Semantics equivalences

Massimo Merro

4 December 2017

Semantic equivalence

A formal semantics of a programming language allows us to reason about program properties of that language.

Intuition:

Two program phrases P_1 and P_2 are said to be semantically equivalent, $P_1 \simeq P_2$, if either can be replaced by the other, in any program context.

With a good semantic equivalence $\simeq$ we can:

- understand what a program is
- prove whether some particolar expression (say an efficient algorithm) is equivalent to another (say a clear specification); that operation is called program verification!
- prove that some compiler optimizations are sound
- understand semantic differences between programs.

Some examples

How about the following two fragments of code?

$$(\mathrm{l} := 0; 4) \;\; \simeq \;\; (\mathrm{l} := 1; 3 + !\mathrm{l}) \quad ???$$

The two fragment will produce the same results in any starting store.

Can we replace one by the other in any arbitrary program contexts?

No! For example, let

$$C[\cdot] \;\; \stackrel{\text{def}}{=} \;\; [\cdot] + !\mathrm{l}$$

then

$$\begin{array}{ccc} C[\mathrm{l} := 0; 4] & \stackrel{?}{\simeq} & C[\mathrm{l} := 1; 3 + !\mathrm{l}] \\ = & & = \\ (\mathrm{l} := 0; 4) + !\mathrm{l} & \not\simeq & (\mathrm{l} := 1; 3 + !\mathrm{l}) + !\mathrm{l} \end{array}$$

In fact, $C[\mathrm{l} := 0; 4]$ returns 4 while $C[\mathrm{l} := 1; 3 + !\mathrm{l}]$ returns 5. How about

$$(\mathrm{l} := !\mathrm{l} + 1); (\mathrm{l} := !\mathrm{l} - 1) \;\; \simeq \;\; \mathrm{l} := !\mathrm{l} \quad ???$$

Equational reasoning

Both examples were for particolar expressions. We may want to know whether some general laws are valid for all e_1, e_2, ... How about these?

$$
\begin{array}{rcl}
e_1; (e_2; e_3) & \simeq & (e_1; e_2); e_3 \ ? \\
(\text{if } e_1 \text{ then } e_2 \text{ else } e_3); e & \simeq & \text{if } e_1 \text{ then } e_2; e \text{ else } e_3; e \ ? \\
e; (\text{if } e_1 \text{ then } e_2 \text{ else } e_3) & \simeq & \text{if } e_1 \text{ then } e; e_2 \text{ else } e; e_3 \ ? \\
e; (\text{if } e_1 \text{ then } e_2 \text{ else } e_3) & \simeq & \text{if } e; e_1 \text{ then } e_2 \text{ else } e_3
\end{array}
$$

What does it mean for $\simeq$ to be "good"?

1. programs that results in observably-different values (starting from some initial store) must not be equivalent:
   $\exists s, s_1, s_2, v_1, v_2. \langle e_1, s \rangle \twoheadrightarrow^* \langle v_1, s_1 \rangle \wedge \langle e_2, s \rangle \twoheadrightarrow^* \langle v_2, s_2 \rangle \wedge v_1 \neq v_2$
   implies $e_1 \not\simeq e_2$
2. programs that terminates must not be equivalent to programs that don't
3. $\simeq$ must be an equivalence relation:
   $e \simeq e, \qquad e_1 \simeq e_2 \;\Rightarrow\; e_2 \simeq e_1, \qquad e_1 \simeq e_2 \simeq e_3 \;\Rightarrow\; e_1 \simeq e_3$
4. $\simeq$ must be a congruence, i.e. preserved by program contexts:
   if $e_1 \simeq e_2$ then for any context $C[\cdot]$ we must have $C[e_1] \simeq C[e_2]$
5. $\simeq$ should relate as many programs as possible.

Program context

- A program context $C[\cdot]$ is a program which is not completely defined.
- Roughly speaking $C[\cdot]$ denotes a program with a "hole" $[\cdot]$ that needs to be instantiated with some program phrase P
- We write $C[P]$ to denote such a program obtained by instantiating the missing code in $C[\cdot]$ with P.

