07—Program Verification

CS 5209: Foundation in Logic and AI

Martin Henz and Aquinas Hobor

March 4, 2010

Generated on Thursday 11th March, 2010, 16:17
1. Core Programming Language

2. Hoare Triples; Partial and Total Correctness

3. Proof Calculus for Partial Correctness
Core Programming Language
Hoare Triples; Partial and Total Correctness
Proof Calculus for Partial Correctness

Motivation

One way of checking the correctness of programs is to explore the possible states that a computation system can reach during the execution of the program.

Problems with this *model checking* approach:
- Models become infinite.
- Satisfaction/validity becomes undecidable.

In this lecture, we cover a proof-based framework for program verification.
Characteristics of the Approach

Proof-based instead of model checking
Semi-automatic instead of automatic
Property-oriented not using full specification
Application domain fixed to sequential programs using integers
Interleaved with development rather than a-posteriori verification
Reasons for Program Verification

Documentation. Program properties formulated as theorems can serve as concise documentation.

Time-to-market. Verification prevents/catches bugs and can reduce development time.

Reuse. Clear specification provides basis for reuse.

Certification. Verification is required in safety-critical domains such as nuclear power stations and aircraft cockpits.
Framework for Software Verification

Convert informal description $R$ of requirements for an application domain into formula $\phi_R$.

Write program $P$ that meets $\phi_R$.

Prove that $P$ satisfies $\phi_R$.

Each step provides risks and opportunities.
Core Programming Language

Hoare Triples; Partial and Total Correctness

Proof Calculus for Partial Correctness
Motivation of Core Language

- Real-world languages are quite large; many features and constructs
- Verification framework would exceed time we have in CS5209
- Theoretical constructions such as Turing machines or lambda calculus are too far from actual applications; too low-level
- Idea: use subset of Pascal/C/C++/Java
- Benefit: we can study useful “realistic” examples
Expressions in Core Language

Expressions come as arithmetic expressions $E$:

$$E ::= n \mid x \mid (\neg E) \mid (E + E) \mid (E - E) \mid (E \ast E)$$

and boolean expressions $B$:

$$B ::= \text{true} \mid \text{false} \mid (!B) \mid (B \& B) \mid (B \| B) \mid (E < E)$$

Where are the other comparisons, for example $==$?
Commands in Core Language

Commands cover some common programming idioms. Expressions are components of commands.

\[ C ::= x = E \mid C; C \mid \text{if } B \{ C \} \text{ else } \{ C \} \mid \text{while } B \{ C \} \]
Example

Consider the factorial function:

\[
0! \overset{\text{def}}{=} 1
\]

\[
(n + 1)! \overset{\text{def}}{=} (n + 1) \cdot n!
\]

We shall show that after the execution of the following Core program, we have \( y = x! \).

\[
y = 1;
\]

\[
z = 0;
\]

\[
\text{while } (z \neq x) \{ z = z + 1; \ y = y \ast z; \} \]
1. Core Programming Language
2. Hoare Triples; Partial and Total Correctness
3. Proof Calculus for Partial Correctness
Example

\[ y = 1; \]
\[ z = 0; \]
\[ \textbf{while} \ (z \neq x) \ \{ \ z = z + 1; \ y = y \times z; \ \} \]
Example

```
y = 1;
z = 0;
while (z != x) {
  z = z + 1;
  y = y * z;
}
```

We need to be able to say that at the end, \( y \) is \( x \)!
Example

\[
y = 1; \\
z = 0; \\
\textbf{while} (z \neq x) \{ z = z + 1; y = y \times z; \}
\]

- We need to be able to say that at the end, \( y \) is \( x! \).
- That means we require a \textit{post-condition} \( y = x! \).
Example

\[
y = 1; \\
z = 0; \\
\textbf{while} (z \neq x) \{ z = z + 1; y = y \ast z; \}
\]

- Do we need pre-conditions, too?
Example

```
y = 1;
z = 0;
while (z != x) {
    z = z + 1;
    y = y * z;
}
```

Do we need pre-conditions, too?

Yes, they specify what needs to be the case before execution.

Example: $x > 0$
Example

\[ \begin{align*}
y &= 1; \\
z &= 0; \\
\textbf{while} (z \neq x) \{ & z = z + 1; \ y = y \ast z; \}
\end{align*} \]

- Do we need pre-conditions, too?
  Yes, they specify what needs to be the case before execution.
  Example: \( x > 0 \)

- Do we have to prove the postcondition in one go?
Example

```plaintext
y = 1;
z = 0;
while (z != x) {
    z = z + 1; y = y * z;
}
```

- Do we need pre-conditions, too?
  Yes, they specify what needs to be the case before execution.
  Example: \(x > 0\)

- Do we have to prove the postcondition in one go?
  No, the postcondition of one line can be the pre-condition of the next!
Assertions on Programs

Shape of assertions

\[(\phi) \quad P \quad (\psi)\]

Informal meaning

If the program $P$ is run in a state that satisfies $\phi$, then the state resulting from $P$'s execution will satisfy $\psi$. 
Informal specification
Given a positive number $x$, the program $P$ calculates a number $y$ whose square is less than $x$.

