ESC Java

Is This Program Correct?

```java
int square(int n) {
    int k = 0, r = 0, s = 1;
    while(k != n) {
        r = r + s; s = s + 2; k = k + 1;
    }
    return r;
}
```

- Type checking not enough to check this
  - Neither is data-flow analysis, nor model checking

Program Verification

- Program verification is the most powerful static analysis method
  - Can reason about all properties of programs
- Cannot fully automate
- But …
  - Can automate certain parts (ESC/Java)
  - Teaches how to reason about programs in a systematic way
Specifying Programs

- Before we check a program we must specify what it does

- We need formal specifications
  - English comments are not enough

- We use logic notation
  - Theory of pre- and post-conditions

State Predicates

- A predicate is a boolean expression on the program state (e.g., variables, object fields)

- Examples:
  - \( x = 8 \)
  - \( x < y \)
  - \( \text{true} \)
  - \( \text{false} \)
  - \( (\forall i. 0 \leq i < a.length \Rightarrow a[i] \geq 0) \)

Using Predicates to Specify Programs

- We focus first on how to specify a statement

- Hoare triple for statement S
  
  \[ \{ P \} S \{ Q \} \]

- Says that if S is started in a state that satisfies P, and S terminates, then it terminates in Q
  
  - This is the liberal version, which doesn’t care about termination
  
  - Strict version: if S is started in a state that satisfies P then S terminates in Q

Hoare Triples. Examples.

- \( \{ \text{true} \} x = 12 \{ x = 12 \} \)

- \( \{ y \geq 0 \} x = 12 \{ x = 12 \} \)

- \( \{ \text{true} \} x = 12 \{ x \geq 0 \} \)

  (Programs satisfy many possible specifications)

- \( \{ x < 10 \} x = x + 1 \{ x < 11 \} \)

- \( \{ n \geq 0 \} x = \text{fact}(n) \{ x = n! \} \)

- \( \{ \text{true} \} a = 0; \text{if}(x != 0) \{ a = 2 * x; \} \{ a = 2 * x \} \)
Computing Hoare Triples

- We compute the triples using rules
  - One rule for each statement kind
  - Rules for composed statements

Assignment

- Assignment is the simplest operation and the trickiest one to reason about!
- \{ y \geq 2 \} x = 5 \{ ? \}
- \{ x \equiv y \} x = x + 1 \{ ? \}
- \{ ? \} x = 5 \{ x \equiv y \}
- \{ ? \} x = x + 1 \{ x \equiv y \}
- \{ ? \} x = x + 1 \{ x^2 + y^2 \equiv z^2 \}
- \{ x^2 + y^2 \equiv z^2 \} x = x + 1 \{ ? \}

Assignment Rule

- Rule for assignment
  \[
  \{ Q[x := E] \} \quad x = E \quad \{ Q \}
  \]

  - \(Q\) with \(x\) replaced by \(E\)

Examples:
- \(\{ 12 \equiv 12 \} \quad x = 12 \quad \{ x \equiv 12 \}
- \{ 12 \geq 0 \} \quad x = 12 \quad \{ x \geq 0 \}
- \{ ? \} \quad x = x + 1 \quad \{ x \geq 0 \}
- \{ x \geq 1 \} \quad x = x + 1 \quad \{ ? \}

Relaxing Specifications

- Consider \(\{ x \geq 1 \} \quad x = x + 1 \quad \{ x \geq 2 \}\)
  - It is very tight specification. We can relax it
  \[
  \{ P \} \quad \text{if} \ P \Rightarrow Q[x:=E] \quad \{ Q \}
  \]

Examples:
- \(\{ x \geq 5 \} \quad x = x + 1 \quad \{ x \geq 2 \}\)
  (since \(x \geq 5 \Rightarrow x + 1 \geq 2\)
Assignments: forward and backward

- Two ways to look at the rules:
  - Backward: given post-condition, what is pre-condition?
    \[
    \begin{array}{l}
    \{ Q \}\ [ x := E ] \rightarrow \{ Q \} \\
    x = E \\
    \{ ?? \}
    \end{array}
    \]
  - Forward: given pre-condition, what is post-condition?
    \[
    \begin{array}{l}
    \{ P \}\ [ x := E ] \rightarrow \{ ?? \} \\
    x = E \\
    \{ ?? \}
    \end{array}
    \]

