by mutual coinduction.)

  (Here \(\Rightarrow^*\) is the coinductive prefix closure of \(\Rightarrow\).)

- The following midpoint lemma is required:
  - If \((s_0; s_1, \sigma) \Rightarrow \tau'\), then there exists \(\tau\) such that
    \((s_0, \sigma) \Rightarrow \tau\) and \((s_1, \tau) \Rightarrow^* \tau'\).
    (Proof: \(\tau\) is constructed by corecursion and the two conditions are proved by coinduction.)
Hoare logic
Hoare logic

- We present a Hoare logic corresponding to the trace-based big-step semantics.
- This uses assertions on both states and traces. We don’t define a fixed syntax of assertions, instead we use predicates. Trace predicates must be stable under bisimilarity.
- For trace assertions (predicates), we need some interesting connectives (operations on predicates).
- Proofs are defined inductively, as in standard Hoare logic.
Assertion connectives

- Some trace predicates:

\[ \sigma \models U \]
\[ \langle \sigma \rangle \models \langle U \rangle \]

\[ \sigma :: \langle \sigma[x \mapsto [e] \sigma] \rangle \models [x \mapsto e] \]
\[ \sigma :: \langle \sigma \rangle \models \Delta \]

\[ \langle \sigma \rangle \models finite \]
\[ \tau \models finite \]
\[ \sigma :: \tau \models finite \]
\[ \tau \models infinite \]
\[ \sigma :: \tau \models infinite \]

- All these predicates are stable under bisimilarity.
Assertion connectives ctd

- Chop and dagger:

\[
\begin{align*}
\tau_0 & \models P \quad \tau \models_{\tau_0} Q \\
\tau & \models P^{**} Q \\
\langle \sigma \rangle & \models P^\dagger \\
\tau & \models P^\dagger
\end{align*}
\]

where

\[
\begin{align*}
\langle \sigma \rangle & \models Q \\
\langle \sigma \rangle & \models_{\langle \sigma \rangle} Q \\
\sigma : \tau & \models Q \\
\sigma : \tau & \models_{\langle \sigma \rangle} Q \\
\tau & \models_{\tau_0} Q \\
\sigma : \tau & \models_{\sigma : \tau_0} Q
\end{align*}
\]

- Cf. interval temporal logic, B. Mosztowski

- If \( P, Q \) are stable under bisimilarity, then so is \( P^{**} Q \). If \( P \) is stable under bisimilarity, so is \( P^\dagger \).

- If \( \tau \) is infinite and \( \tau \models P \), then \( \tau \models P^{**} Q \) for any \( Q \)!
Hoare proofs

Proofs are defined inductively by the rules

\[
\begin{align*}
\{U\} \ x := e \ & \{\langle U\rangle \mapsto [x \mapsto e]\} \\
\{U\} \ \text{skip} \ & \{\langle U\rangle\} \\
\{U\} \ s_0 \ {P} \ & \{V\} \ s_1 \ {Q} \\
\{e \land U\} \ s_t \ {P} \ & \{\neg e \land U\} \ s_f \ {P} \\
\{U\} \ \text{if } e \ \text{then } s_t \ \text{else } s_f \ & \{\triangle \mapsto P\} \\
U \models I \ & \{e \land I\} \ s_t \ {P} \mapsto \{\langle I\rangle\} \\
\{U\} \ \text{while } e \ \text{do} \ s_t \ & \{(\triangle \mapsto P)^{\dagger} \mapsto \triangle \mapsto \langle \neg e\rangle\} \\
U \models U' \ & \{U'\} \ s \ {P'} \ & P' \models P \\
\{U\} \ s \ {P}\end{align*}
\]
Soundness

- The Hoare logic is sound.

- If $\{U\} \ s \ \{P\}$, then $\sigma \models U$ and $(s, \sigma) \Rightarrow \tau$ imply $\tau \models P$.

  (By induction on $\{U\} \ s \ \{P\}$, subordinate coinduction in several cases.)
Completeness

- The Hoare logic is also complete.

- If, for any $\sigma, \tau, \sigma \models U$ and $(s, \sigma) \Rightarrow \tau$ imply $\tau \models P$, then $\{U\} s \{P\}$.

- To prove this, one defines for any $s, U$, a trace predicate $sp(s, U)$ (the strongest postcondition), by structural recursion on $s$.

- Now completeness follows from these lemmata:
  - $\{U\} s \{sp(s, U)\}$. (By induction on $s$).
  - If $\tau \models sp(s, U)$, then $(s, \text{hd } \tau) \Rightarrow \tau$. (By induction on $s$.)

  The latter gives as an immediate corollary:
  - If, for all $\sigma, \tau, \sigma \models U$ and $(s, \sigma) \Rightarrow \tau$ imply $\tau \models P$, then $sp(s, U) \models P$. 

Strongest postconditions

- **Strongest postconditions:**

\[
\begin{align*}
sp(x := e, U) &= \langle U \rangle^{*} \left[ x \mapsto e \right] \\
sp(\text{skip}, U) &= \langle U \rangle \\
sp(s_0; s_1, U) &= sp(s_0, U)^{*} sp(s_1, \text{Last}(sp(s_0, U))) \\
sp(\text{if } e \text{ then } s_t \text{ else } s_f, U) &= \langle U \rangle^{*} \bigtriangleup^{*} (sp(s_t, e \land U) \lor sp(s_f, \neg e \land U)) \\
sp(\text{while } e \text{ do } s_t, U) &= \langle U \rangle^{*} (\bigtriangleup^{*} sp(s_t, e \land \text{Inv}(e, s_t, U)))^{\dagger} \bigtriangleup^{*} \langle \neg e \rangle
\end{align*}
\]

\[
\frac{\tau \downarrow \sigma}{\sigma \models P} \quad \frac{\tau \models P}{\sigma \models \text{Last } P}
\]

\[
\begin{align*}
\sigma \models U & \quad \sigma \models \text{Last}(sp(s, e \land \text{Inv}(e, s, U))) \\
& \quad \sigma \models \text{Inv}(e, s, U)
\end{align*}
\]
Embedding of standard Hoare logic

- The standard Hoare logic embeds into the Hoare logic for trace-based semantics.
  - If \( \{U\} s \{Z\} \) in the standard (partial correctness) Hoare logic, then \( \{U\} s \{true \circ \langle Z\rangle\} \).
    (Could go via soundness and completeness. But there is a direct proof by induction.)

- Similarly, the total-correctness Hoare logic also embeds into our logic.
  - If \( \{U\} s \{Z\} \) in the total correctness Hoare logic, then \( \{U\} s \{finite \circ \langle Z\rangle\} \).