Axiomatic Semantics

Computer Science and Artificial Intelligence Laboratory
MIT
Nadia Polikarpova with slides by Armando Solar-Lezama

October 26, 2015
Example

\[ \{ x = x_0 \text{ and } y = y_0 \} \]

if \( x > y \) {
  
  \[ t = x - y; \]
  
  while \( t > 0 \) {
    
    \[ x = x - 1; \]
    
    \[ y = y + 1; \]
    
    \[ t = t - 1; \]
    
  }

}

\[ \{ x_0 > y_0 \Rightarrow y = x_0 \text{ and } x = y_0 \} \]
Example

```c
if (x > y) {
    { x=y0 and y=y0 }
    t = x - y;
    { x=y0+t and y=x0-t and t>=0 }
    while (t > 0) {
        { x=y0+t and y=x0-t and t>=0 and t>0 }
        { x-1=y0+t-1 and y+1=x0-(t-1) and t-1>=0 }
        x = x - 1;
        { x=y0+t-1 and y+1=x0-(t-1) and t-1>=0 }
        y = y + 1;
        { x=y0+t-1 and y=x0-(t-1) and t-1>=0 }
        t = t - 1;
        { x=y0+t-1 and y=x0-(t-1) and t-1>=0 }
    }
    { x=y0+t and y=x0-t and t>=0 and !(t>0) }
    { y=x0 and x=y0 }
}
{ x0>y0 => y=x0 and x=y0 }
```

\[\vdash \{ A[x \to e] \} x := e \{ A \}\]
\[\vdash \{ A \} c_1 \{ C \} \vdash \{ C \} c_2 \{ B \}\]
\[\vdash \{ A \} c_1 ; c_2 \{ B \}\]
\[\vdash \{ A \} if b then c_1 else c_2 \{ B \}\]
\[\vdash \{ A \} \wedge b \{ B \} \vdash \{ A \} \wedge \neg b \{ B \}\]
\[\vdash \{ A \} \wedge b \{ A \}\]
\[\vdash \{ A \} \wedge \neg b \{ A \}\]
\[\vdash \{ A \} while b do c \{ A \wedge \neg b \}\]
\[\vdash A' \Rightarrow A \vdash \{ A \} c \{ B \} \vdash B \Rightarrow B'\]
\[\vdash \{ A' \} c \{ B' \}\]
From partial to total correctness

Total correctness judgment

- \( \vdash [A] c [B] \)
- Just like before, but must also prove termination

\[
\begin{align*}
\vdash [A \land b] c_1 [B] & \quad \vdash [A \land \text{not } b] c_2 [B] \\
\vdash [A] \text{if } b \text{ then } c_1 \text{ else } c_2 [B] & \quad \vdash [A[x \rightarrow e]] x := e [A] \\
\vdash [A] c_1 [C] & \quad \vdash [C] c_2 [B] \\
\vdash [A] c_1; c_2 [B]
\end{align*}
\]

What about loops
Rank function

Function F of the state that
- a) Maps state to an integer
- b) Decreases with every iteration of the loop
- c) Is guaranteed to stay greater than zero
- Also called variant function

\[
\vdash [A \land b \land F = z] c [A \land F < z] \quad \vdash A \land b \Rightarrow F \geq 0
\]

\[
\vdash [A] \text{while } b \text{ do } c \ [A \land \text{not } b]
\]
Example

Can we prove this?

\[ x=x_0 \text{ and } y=y_0 \]
\[
\text{if}(x > y)\
\hspace{1em}t = x - y; \\
\hspace{1em}\text{while}(t > 0)\
\hspace{2em}x = x - 1; \\
\hspace{2em}y = y + 1; \\
\hspace{2em}t = t - 1; \\
\hspace{1em}\}
\]

\[ x_0 > y_0 \Rightarrow y=x_0 \text{ and } x=y_0 \]
Example

\{ x=x_0 \text{ and } y=y_0 \} 

if (x > y) { 
\{ x>y \text{ and } x=x_0 \text{ and } y=y_0 \} 
\{ x=y_0+x-y \text{ and } y=x_0-(x-y) \text{ and } x-y>=0 \} 
t = x - y;
\{ x=y_0+t \text{ and } y=x_0-t \text{ and } t>=0 \} 
while (t > 0) { 
\{ x=y_0+t \text{ and } y=x_0-t \text{ and } t>=0 \text{ and } t>0 \text{ and } t=z \} 
\{ x-1=y_0+t-1 \text{ and } y+1=x_0-(t-1) \text{ and } t-1>=0 \text{ and } t-1<z \} 
x = x - 1;
\{ x=y_0+t-1 \text{ and } y+1=x_0-(t-1) \text{ and } t-1>=0 \text{ and } t-1<z \} 
y = y + 1;
\{ x=y_0+t-1 \text{ and } y=x_0-(t-1) \text{ and } t-1>=0 \text{ and } t-1<z \} 
t = t - 1;
\{ x=y_0+t \text{ and } y=x_0-t \text{ and } t>=0 \text{ and } t<z \} 
}\{ x=y_0+t \text{ and } y=x_0-t \text{ and } t>=0 \text{ and } !(t>0) \} 
\{ y=x_0 \text{ and } x=y_0 \} 
}\{ x_0>y_0 \Rightarrow y=x_0 \text{ and } x=y_0 \} 

