Mechanized Operational Semantics

J Strother Moore
Department of Computer Sciences
University of Texas at Austin

Marktoberdorf Summer School 2008

(Lecture 4: Inductive Invariant Proofs)
Program

Pre-condition: $P(s)$

Loop invariant: $R(s)$

Post-condition: $Q(s)$

Labels: $\alpha$, $\beta$, $\gamma$

Program $\pi$

Paths:
- $f(s)$
- $g(s)$
- $h(s)$

Assertions:
- $P(s)$: pre-condition
- $R(s)$: loop invariant
- $Q(s)$: post-condition

Paths labels:
- $t$
Conventional Mechanized Code Proofs

<table>
<thead>
<tr>
<th>labels</th>
<th>program $\pi$</th>
<th>paths</th>
<th>assertions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td></td>
<td>$f(s)$</td>
<td>$P(s)$ pre-condition</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$g(s)$</td>
<td>$t$</td>
<td>$R(s)$ loop invariant</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>RETURN</td>
<td>$h(s)$</td>
<td>$Q(s)$ post-condition</td>
</tr>
</tbody>
</table>

VC1. $P(s) \rightarrow R(f(s))$, 
Conventional Mechanized Code Proofs

VC1. \( P(s) \rightarrow R(f(s)), \)

VC2. \( R(s) \land t \rightarrow R(g(s)), \) and
Conventional Mechanized Code Proofs

VC1. \( P(s) \rightarrow R(f(s)) \),

VC2. \( R(s) \land t \rightarrow R(g(s)) \), and

VC3. \( R(s) \land \neg t \rightarrow Q(h(s)) \).
Conventional Mechanized Code Proofs

The process by which proof obligations (verification conditions or VCs) are generated from the code is called verification condition generation and is performed by a VCG program.

Typically, VCGs simplify the VC “on-the-fly.”

Typically, the language semantics is coded into the VCG.

These are a common sources of errors.
Conventional Mechanized Code Proofs

To do conventional mechanized code proofs you need:

- a Hoare semantics
- a VCG (driven off the semantics)
- a theorem prover
Conventional Mechanized Code Proofs

To do conventional mechanized code proofs you need:

- an operational semantics
- a theorem prover
Operational Semantics (Revisited)

The *semantics* of the programming language may be given by a function $\text{run}$ which “interprets” a program against some state and determines the “final” state.

\[
\text{run} (k, s) = \begin{cases} 
  s & \text{if } k = 0 \\
  \text{run} (k - 1, \text{step} (s)) & \text{otherwise}
\end{cases}
\]

Here, $\text{step}$ is the single step state transition function.
labels | program $\pi$ | paths | assertions
--- | --- | --- | ---
$\alpha$ | | $f(s)$ | $P(s)$ pre-condition
$\beta$ | $g(s)$ | $R(s)$ loop invariant
$\gamma$ | $h(s)$ | $Q(s)$ post-condition

RETURN
We assume the program in $s$, $\pi$, does not change during execution.

Let $s_0$ be the initial state of program $\pi$.

$pc(s_0) = \alpha$

Let $s_k$ denote $run(k, s_0)$. 
Formally Stated Correctness Theorems

Total:

\[ \exists k : P(s_0) \rightarrow (Q(s_k) \land pc(s_k) = \gamma). \]

\[ \exists k : P(s_0) \rightarrow (Q(run(k, s_0)) \land \ldots. \]

This is sometimes stated without the quantifier as

\[ P(s_0) \rightarrow (Q(run(sched(s_0), s_0)) \land \ldots). \]
Partial:

\[ P(s_0) \land pc(s_k) = \gamma \rightarrow Q(s_k). \]
Disadvantage of Direct Proofs

Direct proofs of program properties can be complicated (or at least appear so) because of the presence of the interpreter, the program counter, the entire machine state, and the need to define a schedule function.

The inductive assertion method produces such nice proof obligations!
Conundrum

Can you prove

\[ P(s_0) \land pc(s_k) = \gamma \rightarrow Q(s_k). \]

directly – where the only heavy-duty proof work is proving the verification conditions?

Do you need a trusted VCG?