Assertion

$(x > 0) \quad P \quad (y \cdot y < x)$

Example for $P$

$y = 0$

Our first Hoare triple

$(x > 0) \quad y = 0 \quad (y \cdot y < x)$
(Slightly Less Trivial) Example

Same assertion

\[(x > 0) \quad P \quad (y \cdot y < x)\]

Another example for \(P\)

\[
y = 0;
while \quad (y \cdot y < x) \quad \{
    \quad y = y + 1;
\}
\]
\[
y = y - 1;
\]
Recall: Models in Predicate Logic

Definition

Let $\mathcal{F}$ contain function symbols and $\mathcal{P}$ contain predicate symbols. A model $\mathcal{M}$ for $(\mathcal{F}, \mathcal{P})$ consists of:

1. A non-empty set $A$, the *universe*;
2. for each nullary function symbol $f \in \mathcal{F}$ a concrete element $f^\mathcal{M} \in A$;
3. for each $f \in \mathcal{F}$ with arity $n > 0$, a concrete function $f^\mathcal{M} : A^n \to A$;
4. for each $P \in \mathcal{P}$ with arity $n > 0$, a set $P^\mathcal{M} \subseteq A^n$. 
Recall: Satisfaction Relation

The model \( M \) satisfies \( \phi \) with respect to environment \( l \), written \( M \models_l \phi \):

- in case \( \phi \) is of the form \( P(t_1, t_2, \ldots, t_n) \), if the result \( (a_1, a_2, \ldots, a_n) \) of evaluating \( t_1, t_2, \ldots, t_n \) with respect to \( l \) is in \( P^M \);
- in case \( \phi \) has the form \( \forall x \psi \), if the \( M \models_l [x \mapsto a] \psi \) holds for all \( a \in A \);
- in case \( \phi \) has the form \( \exists x \psi \), if the \( M \models_l [x \mapsto a] \psi \) holds for some \( a \in A \);
Recall: Satisfaction Relation (continued)

- in case \( \phi \) has the form \( \neg \psi \), if \( M \models_I \psi \) does not hold;
- in case \( \phi \) has the form \( \psi_1 \lor \psi_2 \), if \( M \models_I \psi_1 \) holds or \( M \models_I \psi_2 \) holds;
- in case \( \phi \) has the form \( \psi_1 \land \psi_2 \), if \( M \models_I \psi_1 \) holds and \( M \models_I \psi_2 \) holds; and
- in case \( \phi \) has the form \( \psi_1 \to \psi_2 \), if \( M \models_I \psi_1 \) holds whenever \( M \models_I \psi_2 \) holds.
Hoare Triples

Definition
An assertion of the form $(\phi) \ P \ (\psi)$ is called a Hoare triple.

- $\phi$ is called the precondition, $\psi$ is called the postcondition.
- A state of a Core program $P$ is a function $l$ that assigns each variable $x$ in $P$ to an integer $l(x)$.
- A state $l$ satisfies $\phi$ if $M \models_I \phi$, where $M$ contains integers and gives the usual meaning to the arithmetic operations.
- Quantifiers in $\phi$ and $\psi$ bind only variables that do not occur in the program $P$. 
Example

Let \( l(x) = -2, \ l(y) = 5 \) and \( l(z) = -1 \). We have:

- \( l \models \neg (x + y < z) \)
- \( l \not\models y = x \cdot z < z \)
- \( l \not\models \forall u(y < u \rightarrow y \cdot z < u \cdot z) \)
Partial Correctness

Definition
We say that the triple $(\phi) \ P \ (\psi)$ is satisfied under partial correctness if, for all states which satisfy $\phi$, the state resulting from $P$’s execution satisfies $\psi$, provided that $P$ terminates.

Notation
We write $\models_{\text{par}} (\phi) \ P \ (\psi)$.
Extreme Example

\((\phi)\) while true \{ x = 0; \} (\psi)

holds for all \(\phi\) and \(\psi\).
Total Correctness

Definition
We say that the triple $(\varphi \mid P \mid \psi)$ is satisfied under total correctness if, for all states which satisfy $\varphi$, $P$ is guaranteed to terminate and the resulting state satisfies $\psi$.