\vdash [ A \land b \land F = z ]c \ [ A \land F < z ] \vdash A \land b \Rightarrow F \geq 0 
\vdash [ A ]\text{while } b \text{ do } c \ [ A \land \text{not } b ]
Weakest Preconditions

Weakest predicate $P$ such that $\models \{P\} c \{A\}$

- $P$ weaker than $Q$ iff $Q \Rightarrow P$

$wpc(skip \{Q\}) = Q$

$wpc(x = e\{Q\}) = Q[e/x]$)

$wpc(C1; C2\{Q\}) = wpc(C1\{wpc(C2\{Q\})\})$

$wpc(if \ B \ then \ C1 \ else \ C2\{Q\}) = (B \ and \ wpc(C1\{Q\})) \ or \ (not \ B \ and \ wpc(C2\{Q\}))$
Weakest Precondition

While-loop is tricky

- Let $W = wpc(\text{while } e \text{ do } c, B)$
- then,

$$W = e \Rightarrow wpc(c, W) \land \neg e \Rightarrow B$$
Verification Condition

Stronger than the weakest precondition

Can be computed by using an invariant

\[ VC(\text{while}_I \ e \ do \ c, B) = \]
\[ I \land \forall x_1, \ldots x_n \ I \implies (e \implies VC(c, I) \land \neg e \implies B) \]
- Where x_i are variables modified in c.
Example

Is this program correct?

\[
i = 5;\\
\text{while } (i > 0) \quad \text{invariant } \{ i \geq 0 \}\\
\{ \quad i = i - 1; \\
\} \quad \{ i == 0 \}
\]

\[VC(\text{while}, e \text{ do } c, B) = I \land \forall x_1, \ldots, x_n I \Rightarrow (e \Rightarrow VC(c, I) \land \neg e \Rightarrow B)\]

\[vc(i := 5; \text{while}(i > 0)i := i - 1, i = 0)\]
\[vc(i := 5, vc(\text{while}(i > 0)i := i - 1, i = 0))\]
\[vc(i := 5, i \geq 0 \land \forall i. i \geq 0 \Rightarrow (i > 0 \Rightarrow i - 1 \geq 0) \land (\neg (i > 0) \Rightarrow i = 0))\]
\[5 \geq 0 \land \forall i. i \geq 0 \Rightarrow (i > 0 \Rightarrow i - 1 \geq 0) \land (\neg (i > 0) \Rightarrow i = 0)\]
Assert and Assume

It is convenient to extend the language with statements that prescribe which executions are correct / feasible:

assert e: e must hold in every correct execution
assume e: e must hold in every feasible execution

```plaintext
{ x=x0 and y=y0 }
z = x;
x = y;
y = z;
{ y=x0 and x=y0 }
```

```plaintext
assume x == x0;
assume y == y0;
z = x;
x = y;
y = z;
assert x == x0;
assert y == y0;
```
Weakest Precondition

\[ wpc(\text{assert } e, Q) = ?? \]
for Q to be true after, e must also be true before, because otherwise we won’t get past the assert

\[ wpc(\text{assume } e, Q) = ?? \]
if e is not true, we don’t care if Q is satisfied
Example

Is this program correct?

```plaintext
y = 5;
if (x > 0) {
    assert x + y > 5;
} else {
    assume x == 0;
    y = y + x;
    assert x + y == 5;
}
```

```plaintext
wpc(y := 5; if ..., T)
wpc(y := 5, wpc(if ..., T))
wpc(y := 5, (x > 0 ∧ wpc(assert x + y > 5, T)) ∨
    (x ≤ 0 ∧ wpc(assume x = 0; y := y + x; assert x + y = 5, T))))
wpc(y := 5, (x > 0 ∧ x + y > 5) ∨ (x ≤ 0 ∧ (x = 0 ⇒ x + y + x = 5)))
(x > 0 ∧ x + 5 > 5) ∨ (x ≤ 0 ∧ (x = 0 ⇒ x + 5 + x = 5))
```

What now? How do we decide if this formula is valid?
SMT-LIB

SMT-LIB is a language for specifying input to SMT solvers

Basic instructions:

\[(\text{declare-fun } x () \text{ Int})\] declare an integer constant x

\[(\text{assert } (> x 0))\] add \(x > 0\) to known facts

\[(\text{check-sat})\] check if there exist an assignment that makes all known facts true

\[(\text{get-model})\] print this assignment
SMT for verification

We need to decide if \( wpc(\text{prog}, \text{true}) \) is valid
- for all values of program variables on entry

How do we encode this as an SMT problem?
- ask if \( \neg wpc(\text{prog}, \text{true}) \) is satisfiable
- if the answer is UNSAT, the problem is correct
- if the answer is SAT, the model gives the input values that violate correctness
Example

Is this formula valid?

\((x > 0 \land x + 5 > 5) \lor (x \leq 0 \land (x = 0 \Rightarrow x + x + 5 = 5))\)

```
(declare-fun x () Int)

(assert (not (and (> x 0) (> (+ x 5) 5))))
(assert (not
       (and (<= x 0) (or (not (= x 0)) (= (+ x (+ x 5)) 5))))))

(check-sat)
```