COMP3610/6361
Principles of Programming Languages

Peter Höfner

Sep 19, 2023

Section 14

Semantic Equivalence

Operational Semantics (Reminder)

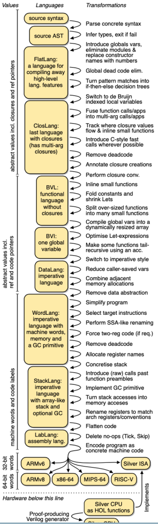
- describe how to evaluate programs
- a valid program is interpreted as sequences of steps
- small-step semantics
  - individual steps of a computation
  - more rules (compared to big-step)
  - allows to reason about non-terminating programs, concurrency, . . .
- big-step semantics
  - overall results of the executions
    'divide-and-conquer manner'
  - can be seen as relations
  - fewer rules, simpler proofs
  - no non-terminating behaviour
- allow non-determinism

Motivation

When are two programs considered the ‘same’

- compiler construction
- program optimisation
- refinement
- …

CakeML



Equivalence: Intuition I

$$l := !l + 2 \quad \stackrel{?}{\simeq} \quad l := !l + (1+1) \quad \stackrel{?}{\simeq} \quad l := !l + 1\,;\, l := !l + 1$$

- are these expressions the same
- in what sense
  - different abstract syntax trees
  - different reduction sequences
- in any (sequential) program one could replace one by the other without affecting the result

Note: mathematicians often take these equivalences for granted

Equivalence: Intuition II

$l := 0 \,;\, 4 \quad \overset{?}{\simeq} \quad l := 1 \,;\, 3 + !l$

- produce same result (for all stores)
- cannot be replaced in an arbitrary context C

For example, let $C[\_] = \_ + \ !l$

$$C[l := 0 \,;\, 4] = (l := 0 \,;\, 4) + \ !l$$

$$\not\simeq$$

$$C[l := 1 \,;\, 3 + \ !l] = (l := 1 \,;\, 3 + \ !l) + \ !l$$

On the other hand $(l := \ !l + 2) \simeq (l := \ !l + 1 \,;\, l := \ !l + 1)$

Equivalence: Intuition III

From particular expressions to general laws

- $E_1 \; ; (E_2 \; ; E_3) \overset{?}{\simeq} (E_1 \; ; E_2) \; ; E_3$
- $(\textbf{if } E_1 \textbf{ then } E_2 \textbf{ else } E_3) \; ; E \overset{?}{\simeq} \textbf{if } E_1 \textbf{ then } E_2 \; ; E \textbf{ else } E_3 \; ; E$
- $E \; ; (\textbf{if } E_1 \textbf{ then } E_2 \textbf{ else } E_3) \overset{?}{\simeq} \textbf{if } E_1 \textbf{ then } E \; ; E_2 \textbf{ else } E \; ; E_3$
- $E \; ; (\textbf{if } E_1 \textbf{ then } E_2 \textbf{ else } E_3) \overset{?}{\simeq} \textbf{if } E \; ; E_1 \textbf{ then } E_2 \textbf{ else } E_3$

Australian National University

Exercise

$$\textbf{let val } x : \text{int ref} = \text{ref } 0 \textbf{ in } (\textbf{fn } y : \text{int} \Rightarrow (x := !x + y) \; ; \; !x) \textbf{ end}$$

$$\stackrel{?}{\simeq}$$

$$\textbf{let val } x : \text{int ref} = \text{ref } 0 \textbf{ in } (\textbf{fn } y : \text{int} \Rightarrow (x := !x - y) \; ; \; (0 - !x)) \textbf{ end}$$

Exercise II

Extend our language with location equality

$$op := \ \dots \mid =$$

(op =) $\dfrac{\Gamma \vdash E_1 : T \text{ ref} \qquad \Gamma \vdash E_2 : T \text{ ref}}{\Gamma \vdash E_1 = E_2 : \text{bool}}$

(op=1) $\langle l = l' \, , \, s \rangle \longrightarrow \langle b \, , \, s \rangle$ if $b = (l = l')$

(op=2) $\dots$

Exercise II

$$f \overset{?}{\simeq} g$$

for

$$\begin{aligned} f = {} & \textbf{let val } x : \text{int ref} = \text{ref } 0 \textbf{ in} \\ & \textbf{let val } y : \text{int ref} = \text{ref } 0 \textbf{ in} \\ & (\textbf{fn } z : \text{int ref} \Rightarrow \textbf{if } z = x \textbf{ then } y \textbf{ else } x) \\ & \ \textbf{end end} \end{aligned}$$

and

$$\begin{aligned} g = {} & \textbf{let val } x : \text{int ref} = \text{ref } 0 \textbf{ in} \\ & \textbf{let val } y : \text{int ref} = \text{ref } 0 \textbf{ in} \\ & (\textbf{fn } z : \text{int ref} \Rightarrow \textbf{if } z = y \textbf{ then } y \textbf{ else } x) \\ & \ \textbf{end end} \end{aligned}$$