Notation
We write $\models_{\text{tot}} (\varphi \mid P \mid \psi)$. 
Consider \textbf{Fac1}:

\begin{verbatim}
y = 1;
z = 0;
while (z != x) {
z = z + 1; y = y * z;
}
\end{verbatim}
Consider \texttt{Fac1}:

\begin{verbatim}
    y = 1;
z = 0;
while (z != x) {
    z = z + 1; y = y * z;
}
\end{verbatim}

\[
\models_{\text{tot}} (x \geq 0) \texttt{Fac1} (|y = x!|)
\]
Back to Factorial

Consider $\text{Fac1}$:

\[
y = 1; \\
z = 0; \\
\textbf{while} (z \neq x) \{ z = z + 1; y = y \times z; \}
\]

- $\models_{\text{tot}} (x \geq 0) \text{Fac1} (y = x!)$
- $\not\models_{\text{tot}} (\top) \text{Fac1} (y = x!)$
Consider \texttt{Fac1}:

\begin{verbatim}
    y = 1;
z = 0;
while (z \neq x) {
    z = z + 1; y = y * z;
}
\end{verbatim}

\( \models_{\text{tot}} (x \geq 0) \text{ Fac1 } (y = x!) \)

\( \not\models_{\text{tot}} (\top) \text{ Fac1 } (y = x!) \)

\( \models_{\text{par}} (x \geq 0) \text{ Fac1 } (y = x!) \)
Consider $\text{Fac1}$:

$$y = 1;$$
$$z = 0;$$
$$\textbf{while} \ (z \neq x) \ \{ \ z = z + 1; \ y = y \ast z; \ \}$$

- $\vdash_{\text{tot}} (x \geq 0) \ Fac1 (y = x!)$
- $\nvdash_{\text{tot}} (\top) \ Fac1 (y = x!)$
- $\vdash_{\text{par}} (x \geq 0) \ Fac1 (y = x!)$
- $\vdash_{\text{par}} (\top) \ Fac1 (y = x!)$
Strategy

We are looking for a proof calculus that allows us to establish

$$\vdash_{\text{par}} (\phi) \ P \ (\psi)$$

where

- $$\vdash_{\text{par}} (\phi) \ P \ (\psi)$$ holds whenever $$\vdash_{\text{par}} (\phi) \ P \ (\psi)$$ (correctness), and
- $$\vdash_{\text{par}} (\phi) \ P \ (\psi)$$ holds whenever $$\vdash_{\text{par}} (\phi) \ P \ (\psi)$$ (completeness).
Rules for Partial Correctness

\[ (\phi) \ C_1 \ (\eta) \quad (\eta) \ C_2 \ (\psi) \]

\[ \frac{}{[\text{Composition}] \quad (\phi) \ C_1 ; C_2 \ (\psi)} \]
Rules for Partial Correctness (continued)

[Assignment]

\((\lbrack x \rightarrow E \rbrack \psi) \rightarrow x = E \psi)
Examples

Let P be the program \( x = 2 \).

Using

\[
\frac{\mathcal{v}}{x = E \mathcal{v}} \]

we can prove:

- \((2 = 2) \ P \ (x = 2)\)
- \((2 = 4) \ P \ (x = 4)\)
- \((2 = y) \ P \ (x = y)\)
- \((2 > 0) \ P \ (x > 0)\)
More Examples

Let $P$ be the program $x = x + 1$.

Using

\[
([x \to E] \psi) \ x = E \ (\psi)\]

we can prove:

- $(x + 1 = 2) \ P \ (x = 2)$
- $(x + 1 = y) \ P \ (x = y)$
Rules for Partial Correctness (continued)

\[
\begin{align*}
(\phi \land B) \ & C_1 \ (\psi) \quad \quad (\phi \land \neg B) \ & C_2 \ (\psi) \\
\hline
\ (\psi) \ & \text{if } B \ \{ \ C_1 \ \} \ \text{else} \ \{ \ C_2 \ \} \ (\psi) \\
\end{align*}
\]

[If-statement]

\[
\begin{align*}
(\psi \land B) \ & C \ (\psi) \\
\hline
\ (\psi) \ & \text{while } B \ \{ \ C \ \} \ (\psi \land \neg B) \\
\end{align*}
\]

[Partial-while]
Rules for Partial Correctness (continued)

\[ \vdash_{AR} \phi' \rightarrow \phi \quad (\phi) \ C \ (\psi) \quad \vdash_{AR} \psi \rightarrow \psi' \]

\[ \frac{\vdash_{AR} \phi' \rightarrow \phi \quad (\phi) \ C \ (\psi) \quad \vdash_{AR} \psi \rightarrow \psi'}{(\phi') \ C \ (\psi')} \text{ [Implied]} \]
Next Week

Lecture 8: Total Correctness; Programming by Contract; Semantics of Hoare Logic