

CS256: Programming Languages and Semantics

Weakest Pre-Conditions

<http://www.eecs.harvard.edu/~greg/cs256sp2006/>

Greg Morrisett
greg@eecs.harvard.edu
327 Maxwell Dworkin

February 17, 2006

1 Overview

In the homework for last time, I asked that you show the soundness of the assertion logic. That is:

$$\vdash \{A_1\}c\{A_2\} \supset \models \{A_1\}c\{A_2\}$$

In English, if we can use the Hoare proof rules to show that a triple holds, then it is true according to the semantic definition of the relation.

Our goal today is to sketch the *relative* completeness of the assertion logic. In particular, given an oracle that can witness the validity of an assertion $\models A$, anything that is true is “provable” relative to this oracle:

$$\models \{A_1\}c\{A_2\} \supset \vdash \{A_1\}c\{A_2\}$$

The proof depends upon the construction of weakest liberal pre-conditions as assertions. Recall that the weakest liberal pre-condition of a command c with respect to a post-condition A_2 is defined semantically as the set of stores:

$$\{s \mid \langle c, s \rangle \Downarrow s' \supset s' \models A_2\}$$

Our proof of completeness will hinge upon the fact that we can represent this set precisely using an assertion.

In particular, the proof of relative completeness goes something like this:

1. We claim that there exists an assertion A such that A is the weakest liberal pre-condition of C with respect to A_2 . That is, a store s is in the set defined above iff $s \models A$.
2. We claim that $\vdash \{A\}c\{A_2\}$ is derivable.
3. We claim that $\models A_1 \supset A$.
4. Therefore, using the rule of consequence with 2 and 3, we can derive $\vdash \{A_1\}c\{A_2\}$.

So now all that remains is to show that we can represent weakest liberal pre-conditions as assertions, that they are derivable using the Hoare proof rules, and that they are indeed the weakest.

The following inductive definition yields an assertion for every command and post-condition (except for while-loops):

$$\text{wlp}[\text{skip}]A = A$$

$$\text{wlp}[x := e]A = A[e/x]$$

$$\text{wlp}[c_1; c_2]A = \text{wlp}[c_1](\text{wlp}[c_2]A)$$

$$\text{wlp}[\text{if } e \text{ then } c_1 \text{ else } c_2]A = (e \neq 0 \wedge \text{wlp}[c_1]A) \vee (e = 0 \wedge \text{wlp}[c_2]A)$$

You'll have to trust me (or read Winskell) to know that wlp for while-loops is also computable. The essence of this argument is that we can encode the behavior of the while loop as a (partial) function from integers to integers, and such functions can be represented as an assertion of the form $\forall i.\exists j.\dots$. The reason that integers suffice is that there's only a finite number of (integer-valued) variables that the command can read or write, so we can reduce the command to a function from tuples of integers to tuples of integers. Those tuples can be encoded using Goedel representations and so forth. It's all very messy and I can't reproduce the argument without looking carefully at some other source. The key thing is that calculating the wlp for while-loops is in principle possible.

For the cases above, it's pretty easy to see that

$$\vdash \{\text{wlp}[c]A\}c\{A\}$$

You can prove this straightforwardly by induction on c except that, of course, the case for while loops is hideously complicated due to the encoding.

From our soundness result, it follows that:

$$\models \{\text{wlp}[c]A\}c\{A\}$$

so we at least know that wlp gives some pre-condition which ensures that if we run c , we end up in a state that satisfies A .

What remains is to show that wlp is indeed the weakest such pre-condition. This follows from the following theorem:

$$\forall c.\forall A, s_1, s_2.(\langle c, s_1 \rangle \Downarrow s_2 \wedge s_2 \models A) \supset s_1 \models \text{wlp}[c]A.$$

and the proof of this proceeds by induction on c . I'll do the two easy cases and let you think about the other two:

- Suppose $c = \text{skip}$. Pick A , s_1 and s_2 such that $\langle \text{skip}, s_1 \rangle \Downarrow s_2$ and $s_2 \models A$. From the big-step evaluation rule for **skip**, we know that $s_1 = s_2$. From the definition of wlp, we know that $\text{wlp}[\text{skip}]A = A$. Thus, $s_1 \models \text{wlp}[\text{skip}]A$.
- Suppose $c = c_1; c_2$. Pick A , s_1 , and s_2 such that $\langle c_1; c_2, s_1 \rangle \Downarrow s_2$ and $s_2 \models A$. From the big-step evaluation rule for composition, we know that there exists an s such that $\langle c_1, s_1 \rangle \Downarrow s$ and $\langle c_2, s \rangle \Downarrow s_2$. Using the induction hypothesis on c_2 , we can conclude that $s \models \text{wlp}[c_2]A$. Then using the induction hypothesis on c_1 , we can conclude that $s_1 \models \text{wlp}[c_1](\text{wlp}[c_2]A)$. Therefore, $s_1 \models \text{wlp}[c_1; c_2]A$.

The cases for if-commands and assignments are relatively straightforward, though the latter demands reasoning about the substitution of expressions in assertions. In particular, we need a lemma that tells us whenever $\langle e, s \rangle \Downarrow i$ and $s \models A$, then $s[x \mapsto i] \models A[e/x]$.