Exercise II (cont'd)

$$f \overset{?}{\simeq} g \qquad \textbf{NO}$$

Consider $C[\_] = t\_$ with

$$\begin{aligned} t = \ & (\textbf{fn}\ h : (\text{int ref} \to \text{int ref}) \Rightarrow \\ & \quad \textbf{let val}\ z : \text{int ref} = \text{ref}\ 0\ \textbf{in}\ h\ (h\ z) = h\ z\ \textbf{end}) \end{aligned}$$

$$\langle t\ f\ ,\ s\rangle \longrightarrow^{*}\ ?$$
$$\langle t\ g\ ,\ s\rangle \longrightarrow^{*}\ ?$$

A ‘good’ notion of semantic equivalence

We might

- understand what a program *is*
- prove that some particular expressions to be equivalent (e.g. efficient algorithm vs. clear specification)
- prove the soundness of general laws for equational reasoning about programs
- prove some compiler optimisations are sound (see CakeML or CertiCos)
- understand the differences between languages

What does 'good' mean?

1. programs that result in observably-different values (for some store) must not be equivalent

$$
\begin{aligned}
&(\exists s, s_1, s_2, v_1, v_2. \\
&\quad \langle E_1 \, , \, s \rangle \longrightarrow^* \langle v_1 \, , \, s_1 \rangle \wedge \\
&\quad \langle E_2 \, , \, s \rangle \longrightarrow^* \langle v_2 \, , \, s_2 \rangle \wedge \\
&\quad v_1 \neq v_2) \\
&\Rightarrow E_1 \not\simeq E_2
\end{aligned}
$$

2. programs that terminate must not be equivalent to programs that do not terminate

What does 'good' mean?

3. $\simeq$ must be an equivalence relation, i.e.

reflexivity	$E \simeq E$
symmetry	$E_1 \simeq E_2 \Rightarrow E_2 \simeq E_1$
transitivity	$E_1 \simeq E_2 \wedge E_2 \simeq E_3 \Rightarrow E_1 \simeq E_3$

4. $\simeq$ must be a congruence, i.e,

   if $E_1 \simeq E_2$ then for any context C we must have $C[E_1] \simeq C[E_2]$

   (for example, $(E_1 \simeq E_2) \Rightarrow (E_1 \,;\, E \simeq E_2 \,;\, E)$)

5. $\simeq$ should relate as many programs as possible

– an equivalence relation that is a congruence is sometimes called *congruence relation*
– this semantic equivalence, is called observable operational or contextual equivalence
– congruence proofs are often tedious, and incredible hard when it comes to recursion

Semantic Equivalence for (simple) Typed IMP

Definition

$E_1 \simeq_\Gamma^T E_2$ iff for all stores s with $\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(s)$ we have

$$\Gamma \vdash E_1 : T \quad \text{and} \quad \Gamma \vdash E_2 : T \ ,$$

and either

(i) $\langle E_1\ ,\ s\rangle \longrightarrow^\omega$ and $\langle E_2\ ,\ s\rangle \longrightarrow^\omega$, or

(ii) for some v, s' we have $\langle E_1\ ,\ s\rangle \longrightarrow^* \langle v\ ,\ s'\rangle$ and $\langle E_2\ ,\ s\rangle \longrightarrow^* \langle v\ ,\ s'\rangle$.

$\longrightarrow^\omega$: infinite sequence

$\longrightarrow^*$: finite sequence (reflexive transitive closure)

Justification

Part (ii) requires same value v and same store s'. If a program generates different stores, we can distinguish them using contexts:

- If $T =$ unit then $C[\_] = \_ \,;\, !l$
- If $T =$ bool then $C[\_] =$ **if** $\_$ **then** $!l$ **else** $!l$
- If $T =$ int then $C[\_] = (l_1 := \_ \,;\, !l)$

Equivalence Relation

Theorem
The relation $\simeq_\Gamma^T$ is an equivalence relation.