Can you make the automatic proof attempt generate the standard verification conditions from the operational semantics?
Caveat

The observations I make below are not deep, but I think they have important practical implications:
**Theorem:** \( P(s_0) \land pc(s_k) = \gamma \rightarrow Q(s_k) \)

**Proof:** Define

\[
Inv(s) \equiv \begin{cases} 
    P(s) & \text{if } pc(s) = \alpha \\
    R(s) & \text{if } pc(s) = \beta \\
    Q(s) & \text{if } pc(s) = \gamma \\
    Inv(step(s)) & \text{otherwise}
\end{cases}
\]

(Actually, we assert "\( prog(s) = \pi \)" at \( \alpha, \beta \) and \( \gamma \), but we omit that here by our convention that the program is always \( \pi \).)
Theorem: $P(s_0) \land pc(s_k) = \gamma \rightarrow Q(s_k)$

Proof: Define

$$Inv(s) \equiv \begin{cases} 
P(s) & \text{if } pc(s) = \alpha \\
R(s) & \text{if } pc(s) = \beta \\
Q(s) & \text{if } pc(s) = \gamma \\
Inv(step(s)) & \text{otherwise}
\end{cases}$$

Objection: Is this definition consistent? Yes: Every tail-recursive definition is witnessed by a total function. (Manolios and Moore, 2000)
Theorem: \[ P(s_0) \land pc(s_k) = \gamma \rightarrow Q(s_k) \]

Proof: Define

\[
Inv(s) \equiv \begin{cases} 
    P(s) & \text{if } pc(s) = \alpha \\
    R(s) & \text{if } pc(s) = \beta \\
    Q(s) & \text{if } pc(s) = \gamma \\
    Inv(step(s)) & \text{otherwise} 
\end{cases}
\]

Assume we’ve proved

\[ Inv(s) \rightarrow Inv(step(s)). \]

(We’ll see the proof in a moment.)
**Theorem:** \( P(s_0) \land pc(s_k) = \gamma \rightarrow Q(s_k) \)

**Proof:** Define

\[
Inv(s) \equiv \begin{cases} 
P(s) & \text{if } pc(s) = \alpha \\
R(s) & \text{if } pc(s) = \beta \\
Q(s) & \text{if } pc(s) = \gamma \\
Inv(step(s)) & \text{otherwise}
\end{cases}
\]

\[Inv(s_0) \rightarrow Inv(s_k) \quad (By \ induction)\]
**Theorem:** $P(s_0) \land pc(s_k) = \gamma \rightarrow Q(s_k)$

**Proof:** Define

$$Inv(s) \equiv \begin{cases} 
P(s) & \text{if } pc(s) = \alpha \\
R(s) & \text{if } pc(s) = \beta \\
Q(s) & \text{if } pc(s) = \gamma \\
Inv(step(s)) & \text{otherwise}
\end{cases}$$

$$Inv(s_0) \rightarrow Inv(s_k)$$

$$pc(s_0) = \alpha \quad (By \ construction)$$
**Theorem:** \( P(s_0) \land pc(s_k) = \gamma \rightarrow Q(s_k) \)

**Proof:** Define

\[
Inv(s) \equiv \begin{cases} 
P(s) & \text{if } pc(s) = \alpha \\
R(s) & \text{if } pc(s) = \beta \\
Q(s) & \text{if } pc(s) = \gamma \\
Inv(step(s)) & \text{otherwise}
\end{cases}
\]

\[P(s_0) \rightarrow Inv(s_k)\]
Theorem: \[ P(s_0) \land pc(s_k) = \gamma \rightarrow Q(s_k) \]

Proof: Define

\[ Inv(s) \equiv \begin{cases} 
  P(s) & \text{if } pc(s) = \alpha \\
  R(s) & \text{if } pc(s) = \beta \\
  Q(s) & \text{if } pc(s) = \gamma \\
  Inv(\text{step}(s)) & \text{otherwise}
\end{cases} \]

\[ P(s_0) \rightarrow Inv(s_k) \]

\[ P(s_0) \quad (Given) \]
Theorem: \( P(s_0) \land pc(s_k) = \gamma \rightarrow Q(s_k) \)

Proof: Define

\[
Inv(s) \equiv \begin{cases} 
    P(s) & \text{if } pc(s) = \alpha \\
    R(s) & \text{if } pc(s) = \beta \\
    Q(s) & \text{if } pc(s) = \gamma \\
    Inv(step(s)) & \text{otherwise}
\end{cases}
\]

\( Inv(s_k) \)
**Theorem:** \( P (s_0) \land pc (s_k) = \gamma \rightarrow Q (s_k) \)

**Proof:** Define

\[
Inv (s) \equiv \begin{cases} 
P (s) & \text{if } pc (s) = \alpha \\
R (s) & \text{if } pc (s) = \beta \\
Q (s) & \text{if } pc (s) = \gamma \\
Inv (\text{step} (s)) & \text{otherwise}
\end{cases}
\]

\[Inv (s_k)\]

\[pc (s_k) = \gamma \quad (Given)\]
Theorem: \( P(s_0) \land pc(s_k) = \gamma \rightarrow Q(s_k) \)