Example of running it forward

- \{ x == y \} \ x = x + 1 \{ ? \}
Example of running it forward

\[ \{ x = y \} x = x + 1 \{ ? \} \]

\[ \exists \nu. \ (\nu = y \land x = \nu + 1) \]
\[ \iff \ x = y + 1 \]

Forward or Backward

- **Forward reasoning**
  - Know the precondition
  - Want to know what postcondition the code establishes

- **Backward reasoning**
  - Know what we want to code to establish
  - Must find in what precondition this happens

- Backward is used most often
  - Start with what you want to verify
  - Instead of verifying everything the code does

Weakest precondition

- \( wp(S, Q) \) is the weakest \( P \) such that \( \{ P \} S \{ Q \} \)
  - Order on predicates: Strong \( \rightarrow \) Weak
  - \( wp \) returns the “best” possible predicate
- \( wp(x := E, Q) = Q[x := E] \)
- In general:

\[
\begin{array}{c}
\{ P \} \\
S \\
\{ Q \}
\end{array}
\]

\[
\text{if } P \Rightarrow wp(S,Q)
\]

Weakest precondition

- This points to a verification algorithm:
  - Given function body annotated with pre-condition \( P \) and post-condition \( Q \):
    - Compute \( wp \) of \( Q \) with respect to function body
    - Ask a theorem prover to show that \( P \) implies the \( wp \)
- The \( wp \) function we will use is liberal (\( P \) does not guarantee termination)
  - If using both strict and liberal in the same context, the usual notation is \( wp \) the liberal version and \( wp \) for the strict one
**Strongest precondition**

- \( sp(S, P) \) is the strongest \( Q \) such that \( \{ P \} S \{ Q \} \)
  - Recall: Strong \( \Rightarrow \) Weak
  - \( sp \) returns the "best" possible predicate
- \( sp(x := E, P) = \ldots \)
- In general:
  \[
  \begin{array}{c}
  \{ P \} \\
  S \\
  \{ Q \} \quad \text{if} \ sp(S,P) \Rightarrow Q
  \end{array}
  \]

**Strongest postcondition**

- Strongest postcondition and weakest preconditions are symmetric
- This points to an equivalent verification algorithm:
  - Given function body annotated with pre-condition \( P \)
    and post-condition \( Q \):
    - Compute \( sp \) of \( P \) with respect to function body
    - Ask a theorem prover to show that the \( sp \) implies \( Q \)

**Composing Specifications**

- If \( \{ P \} S_1 \{ R \} \) and \( \{ R \} S_2 \{ Q \} \) then \( \{ P \} S_1; S_2 \{ Q \} \)
- Example:
  \[
  \begin{array}{l}
  x = x - 1; \\
  y = y - 1
  \end{array}
  \quad \{ x \geq y \} \]

**Composing Specifications**

- If \( \{ P \} S_1 \{ R \} \) and \( \{ R \} S_2 \{ Q \} \) then \( \{ P \} S_1; S_2 \{ Q \} \)
- Example:
  \[
  \begin{array}{l}
  x = x - 1; \\
  x \geq y - 1 \Leftrightarrow \{ x \geq y \}
  \end{array}
  \quad \begin{array}{l}
  y = y - 1 \\
  x \geq y - 1 \Leftrightarrow \{ x \geq y \}
  \end{array}
  \]
In terms of \(wp\) and \(sp\)

- \(wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q))\)
- \(sp(S_1; S_2, P) = sp(S_2, sp(S_1, P))\)

\[7\]

### Conditionals

- Rule for the conditional (flow graph)

\[\begin{align*}
T & \quad \{ P \} \\
E & \quad F
\end{align*}\]

- If \(P \land E \Rightarrow P_1\)
- If \(P \land \neg E \Rightarrow P_2\)

- Example:

\[\begin{align*}
T & \quad \{ x \geq 0 \} \\
E & \quad F
\end{align*}\]

- \(P_1\): \(x = 0\)
- \(P_2\): \(x = 1\)

\(x \geq 0 \land x = 0 \Rightarrow x = 0\) since \(x \geq 0 \land x \neq 0 \Rightarrow x \geq 1\)