As an example, in the language *While* program contexts are defined via the following grammar:

$$\begin{array}{rcl} C[\cdot] \in Cxt & ::= & [\cdot] \;\mid\; C[\cdot]\ op\ e_2 \;\mid\; e_1\ op\ C[\cdot] \;\mid\; l := C[\cdot] \\ & \mid & \text{if } C[\cdot] \text{ then } e_2 \text{ else } e_3 \;\mid\; \text{if } e_1 \text{ then } C[\cdot] \text{ else } e_3 \\ & \mid & \text{if } e_1 \text{ then } e_2 \text{ else } C[\cdot] \;\mid\; C[\cdot]; e_2 \;\mid\; e_1; C[\cdot] \\ & \mid & \text{while } e_1 \text{ do } C[\cdot] \;\mid\; \text{while } C[\cdot] \text{ do } e_2 \end{array}$$

For example, if $C[\cdot]$ is the context $\text{while } !\mathrm{l} = 0 \text{ do } [\cdot]$ then $C[\mathrm{l} := !\mathrm{l} + 1]$ is $\text{while } !\mathrm{l} = 0 \text{ do } \mathrm{l} := !\mathrm{l} + 1$.

On congruences

- It is very important that for program equivalence be a congruence!
- Suppose you have a big program *Sys* governing some big system and containing some sub-program P.
- We could write $Sys \stackrel{\text{def}}{=} C[P]$, for some appropriate context $C[\cdot]$.
- And suppose your boss asks you to write down an optimised version P_{fast} of P, with better performances.
- How can you be sure, apart for performances, whether the behaviour of the whole system remains unchanged when replacing the sub-program P with P_{fast}?
- You would have to check whether $C[P] \simeq C[P_{\text{fast}}]$!
- But the two systems $C[P]$ and $C[P_{\text{fast}}]$ may be VERY LARGE!!! This means that their comparison may take months perhaps years!!!
- **Solution**: if the equality $\simeq$ is a congruence then it suffices to prove that the two sub-programs are equivalent: $P \simeq P_{\text{fast}}$. The equality of the whole systems, i.e. $C[P] \simeq C[P_{\text{fast}}]$ follows for free!

A trace-based semantic equivalence for the language *While*

Let us consider our typed language *While* without functions, etc.

Trace equivalence $\simeq^T_\Gamma$

Define $e_1 \ \simeq^T_\Gamma \ e_2$ to hold iff for all stores s such that $\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(s)$, we have $\Gamma \vdash e_1 : T$, $\Gamma \vdash e_2 : T$, and

- $\langle e_1, s \rangle \twoheadrightarrow^* \langle v, s' \rangle$ implies $\langle e_2, s \rangle \twoheadrightarrow^* \langle v, s' \rangle$
- $\langle e_2, s \rangle \twoheadrightarrow^* \langle v, s' \rangle$ implies $\langle e_1, s \rangle \twoheadrightarrow^* \langle v, s' \rangle$.

Congruence property

The equivalence relation $\simeq^T_\Gamma$ enjoys the congruence property because whenever $e_1 \ \simeq^T_\Gamma \ e_2$ we have, for all contexts C and types T', if $\Gamma \vdash C[e_1] : T'$ and $\Gamma \vdash C[e_2] : T'$ then $C[e_1] \ \simeq^{T'}_\Gamma \ C[e_2]$.

On the trace equivalence $\simeq_\Gamma^T$

Let $e_1 \simeq_\Gamma^T e_2$, then:

- If one of the two configurations diverges form some store s then also the other configuration must diverge with the same store.
- Given a store s, if the two configurations converges then it must be on the same value and the same store.

Suppose that given a store s the two configurations $\langle e_1, s\rangle$ and $\langle e_2, s\rangle$ converges, respectively, to $\langle v, s_1\rangle$ and $\langle v, s_2\rangle$, with $s_1(\mathrm{l}) \neq s_2(\mathrm{l})$, for some l, and v of type T. Then a distinguishing context would be the following:

- If $T = \mathsf{unit}$ then $C[\cdot] \stackrel{\text{def}}{=} [\cdot]; !\mathrm{l}$
- If $T = \mathsf{bool}$ then $C[\cdot] \stackrel{\text{def}}{=} \mathsf{if}\ [\cdot]\ \mathsf{then}\ !\mathrm{l}\ \mathsf{else}\ !\mathrm{l}$
- If $T = \mathsf{int}$ then $C[\cdot] \stackrel{\text{def}}{=} \mathrm{l}_1 := [\cdot]; !\mathrm{l}$

Where $\langle C[e_1], s\rangle \rightarrow^* \langle v_1, s_1'\rangle$ and $\langle C[e_2], s\rangle \rightarrow^* \langle v_2, s_2'\rangle$, with $v_1 \neq v_2$.