Proof: Define

\[
Inv(s) \equiv \begin{cases} 
  P(s) & \text{if } pc(s) = \alpha \\
  R(s) & \text{if } pc(s) = \beta \\
  Q(s) & \text{if } pc(s) = \gamma \\
  Inv(step(s)) & \text{otherwise}
\end{cases}
\]

\( Q(s_k) \)

Q.E.D.
So it’s trivial to prove the theorem

\[ P(s_0) \land pc(s_k) = \gamma \rightarrow Q(s_k) \]

if we can prove

\[ Inv(s) \rightarrow Inv(step(s)). \]
\[ Inv(s) \equiv \begin{cases} 
  P(s) & \text{if } pc(s) = \alpha \\
  R(s) & \text{if } pc(s) = \beta \\
  Q(s) & \text{if } pc(s) = \gamma \\
  Inv(step(s)) & \text{otherwise} 
\end{cases} \]
$$\text{Inv}(s) \rightarrow \text{Inv}(\text{step}(s))$$

**Proof.**

Expanding \(\text{Inv}(s)\) generates four cases:

- Case \(pc(s) = \alpha\):
- Case \(pc(s) = \beta\):
- Case \(pc(s) = \gamma\):
- Case *otherwise*:
$\text{Inv}(s) \rightarrow \text{Inv}(\text{step}(s))$

[Case $\text{pc}(s) = \alpha$

labels \hspace{1cm} program $\pi$ \hspace{1cm} paths
\begin{align*}
\alpha & \quad \_ \_ \_ \_ \_ \\
\beta & \quad \_ \_ \_ \_ \_ \\
\gamma & \quad \text{RETURN}
\end{align*}

assertions
\begin{align*}
P(s) & \quad \text{pre-condition} \\
R(s) & \quad \text{loop invariant}
\end{align*}

\begin{align*}
f(s) \\
g(s) \\
h(s) \\
Q(s) & \quad \text{post-condition}
\end{align*}
\( P(s) \rightarrow Inv(step(s)) \)

\([Case \ pc(s) = \alpha]\)

- **labels**
  - \( \alpha \)
  - \( \beta \)
  - \( \gamma \)

- **program \( \pi \)**
  - \( f(s) \)
  - \( g(s) \)
  - \( h(s) \)

- **paths**
  - \( t \)

- **assertions**
  - \( P(s) \) pre-condition
  - \( R(s) \) loop invariant
  - \( Q(s) \) post-condition
\[ Inv(s) \equiv \begin{cases} 
  P(s) & \text{if } pc(s) = \alpha \\
  R(s) & \text{if } pc(s) = \beta \\
  Q(s) & \text{if } pc(s) = \gamma \\
  Inv(step(s)) & \text{otherwise} 
\end{cases} \]

\[ Inv(s) = Inv(step(s)) = Inv(step(step(s))) \ldots \]

as long as the \( pc \notin \{\alpha, \beta, \gamma\} \).
\[ P(s) \rightarrow Inv \left( step \left( s \right) \right) \]

\[ [Case \ \text{pc}(s) = \alpha] \]

<table>
<thead>
<tr>
<th>labels</th>
<th>program ( \pi )</th>
<th>paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>( f(s) )</td>
<td>( g(s) )</td>
</tr>
<tr>
<td>( \beta )</td>
<td>( h(s) )</td>
<td></td>
</tr>
<tr>
<td>( \gamma )</td>
<td>( \text{RETURN} )</td>
<td></td>
</tr>
</tbody>
</table>

assertions

- \( P(s) \) pre–condition
- \( R(s) \) loop invariant
- \( Q(s) \) post–condition
\[ P(s) \rightarrow R(f(s)) \]

[Case \( pc(s) = \alpha \)]

- **Labels**: \( \alpha, \beta, \gamma \)
- **Program**: \( \pi \)
- **Paths**: \( f(s), g(s), h(s) \)
- **Assertions**: 
  - \( P(s) \): pre-condition
  - \( R(s) \): loop invariant
  - \( Q(s) \): post-condition

**RETURN** program
$$Inv(s) \rightarrow Inv(step(s))$$

**[Case pc(s) = \beta]**

- **Labels:** α, β, γ
- **Program:** π
- **Paths:** $f(s)$, $g(s)$, $h(s)$, $t$

- **Assertions:**
  - Pre-condition: $P(s)$
  - Loop invariant: $R(s)$
  - Post-condition: $Q(s)$
\[(R(s) \land t \rightarrow R(g(s)))\]
\[(R(s) \land \neg t \rightarrow Q(h(s)))\]

