Semantics equivalences

Massimo Merro

13 December 2011
A formal semantics of a programming language allows us to reason about program properties of that language.

**Intuition:**
Two program phrases are semantically equivalent if either can be replaced by the other, in any program context.

**Application:**
- Program transformation
- Program optimisation
- Program verification.
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:

$$
C[\cdot] \in Cxt \quad ::= \quad [\cdot] \quad | \quad \text{if } B \text{ then } C[\cdot] \text{ else } C_1 \\
| \quad \text{if } B \text{ then } C_1 \text{ else } C[\cdot] \\
| \quad C[\cdot]; C_1 \quad | \quad C_1; C[\cdot] \quad | \quad \text{while } B \text{ do } C[\cdot]
$$

For example, if $C[\cdot]$ is *while* $x = 0$ do $[\cdot]$ then $C[x := x + 1]$ is *while* $x = 0$ do $x := x + 1$.
Basic properties of a semantic equality $\approx_{sem}$

Reflexivity:\[ (\text{Ref}) \quad P \approx_{sem} P \]

Symmetry:\[ (\text{Sym}) \quad Q \approx_{sem} P \quad \frac{P \approx_{sem} P}{P \approx_{sem} Q} \]

Transitivity:\[ (\text{Trans}) \quad P \approx_{sem} Q \quad Q \approx_{sem} R \quad \frac{P \approx_{sem} Q \quad Q \approx_{sem} R}{P \approx_{sem} R} \]

Contextuality:\[ (\text{Cxt}) \quad P \approx_{sem} Q \quad \frac{P \approx_{sem} Q}{C[P] \approx_{sem} C[Q]} \]

Reflexivity, Symmetry and Transitivity says that $\approx_{sem}$ is an equivalence relation. If also contextuality holds then $\approx_{sem}$ is said to be a congruence relation.
On congruences

- It is very important that a program equivalence be a congruence!
- Suppose you have a big program \( \text{Sys} \) governing some big system and containing some sub-program \( P \).
- We could write \( \text{Sys} \overset{\text{def}}{=} C[P] \), for some appropriate context \( C[\cdot] \).
- And suppose your boss asks you to write down an optimised version \( P_{\text{fast}} \) 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 \( P_{\text{fast}} \)?
- You would have to check whether \( C[P] \approx_{\text{sem}} C[P_{\text{fast}}] \)!
- 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 \( \approx_{\text{sem}} \) is a congruence then it suffices to prove that the two sub-programs are equivalent: \( P \approx_{\text{sem}} P_{\text{fast}} \). The equality of the whole systems, i.e. \( C[P] \approx_{\text{sem}} C[P_{\text{fast}}] \) follows for free!
Semantic equivalence for the language While

Commands:

\[ C_1 \approx_{\text{sem}} C_2 \]

- \( \langle C_1, s \rangle \Downarrow s' \) implies \( \langle C_2, s \rangle \Downarrow s' \)
- \( \langle C_2, s \rangle \Downarrow s' \) implies \( \langle C_1, s \rangle \Downarrow s' \)

for all states \( s \) and \( s' \).

Expressions:

\[ E_1 \approx_{\text{sem}} E_2 \]

- \( \langle E_1, s \rangle \Downarrow v \) implies \( \langle E_2, s \rangle \Downarrow v \)
- \( \langle E_2, s \rangle \Downarrow v \) implies \( \langle E_1, s \rangle \Downarrow v \)

for all states \( s \) and value \( v \).
Specific commands:

\[ l := l + 1; k := j + 1 \approx_{\text{sem}} k := j + 1; l := l + 1 \]

Proof reasonably straightforward:

For every possible \( s, s' \) show

- \( \langle l := l + 1; k := j + 1, s \rangle \downarrow s' \) implies \( \langle k := j + 1; l := l + 1, s \rangle \downarrow s' \)
- \( \langle k := j + 1; l := l + 1, s \rangle \downarrow s' \) implies \( \langle l := l + 1; k := j + 1, s \rangle \downarrow s' \)
Examples

General laws:
Which of these laws are true?

\[ C; \text{if } B \text{ then } C_1 \text{ else } C_2 \approx_{\text{sem}} \text{if } B \text{ then } C; C_1 \text{ else } C; C_2 \]

\[ (\text{if } B \text{ then } C_1 \text{ else } C_2); C \approx_{\text{sem}} \text{if } B \text{ then } C_1; C \text{ else } C_2; C \]

Laws must be proved for all possible instantiations of \( C, C_1, C_2 \) and \( B \).
Counterexamples

\[ C; \text{if } B \text{ then } C_1 \text{ else } C_2 \not\approx_{\text{sem}} \text{ if } B \text{ then } C; C_1 \text{ else } C; C_2 \]

Take:
- \( C \) to be \( x := 1 \)
- \( C_1 \) to be \( \text{skip} \)
- \( C_2 \) to be while true do \( \text{skip} \) (loop)
- \( B \) to be \( x = 0 \)

