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
  - true
  - false
  - (\forall i. 0 <= i < a.length \Rightarrow a[i] >= 0)
## Using Predicates to Specify Programs

- We focus first on how to specify a statement
- **Hoare triple for statement** $S$
  
  $$\{ P \} \ S \ { Q}$$

  - **precondition** $P$
  - **postcondition** $Q$
  - 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.

- $\{ \text{true} \} \ x = 12 \ { x == 12 }$
- $\{ y >= 0 \} \ x = 12 \ { x == 12 }$
- $\{ \text{true} \} \ x = 12 \ { x >= 0 }$
  
  (Programs satisfy many possible specifications)
- $\{ x < 10 \} \ x = x + 1 \ { x < 11 }$
- $\{ n >= 0 \} \ x = \text{fact}(n) \ { x == n ! }$
- $\{ \text{true} \} \ a = 0; \text{if}(x != 0) \ { a = 2 \cdot x } \ { a == 2 \cdot x }$

## 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!

## Assignment Rule

- **Rule for assignment**
  $$\{ Q(x := E) \} \ x = E \ { Q}$$

- **Examples:**
  - $\{ 12 == 12 \} \ x = 12 \ { x == 12 }$
  - $\{ 12 >= 0 \} \ x = 12 \ { x >= 0 }$
  - $\{ x + 1 \} \ x = x + 1 \ { ? }$
  - $\{ x >= 1 \} \ x = x + 1 \ { ? }$

## Relaxing Specifications

- **Consider** $\{ x >= 1 \} \ x = x + 1 \ { x >= 2 }$
  - It is very tight specification. We can relax it

## Examples:

- $\{ P \} \ x = E \ { Q}$
  - If $P \Rightarrow Q(x := E)$
  - Example: $\{ x >= 5 \} \ x = x + 1 \ { x >= 2 }$

- Since $x >= 5 \Rightarrow x + 1 >= 2$
Assignments: forward and backward
• Two ways to look at the rules:
  – Backward: given post-condition, what is pre-condition?
    \[
    \begin{align*}
    x &= E \\
    \{ Q \}
    \end{align*}
    \]
  – Forward: given pre-condition, what is post-condition?
    \[
    \begin{align*}
    x &= E \\
    \{ P \}
    \end{align*}
    \]

Example of running it forward
• \{ x == y \} \ x = x + 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:

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

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 \( wp \)
  - 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 \( wlp \) 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
- \( sp(x := E, P) = … \)
- In general:

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

if \( sp(S, P) \Rightarrow Q \)

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:

\[
\begin{array}{c}
x = x - 1; \\
y = y - 1 \\
\{ x \geq y \} \\
\end{array}
\]

Composing Specifications

- If \( \{ P \} S_1 \{ R \} \) and \( \{ R \} S_2 \{ Q \} \) then \( \{ P \} S_1 ; S_2 \{ Q \} \)
- Example:

\[
\begin{array}{c}
x = x - 1; \\
\{ x \geq y - 1 \} \\
y = y - 1 \\
\{ x \geq y \} \\
\end{array}
\]
In terms of wp and sp

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

Conditionals

- Rule for the conditional (flow graph)

\[
\begin{aligned}
&\text{if } P \land E \Rightarrow P_1 \\
\Rightarrow &\{ P \} \\
&\text{if } P \land \neg E \Rightarrow P_2 \\
\Rightarrow &\{ P_2 \}
\end{aligned}
\]

- Example:

\[
\begin{aligned}
&\{ x = 0 \} \\
&\text{if } P \land E \Rightarrow P_1 \\
\Rightarrow &\{ x = 0 \} \\
&\text{if } P \land \neg E \Rightarrow P_2 \\
\Rightarrow &\{ x = 1 \}
\end{aligned}
\]

since \( x = 0 \) \& \( x = 0 \) \Rightarrow \( x = 0 \) since \( x = 0 \) \& \( x \neq 0 \) \Rightarrow \( x = 1 \)

Conditionals: Forward and Backward

- Recall: rule for the conditional

\[
\begin{aligned}
&\text{if } P \land E \Rightarrow P_1 \\
\Rightarrow &\{ P \} \\
&\text{if } P \land \neg E \Rightarrow P_2 \\
\Rightarrow &\{ P_2 \}
\end{aligned}
\]

