CS720

Logical Foundations of Computer Science

Lecture 13: Program equivalence

Tiago Cogumbreiro
Summary

- Behavioral equivalence
- Properties on behavioral equivalence
- Program transformations
Program equivalence

- A framework to compare "equivalent" programs, notation $P \equiv Q$
- The notion of equivalent is generic
- Program equivalence can be used to reason about correctness of algorithms
- Program equivalence can be used to reason about the correctness of program transformations

Examples:

- compilable programs
- programs that produce the same output
- programs that perform the same assignments
- programs that read the same variables
Usual equivalence properties

- Reflexive: $P \equiv P$
- Symmetric: $P \equiv Q \implies Q \equiv P$
- Transitive: $P \equiv Q \implies Q \equiv R \implies P \equiv R$
- Congruence: $P \equiv Q \implies C(P) \equiv C(Q)$ where $C : \mathcal{P} \to \mathcal{P}$ is known as a context, a program with a "whole" that is filled with the input program, outputting a "complete" program; it is expected that the input occurs in the output.
Syntactic equivalence

If two programs are textually equal (are the same syntactic term), then we say that the two programs are syntactically equivalent.

Example: `APlus (ANum 3) (ANum 0)` is syntactically equivalent to `APlus (ANum 3) (ANum 0)`.

Behavioral equivalence

If two programs start from an initial state and reach the same final state, then we say that the two programs are behaviorally equivalent.

Example:

```
X:=3;; WHILE 1≤X DO Y:=Y+1;; X:=X-1 END
```

is behaviorally equivalent to

```
X:=0 ;; Y:=3
```
How do we formalize behavioral equivalence for arithmetic expressions, boolean expressions, commands?
Behavioral equivalence

For arithmetic expressions $a_1 \equiv a_2$, e.g., $x - x \equiv 0$:
Behavioral equivalence

For arithmetic expressions $a_1 \equiv a_2$, e.g., $x - x \equiv 0$:

$$\forall s: \text{aeval}(s, a_1) = \text{aeval}(s, a_2)$$

$$a_1 \equiv a_2$$

For boolean expressions $b_1 \equiv b_2$, e.g., $(x - x = 0) \equiv \top$:
Behavioral equivalence

For arithmetic expressions $a_1 \equiv a_2$, e.g., $x - x \equiv 0$: 

$$\forall s:\ aeval(s, a_1) = aeval(s, a_2)$$

$$a_1 \equiv a_2$$

For boolean expressions $b_1 \equiv b_2$, e.g., $(x - x = 0) \equiv \top$: 

$$\forall s:\ beval(s, b_1) = beval(s, b_2)$$

$$b_1 \equiv b_2$$

For commands $c_1 \equiv c_2$: 

Behavioral equivalence

For arithmetic expressions $a_1 \equiv a_2$, e.g., $x - x \equiv 0$:

$$\forall s : \text{aeval}(s, a_1) = \text{aeval}(s, a_2)$$

$$a_1 \equiv a_2$$

For boolean expressions $b_1 \equiv b_2$, e.g., $(x - x = 0) \equiv \top$:

$$\forall s : \text{beval}(s, b_1) = \text{beval}(s, b_2)$$

$$b_1 \equiv b_2$$

For commands $c_1 \equiv c_2$:

$$\forall s_1, \forall s_2 : s_1 = [c_1] \Rightarrow s_2 \iff s_1 = [c_2] \Rightarrow s_2$$

$$c_1 \equiv c_2$$
Exercise: skip

Prove that

\[ \text{skip;} \ c \equiv c \]

**Theorem skip_left**: for all \( c \),

\[ cequiv \langle \text{skip;} \ c \rangle c. \]
Exercise: if

If $b \equiv \top$, then if $b$ then $c_1$ else $c_2$ end $\equiv c_1$.

Theorem if_true: forall b c1 c2, bequiv b true cequiv c1.

What could $b$ in $b \equiv \top$ be? For instance, the following statement holds. (By using lemmas Nat.add_0_r, Nat.eqb_refl.)

$$(x + x = 2 \times x) \equiv \top$$

Require Import PeanoNat.
Goal forall x, bequiv (x + x = 2 * x) BTrue.
Exercise: while

**Theorem:** If \( b \equiv \bot \), then \( \text{while } b \text{ do } c \text{ end } \equiv \text{skip} \).

**Theorem:** If \( b \equiv \top \), then for all \( s \) and \( s' \), we have \( \neg s \equiv \text{[while } b \text{ do } c \text{ end]} \Rightarrow s' \).

**Theorem:** \( \text{while } b \text{ do } c \text{ end } \equiv \text{if } b \text{ then } c; \text{ while } b \text{ do } c \text{ end else skip end} \).
Properties of equivalences

An equivalence relation is:

- reflexive
- symmetric
- transitive

Show that \( \equiv_v \), \( \equiv_b \), and \( \equiv_c \) each is an equivalence relation.

**Lemma** refl_cequiv : \( \forall (c : \text{com}), \equiv_c c c \).

**Lemma** sym_cequiv : \( \forall (c1 c2 : \text{com}), \equiv_c c1 c2 \Rightarrow \equiv_c c2 c1 \).

**Lemma** trans_cequiv : \( \forall (c1 c2 c3 : \text{com}), \equiv_c c1 c2 \Rightarrow \equiv_c c2 c3 \Rightarrow \equiv_c c1 c3 \).
is a congruence

Generally a congruence can be described as

\[ c \equiv c' \implies C(c) \equiv C(c') \]

For commands this corresponds to proving

\[
\begin{align*}
  a &\equiv a' \\
  (x := a) &\equiv (x := a') \\
  c_1 &\equiv c_1' \\
  c_2 &\equiv c_2'
\end{align*}
\]

\[
\begin{align*}
  b &\equiv b' \\
  c_1 &\equiv c_1' \\
  c_2 &\equiv c_2'
\end{align*}
\]

\[
\text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end } \equiv \text{if } b' \text{ then } c_1' \text{ else } c_2' \text{ end}
\]

\[
\begin{align*}
  b &\equiv b' \\
  c &\equiv c'
\end{align*}
\]

\[
\text{while } b \text{ do } c \text{ end } \equiv \text{while } b' \text{ do } c' \text{ end}
\]
Example congruence_example:

```
cequiv
  (* Program 1: *)
  <{ X := 0;
    if (X = 0)
      then Y := 0
    else Y := 42 end }>

  (* Program 2: *)
  <{ X := 0;
    if (X = 0)
      then Y := X - X
    else Y := 42 end }>
```
Sound transformations

- We can specify the notion of a transformation that is **sound**
- Example: source-to-source compiler, code optimizer.

**Definition** atrans_sound (atrans : aexp → aexp) : Prop :=
  \[\forall (a : aexp),\]
  \[aequiv a (atrans a).\]

**Definition** btrans_sound (btrans : bexp → bexp) : Prop :=
  \[\forall (b : bexp),\]
  \[bequiv b (btrans b).\]

**Definition** ctrans_sound (ctrans : com → com) : Prop :=
  \[\forall (c : com),\]
  \[cequiv c (ctrans c).\]