### Conditionals: Forward and Backward

- Recall: rule for the conditional

\[\begin{align*}
T & \quad \{ P \} \\
E & \quad F
\end{align*}\]

- Provided \(P \land E \Rightarrow P_1\)
- Provided \(P \land \neg E \Rightarrow P_2\)

- Forward: given \(P\), find \(P_1\) and \(P_2\)
  - Pick \(P_1\) to be \(P \land E\), and \(P_2\) to be \(P \land \neg E\)

- Backward: given \(P_1\) and \(P_2\), find \(P\)
  - Pick \(P\) to be \((P_1 \land E) \lor (P_2 \land \neg E)\)
  - Or pick \(P\) to be \((E \Rightarrow P_1) \land (\neg E \Rightarrow P_2)\)

### Joins

- Rule for the join:

\[\begin{align*}
T & \quad \{ P_1 \} \\
E & \quad F
\end{align*}\]

- Provided \(P_1 \Rightarrow P\) and \(P_2 \Rightarrow P\)

- Forward: pick \(P\) to be \(P_1 \parallel P_2\)

- Backward: pick \(P_1, P_2\) to be \(P\)
Review

<table>
<thead>
<tr>
<th>Condition</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P )</td>
<td>( P^1 ) and ( P^2 ) imply ( P )</td>
</tr>
</tbody>
</table>

Review: forward

<table>
<thead>
<tr>
<th>Condition</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P )</td>
<td>( P^1 ) and ( P^2 ) imply ( P )</td>
</tr>
</tbody>
</table>

Implication is always in the direction of the control flow

Review: backward

<table>
<thead>
<tr>
<th>Condition</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Q(x := E) )</td>
<td>( P ) and ( P^1 ) imply ( Q )</td>
</tr>
</tbody>
</table>

Example: Absolute value

```
static int abs(int x)
//@ ensures \result >= 0
{
    if (x < 0) {
        x = -x;
    }
    if (c > 0) {
        c--; //...
    }
    return x;
}
```
Example: Absolute value

In Simplify

```lisp
> (and (implies (= a 0) (implies (or (implies (+ a 0) (= (+ a 0) 0))
                              (implies (= a 0) (implies (or (implies (+ a 0) (= (+ a 0) 0)))))
                              (implies (= a 0) (implies (or (implies (+ a 0) (= (+ a 0) 0)))))
                              (= a 0))))
1: Valid.
> 
```

Example: Absolute value

So far...

- Framework for checking pre and post conditions of computations without loops
- Suppose we want to check that some condition holds inside the computation, rather than at the end

```java
static int abs(int x) {
    if (x < 0) {
        x = -x;
        if (x > 0) {
            c--;  
        }
    } else {
        return x;
    }
}
```
Asserts

- \{ Q \land E \} \ assert(E) \{ Q \}
- Backward: wp(\assert(E), Q) = Q \land E
  \begin{align*}
  & assert(E) \\
  & \downarrow \\
  & Q
  \end{align*}
- Forward: sp(\assert(E), P) = ???
  \begin{align*}
  & assert(E) \\
  & \downarrow \\
  & ????
  \end{align*}

Example: Absolute value with assert

```
static int abs(int x)
{
    if (x < 0) {
        x = -x;
        assert(x > 0);
    }
    if (c > 0) {
        c--;
    }
    return x;
}
```
Adding the postcondition back in

```
x < 0
x = -x
assert(x > 0)
```

```
c > 0
c--
```

Another Example: Double Locking

"An attempt to re-acquire an acquired lock or release a released lock will cause a deadlock."

Calls to `lock` and `unlock` must alternate.

Locking Rules

- We assume that the boolean predicate `locked` says if the lock is held or not

- `{ ! locked && P[locked := true] } lock { P }`
  - `lock` behaves as `assert(! locked); locked = true`

- `{ locked && P[locked := false] } unlock { P }`
  - `unlock` behaves as `assert(locked); locked = false`
Locking Example

\begin{align*}
\{ !L \land P[L := true] \} & \text{lock } \{ P \} \\
\{ L \land P[L := false] \} & \text{unlock } \{ P \}
\end{align*}


Locking Example: forward direction

\begin{align*}
\{ !L \land x = 0 \} & \text{lock } \{ P \} \\
\{ !L \land P[L := true] \land x = 0 \} & \text{unlock } \{ P \}
\end{align*}