**Labels**
- \(\alpha\)
- \(\beta\)
- \(\gamma\)

**Program** \(\pi\)

**Paths**
- \(f(s)\)
- \(t\)

**Assertions**
- \(P(s)\) pre-condition
- \(R(s)\) loop invariant
- \(Q(s)\) post-condition

[C]ase \(pc(s) = \beta\)
$\text{Inv}(s) \rightarrow \text{Inv}(\text{step}(s))$  

[Case $pc(s) = \gamma$]

<table>
<thead>
<tr>
<th>labels</th>
<th>program $\pi$</th>
<th>paths</th>
<th>assertions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td></td>
<td></td>
<td>$P(s)$ pre-condition</td>
</tr>
<tr>
<td>$\beta$</td>
<td></td>
<td></td>
<td>$R(s)$ loop invariant</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>RETURN</td>
<td></td>
<td>$Q(s)$ post-condition</td>
</tr>
</tbody>
</table>

$\text{RETURN}$
\[ \text{Inv}(s) \rightarrow \text{Inv}(s) \]

\[ \text{[Case } pc(s) = \gamma ] \]

- **labels**: \( \alpha \), \( \beta \), \( \gamma \)
- **program \( \pi \)**
- **paths**: \( f(s) \), \( g(s) \), \( h(s) \)
- **assertions**: \( P(s) \) (pre-condition), \( R(s) \) (loop invariant), \( Q(s) \) (post-condition)
\( Inv(s) \rightarrow Inv(\text{step}(s)) \)

[Case otherwise]

- **labels**: \( \alpha \), \( \beta \), \( \gamma \)
- **program** \( \pi \)
- **paths**: \( f(s) \), \( g(s) \), \( h(s) \), \( t \)

- **assertions**
  - \( P(s) \) pre-condition
  - \( R(s) \) loop invariant
  - \( Q(s) \) post-condition
\[ \text{Inv (step (s))} \rightarrow \text{Inv (step (s))} \text{ [Case otherwise]} \]

<table>
<thead>
<tr>
<th>labels</th>
<th>program π</th>
<th>paths</th>
<th>assertions</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td></td>
<td></td>
<td>( P(s) ) pre-condition</td>
</tr>
<tr>
<td>β</td>
<td></td>
<td></td>
<td>( R(s) ) loop invariant</td>
</tr>
<tr>
<td>γ</td>
<td>RETURN</td>
<td></td>
<td>( Q(s) ) post-condition</td>
</tr>
</tbody>
</table>

\( f(s) \), \( g(s) \), \( h(s) \), \( t \)
**Recap:** Given the definition of $\text{Inv}$, the “natural” proof of

$$\text{Inv}(s) \rightarrow \text{Inv}(\text{step}(s))$$

generates the standard verification conditions

**VC1.** $P(s) \rightarrow R(f(s))$,
**VC2.** $R(s) \land t \rightarrow R(g(s))$, and
**VC3.** $R(s) \land \neg t \rightarrow Q(h(s))$

as subgoals from the operational semantics!

It generates no other non-trivial proof obligations.

The VCs are simplified as they are generated.
Demo 1
Discussion

We did not write a VCG for M1.

The VCs were generated directly from the operational semantics by the theorem prover.

Since VCs are generated by proof, the paths explored and the VCs generated are sensitive to the pre-condition specified.

The VCs are simplified (and possibly proved) by the same process.

We did not count instructions or define a schedule.
We did not constrain the inputs so that the program terminated.

Indeed, we can deal with non-terminating programs.
Demo 2
Total Correctness via Inductive Assertions

We have also handled total correctness via the VCG approach.

An ordinal measure is provided at each cut point and the VCs establish that it decreases upon each arrival at the cut point.

Schedule functions can be automatically generated and admitted from such proofs.
Primary Citation

Other Examples

Nested loops are handled exactly as by standard VCG methods.

```java
public static int tfact(int n) {
    /* Factorial by repeated addition. */
    /* Verified using inductive assertions */
    /* by Alan Turing, 1949. */
    int i = 1;
    int b = 1;
    while (i <= n) {
        int j = 1;
        int a = b;
        while (j < i) {
            b = a + b;
            j++;
        }
        i++;
    }
    return b;
}
```
Recursive methods can be handled.

```java
public static int fact(int n) {
    if (n > 0) {
        return n * fact(n - 1);
    } else return 1;
}
```

To handle recursive methods we

- modify run to terminate upon top-level return, and
- add a standard invariant about the shape of the call stack.
Conclusion

If you have

• a theorem prover and

• a formal operational semantics,

you can prove formally stated *partial program correctness* theorems using *inductive assertions* without building or verifying a VCG.
Related Work


Next Time

a much more interesting correctness proof