Back to Examples

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

General laws (1)

Associativity of ;

$$e_1; (e_2; e_3) \simeq^T_\Gamma (e_1; e_2); e_3$$

for any Γ, T, e_1, e_2 and e_3 such that $\Gamma \vdash e_1 : \text{unit}$, $\Gamma \vdash e_2 : \text{unit}$ and $\Gamma \vdash e_3 : T$.

***skip* removal**

- $e_2 \simeq^T_{\Gamma_2} \mathit{skip}; e_2$
- $e_1; \mathit{skip} \simeq^{\text{unit}}_{\Gamma_1} e_1$

for any Γ_1, Γ_2, T, e_1, e_2 such that $\Gamma_2 \vdash e_2 : T$ and $\Gamma_1 \vdash e_1 : \text{unit}$.

if *true*

$$\text{if } \mathit{true} \text{ then } e_1 \text{ else } e_2 \simeq^T_\Gamma e_1$$

for any Γ, T, e_1 and e_2 such that $\Gamma \vdash e_1 : T$ and $\Gamma \vdash e_2 : T$.

General laws (2)

if *false*

$$\text{if } \mathit{false} \text{ then } e_1 \text{ else } e_2 \ \simeq_\Gamma^T \ e_2$$

for any Γ, T, e_1 and e_2 such that $\Gamma \vdash e_1 : T$ and $\Gamma \vdash e_2 : T$.

Distributivity of 'if' wrt ;

$$(\text{if } e_1 \text{ then } e_2 \text{ else } e_3); e \ \simeq_\Gamma^T \ (\text{if } e_1 \text{ then } e_2; e \text{ else } e_3; e)$$

for any Γ, T, e_1, e_2 and e_3 such that $\Gamma \vdash e_1 : \text{bool}$, $\Gamma \vdash e_2 : \text{unit}$, $\Gamma \vdash e_3 : \text{unit}$ and $\Gamma \vdash e : T$.

Distributivity of ; wrt 'if'

$$e; (\text{if } e_1 \text{ then } e_2 \text{ else } e_3) \ \simeq_\Gamma^T \ (\text{if } e; e_1 \text{ then } e_2 \text{ else } e_3)$$

for any Γ, T, e_1, e_2 and e_3 such that $\Gamma \vdash e : \text{unit}$, $\Gamma \vdash e_1 : \text{bool}$, $\Gamma \vdash e_2 : T$, $\Gamma \vdash e_3 : T$.

Wrong laws

$$(e; \text{if } e_1 \text{ then } e_2 \text{ else } e_3) \not\simeq_\Gamma^T (\text{if } e_1 \text{ then } e; e_2 \text{ else } e; e_3)$$

Take:

- e to be $\mathrm{l} := 1$
- e_1 to be $!\mathrm{l} = 0$
- e_2 to be *skip*
- e_3 to be while true do *skip* (loop)

Then, in any store s, where location l is associated to 0, the expression on the left diverges whereas that one on the right converges.

Semantic equivalence: a simulation approach

Simulation

We say that e_1 is simulated by e_2, written $e_1 \sqsubseteq_\Gamma^T e_2$, iff

- $\Gamma \vdash e_1 : T$ and $\Gamma \vdash e_2 : T$, for some T
- for any s with $\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(s)$, if $\langle e_1, s\rangle \rightarrow \langle e_1', s_1'\rangle$ then there is e_2' such that $\langle e_2, s\rangle \rightarrow^* \langle e_2', s_2'\rangle$, with $e_1' \sqsubseteq_\Gamma^T e_2'$ and $s_1' = s_2'$.

Bisimulation

We say that e_1 is bisimilar to e_2, written $e_1 \approx_\Gamma^T e_2$, iff

- $\Gamma \vdash e_1 : T$ and $\Gamma \vdash e_2 : T$, for some T
- for any s with $\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(s)$, if $\langle e_1, s\rangle \rightarrow \langle e_1', s_1'\rangle$ then there is e_2' such that $\langle e_2, s\rangle \rightarrow^* \langle e_2', s_2'\rangle$, with $e_1' \approx_\Gamma^T e_2'$ and $s_1' = s_2'$
- for any s with $\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(s)$, if $\langle e_2, s\rangle \rightarrow \langle e_2', s_2'\rangle$ then there is e_1' such that $\langle e_1, s\rangle \rightarrow^* \langle e_1', s_1'\rangle$, with $e_1' \approx_\Gamma^T e_2'$ and $s_1' = s_2'$.