Proof.
trivial $\square$

Congruence for (simple) Typed IMP

contexts are:

$$
\begin{aligned}
C[\_] \quad ::= \; & \_ \; op \; E_2 \mid E_1 \; op \; \_ \mid \\
& \textbf{if } \_ \textbf{ then } E_2 \textbf{ else } E_3 \mid \\
& \textbf{if } E_1 \textbf{ then } \_ \textbf{ else } E_3 \mid \\
& \textbf{if } E_1 \textbf{ then } E_2 \textbf{ else } \_ \mid \\
& l := \_ \mid \\
& \_ \; ; E_2 \mid E_1 \; ; \; \_ \\
& \textbf{while } \_ \textbf{ do } E_2 \mid \textbf{while } E_1 \textbf{ do } \_
\end{aligned}
$$

Definition

The relation $\simeq_\Gamma^T$ has the *congruence property* if, for all E_1 and E_2, whenever $E_1 \simeq_\Gamma^T E_2$ we have for all C and T', if $\Gamma \vdash C[E_1] : T'$ and $\Gamma \vdash C[E_2] : T'$ then

$$C[E_1] \simeq_\Gamma^{T'} C[E_2]$$

Congruence Property

Theorem (Congruence for (simple) typed IMP)

The relation $\simeq^T_\Gamma$ has the congruence property.

Proof.

By case distinction, considering all contexts C. □

For each context C (and arbitrary expression E and store s) consider the possible reduction sequence

$$\langle C[E]\,,\,s\rangle \longrightarrow \langle E_1\,,\,s_1\rangle \longrightarrow \langle E_2\,,\,s_2\rangle \longrightarrow \ldots$$

and deduce the behaviour of E:

$$\langle E\,,\,s\rangle \longrightarrow \langle \hat{E}_1\,,\,s_1\rangle \longrightarrow \ldots$$

Use $E \simeq^T_\Gamma E'$ find a similar reduction sequence of E' and use the reduction rules to construct a sequence of $C[E']$.

Proof of Congruence Property

Case $C = (l := \_)$

Suppose $E \simeq_{\Gamma}^{T} E'$, $\Gamma \vdash l := E : T'$ and $\Gamma \vdash l := E' : T'$.
By examination of the typing rule, we have $T =$ int and $T' =$ unit.
To show $(l := E) \simeq_{\Gamma}^{T'} (l := E')$ we have to show that for all stores s if dom$(\Gamma) \subseteq$ dom(s) then

- $\Gamma \vdash l := E : T'$, (obvious)
- $\Gamma \vdash l := E' : T'$,(obvious)
- and either
  - (i) $\langle l := E \, , \, s \rangle \longrightarrow^{\omega}$ and $\langle l := E' \, , \, s \rangle \longrightarrow^{\omega}$
  - (ii) for some v, s' we have $\langle l := E \, , \, s \rangle \longrightarrow^{*} \langle v \, , \, s' \rangle$ and $\langle l := E' \, , \, s \rangle \longrightarrow^{*} \langle v \, , \, s' \rangle$.

Proof of Congruence Property

Subcase $\langle l := E\ ,\ s\rangle \longrightarrow^{\omega}$

That is

$$\langle l := E\ ,\ s\rangle \longrightarrow \langle E_1\ ,\ s_1\rangle \longrightarrow \langle E_2\ ,\ s_2\rangle \longrightarrow \ldots$$

All these must be instances of Rule (assign2), with

$$\langle E\ ,\ s\rangle \longrightarrow \langle \hat{E}_1\ ,\ s_1\rangle \longrightarrow \langle \hat{E}_2\ ,\ s_2\rangle \longrightarrow \ldots$$

and $E_1 = (l := \hat{E}_1)$, $E_2 = (l := \hat{E}_2)$, ...
By $E \simeq^T_\Gamma E'$ there is an infinite reduction sequence of $\langle E'\ ,\ s\rangle$.
Using Rule (assign2) there is an infinite reduction sequence of $\langle l := E'\ ,\ s\rangle$.

We made the proof simple by staying in a deterministic language with unique derivation trees.

Proof of Congruence Property

Subcase $\neg(\langle l := E\,,\, s\rangle \longrightarrow^{\omega})$

That is

$$\langle l := E\,,\, s\rangle \longrightarrow \langle E_1\,,\, s_1\rangle \longrightarrow \langle E_2\,,\, s_2\rangle \longrightarrow \ldots \longrightarrow \langle E_k\,,\, s_k\rangle \not\longrightarrow$$

All these must be instances of Rule (assign2), except the last step which is an instance of (assign1)

$$\langle E\,,\, s\rangle \longrightarrow \langle \hat{E}_1\,,\, s_1\rangle \longrightarrow \langle \hat{E}_2\,,\, s_2\rangle \longrightarrow \ldots \longrightarrow \langle \hat{E}_{k-1}\,,\, s_{k-1}\rangle$$

and $E_1 = (l := \hat{E}_1)$, $E_2 = (l := \hat{E}_2)$, ..., $E_{k-1} = (l := \hat{E}_{k-1})$ and $\hat{E}_{k-1} = n$, $E_k =$ **skip** and $s_k = s_{k-1} + \{l \mapsto n\}$, for some n.