- Forward: given \( P_1 \) 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:

\[
\begin{aligned}
&\{ P_1 \} \\
\Rightarrow &\{ P_2 \} \\
\Rightarrow &\{ P \}
\end{aligned}
\]

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

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

\[ x = E \]
\[ \{ P \} \]

if \( P \Rightarrow Q(x=E) \)

\[ \{ P \} \]
\[ \{ P_1 \} \]
\[ \{ P_2 \} \]

if \( P \land E \Rightarrow P \)

\[ (P_1) \]
\[ (P_2) \]

if \( P \land E \Rightarrow P \)

Implication is always in the direction of the control flow

Review: forward

\[ x = E \]
\[ \{ P \} \]

\[ \{ P_1 \} \]
\[ \{ P_2 \} \]

\[ \{ P_1 \} \]
\[ \{ P \} \]

\[ \{ P_1 \} \]
\[ \{ P_2 \} \]

\[ \{ P \land E \} \]
\[ \{ P \land \neg E \} \]

Review: backward

\[ x = E \]
\[ \{ Q(x=E) \} \]

\[ \{ P \} \]
\[ \{ P_1 \} \]
\[ \{ P_2 \} \]

\[ \{ P_1 \} \]
\[ \{ P \} \]

\[ \{ P_1 \} \]
\[ \{ P_2 \} \]

\[ \{ E \Rightarrow P_1 \} \land \{ \neg E \Rightarrow P_2 \} \]

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

\[ x < 0 \]
\[ x = -x \]
\[ T \ F \]

\[ c > 0 \]
\[ c-- \]
\[ T \ F \]

Example: Absolute value

\[ x < 0 \Rightarrow P \]
\[ (x < 0 \Rightarrow P) \land \neg x \Rightarrow P \]
\[ x \Rightarrow P \]
\[ \neg x \Rightarrow P \]
\[ x < 0 \land (x > 0) \]

Example: Absolute value

\[ x < 0 \Rightarrow P \]
\[ (x < 0 \Rightarrow P) \land \neg x \Rightarrow P \]
\[ x \Rightarrow P \]
\[ \neg x \Rightarrow P \]
\[ x < 0 \land (x > 0) \]

Ask ATP to show:

\[ \text{Timed} \Rightarrow \neg P' \]
In Simplify

\[
\begin{align*}
\text{So far...} & \\
\text{• Framework for checking pre and post conditions of computations without loops} & \\
\text{• Suppose we want to check that some condition holds inside the computation, rather than at the end} & \\
\text{Say we want to check that } x > 0 \text{ here} & \\
\text{Example: Absolute value with assert} & \\
\end{align*}
\]

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
- Say we want to check that \( x > 0 \) here

Example: Absolute value with assert

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

\[
T \quad \neg x < 0 \\
\neg x = -x \\
\text{assert}(x > 0)
\]

Adding the postcondition back in

\[
T \quad x < 0 \quad \neg x > 0 \\
\neg x = -x \\
\text{assert}(x > 0)
\]

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

\[
\{ \neg \text{locked} \land P[\text{locked := true}] \} \text{ lock } \{ P \} \\
\text{-- lock behaves as assert}(\neg \text{locked}); \text{locked} = \text{true}
\]

\[
\{ \text{locked} \land P[\text{locked := false}] \} \text{ unlock } \{ P \} \\
\text{-- unlock behaves as assert}((\neg \text{locked}); \text{locked} = \text{false}
\]

Locking Example

\[
\{ \neg \text{L} \land P[\text{L := true}] \} \text{ lock } \{ P \} \\
\{ \text{L} \land P[\text{L := false}] \} \text{ unlock } \{ P \}
\]

\[
\{ \neg \text{L} \land \neg x < 0 \} \\
\{ x = 0 \} \\
\{ \text{L} \land \neg x < 0 \} \\
\{ \neg x = 0 \} \\
\{ \text{L} \land x = 0 \}
\]

\[
\{ \neg \text{L} \land \neg x < 0 \} \\
\{ x = 0 \} \\
\{ \text{L} \land \neg x < 0 \} \\
\{ \neg x = 0 \} \\
\{ \text{L} \land x = 0 \}
\]
Locking Example: forward direction