Then
- \( \langle \text{if } x = 0 \text{ then } x := 1; \text{skip} \text{ else } x := 1; C_2, s_0 \rangle \downarrow s_1 \)
- \( \langle x := 1; \text{if } x = 0 \text{ then } \text{skip} \text{ else } C_2, s_0 \rangle \not\triangleright s_1 \)

assuming the notation that \( s_k \) has \( k \) stored in \( x \).
Counterexamples

For specific commands:

To show that $C_1 \not\approx_{sem} C_2$

find states $s$ and $s'$ such that

- either $\langle C_1, s \rangle \downarrow s'$ and $\langle C_2, s \rangle \not\downarrow s'$
- or $\langle C_1, s \rangle \not\downarrow s'$ and $\langle C_2, s \rangle \downarrow s'$
Proving general laws

Example

(if \( B \) then \( C_1 \) else \( C_2 \); \( C \) \( \approx \text{sem} \) if \( B \) then \( C_1 \); \( C \) else \( C_1 \); \( C \)

Must be proved for every possible \( B, C_1, C_2, C \).

Proof outline

- Abbreviate (if \( B \) then \( C_1 \) else \( C_2 \); \( C \) to \( P \)
- Abbreviate if \( B \) then \( C_1 \); \( C \) else \( C_1 \); \( C \) to \( Q \)
- Prove \( \langle P, s \rangle \downarrow s' \) implies \( \langle Q, s \rangle \downarrow s' \) for all possible states
- Conversely, prove \( \langle Q, s \rangle \downarrow s' \) implies \( \langle P, s \rangle \downarrow s' \) for all possible states.
Is $\approx_{\text{sem}}$ a good semantic equivalence?

Is $\approx_{\text{sem}}$ a congruence relation for $\text{While}$?

- Reflexivity, symmetry and transitivity are easy to prove
- Contextuality is quite difficult to establish!

Theorem (Strong Contextuality of $\approx_{\text{sem}}$)

$P \approx_{\text{sem}} Q$ implies $C[P] \approx_{\text{sem}} C[Q]$ for every $\text{While}$ context $C[\cdot]$.

Proof By structural induction on $C[\cdot]$.

However, when proving contextuality we can focus on a simpler family of program contexts called elementary contexts.
Elementary contexts

Intuitively, an elementary context \( \mathcal{E}[:.] \) contains the hole at depth at most 1.

\[
\mathcal{E}[:.] \in \text{EleCxt} ::= [:.] \mid \text{if } B \text{ then } [:.] \text{ else } C_1 \\
\mid \text{if } B \text{ then } C_1 \text{ else } [:.] \\
\mid [:.]; C_1 \mid C_1; [:.] \mid \text{while } B \text{ do } [:.]
\]

Obviously, \( \text{EleCxt} \subseteq \text{Cxt} \), i.e. an elementary context is always a context, while the converse is not always true.

So, we can redefine a notion of contextuality with respect to the notion of elementary context.

**Theorem (Weak Contextuality of \( \approx_{\text{sem}} \))**

\( P \approx_{\text{sem}} Q \) implies \( \mathcal{E}[P] \approx_{\text{sem}} \mathcal{E}[Q] \) for every While elementary context \( \mathcal{E}[:.] \).
Proving weak contextuality

In order to prove weak contextuality we have to show the following sub-statements.

If $P \approx_{\text{sem}} Q$ then for every $C$ and $B$ it must hold the following:

- $P; C \approx_{\text{sem}} Q; C$
- $C; P \approx_{\text{sem}} C; Q$
- if $B$ then $P$ else $C \approx_{\text{sem}}$ if $B$ then $Q$ else $C$
- if $B$ then $C$ else $P \approx_{\text{sem}}$ if $B$ then $C$ else $Q$
- while $B$ do $P \approx_{\text{sem}}$ while $B$ do $Q$

Last property is the difficult one.

The proof of these statements does not require induction. As a consequence, proving weak contextuality is somehow simpler than proving strong contextuality.
Is strong contextuality really stronger than the weak one?

Obviously, strong contextuality implies weak contextuality!

What about the converse?

Working around structural induction on contexts one can prove that weak contextuality implies strong contextuality.

As a consequence, the two forms of contextuality are equivalent and when proving that a semantic equivalence is a congruence is enough to prove the statements indicated in the previous slide!
\( \approx_{\text{sem}} \) is a congruence relation for \textit{While}

**Consequence:**

General laws for \( \approx_{\text{sem}} \) can be applied \textit{anywhere} in a \textit{While} program

**Some laws:**

\[
\begin{align*}
C; \text{skip} & \approx_{\text{sem}} C \\
\text{skip}; C & \approx_{\text{sem}} C \\
\text{if true then } C_1 \text{ else } C_2 & \approx_{\text{sem}} C_1 \\
\text{if false then } C_1 \text{ else } C_2 & \approx_{\text{sem}} C_2 \\
l := n; k := m & \approx_{\text{sem}} k := m; l := n
\end{align*}
\]