Proof of Congruence Property

Subcase $\neg(\langle l := E\,,\, s\rangle \longrightarrow^{\omega})$ **(cont'd)**

Hence there is some n and s_{k-1} such that

$$\langle E\,,\, s\rangle \longrightarrow^{*} \langle n\,,\, s_{k-1}\rangle \quad \text{and} \quad \langle l := E\,,\, s\rangle \longrightarrow \langle \textbf{skip}\,,\, s_{k-1} + \{l \mapsto n\}\rangle\ .$$

By $E \simeq_{\Gamma}^{T} E'$ we have $\langle E'\,,\, s\rangle \longrightarrow^{*} \langle n\,,\, s_{k-1}\rangle$.

Using Rules (assign2) and (assign1)

$$\langle l := E'\,,\, s\rangle \longrightarrow^{*} \langle l := n\,,\, s_{k-1}\rangle \rightarrow \langle \textbf{skip}\,,\, s_{k-1} + \{l \mapsto n\}\rangle\ .$$

Congruence Proofs

Congruence proofs are

- tedious
- long
- mostly boring (up to the point where they brake)
- error prone
- recursion is often the killer case

There are dozens of different semantic equivalences
(and each requires a proof)

Back to Examples

- $1 + 1 \simeq_{\Gamma}^{\mathsf{int}} 2$ for any Γ
- $(l := 0\ ;\ 4) \not\simeq_{\Gamma}^{\mathsf{int}} (l := 1\ ;\ 3 +\ !l)$ for any Γ
- $(l :=\ !l + 1)\ ;\ (l :=\ !l + 1) \simeq_{\Gamma}^{\mathsf{unit}} (l :=\ !l + 2)$ for any Γ including l : intref

General Laws

Conjecture

$E_1 \; ; (E_2 \; ; E_3) \simeq^T_\Gamma (E_1 \; ; E_2) \; ; E_3$
for any Γ, T , E_1, E_2 *and* E_3 *such that* $\Gamma \vdash E_1$: *unit,* $\Gamma \vdash E_2$: *unit and* $\Gamma \vdash E_3 : T$.

Conjecture

$((\textbf{if } E_1 \textbf{ then } E_2 \textbf{ else } E_3) \; ; E) \simeq^T_\Gamma (\textbf{if } E_1 \textbf{ then } E_2 \; ; E \textbf{ else } E_3 \; ; E)$
for any Γ, T, E, E_1, E_2 *and* E_3 *such that* $\Gamma \vdash E_1$: *bool,* $\Gamma \vdash E_2$: *unit,* $\Gamma \vdash E_3$: *unit, and* $\Gamma \vdash E : T$.

Conjecture

$(E \; ; (\textbf{if } E_1 \textbf{ then } E_2 \textbf{ else } E_3)) \not\simeq^T_\Gamma (\textbf{if } E_1 \textbf{ then } E \; ; E_2 \textbf{ else } E \; ; E_3)$

General Laws

Suppose $\Gamma \vdash E_1 : \text{unit}$ and $\Gamma \vdash E_2 : \text{unit}$.
When is $E_1 \ ; E_2 \simeq_{\Gamma}^{\text{unit}} E_2 \ ; E_1$?

A Philosophical Question

What is a typed expression $\Gamma \vdash E : T$?

for example $l : \text{intref} \vdash \textbf{if}\ \ !l \geq 0\ \textbf{then skip else}\ (\textbf{skip}\ ; l := 0) : \text{unit}$.

1. a list of tokens (after parsing) `[IF, DEREF, LOC "l", GTEQ, ...]`
2. an abstract syntax tree
3. the function taking store s to the reduction sequence

$$\langle E\ ,\ s \rangle \longrightarrow \langle E_1\ ,\ s_1 \rangle \longrightarrow \langle E_2\ ,\ s_2 \rangle \longrightarrow \ldots$$

4. the equivalence class $\{E' \mid E \simeq^{T}_{\Gamma} E'\}$
5. the partial function $[\![E]\!]_{\Gamma}$ that takes any store s with $\text{dom}(s) = \text{dom}(\Gamma)$ and either is undefined if $\langle E\ ,\ s \rangle \longrightarrow^{\omega}$, or is $\langle v\ ,\ s' \rangle$, if $\langle E\ ,\ s \rangle \longrightarrow^{*} \langle v\ ,\ s' \rangle$