10—Program Verification

CS 3234: Logic and Formal Systems

Aquinas Hobor and Martin Henz

October 21, 2010

Generated on Friday 22nd October, 2010, 08:16
1 Core Programming Language

2 Hoare Triples; Partial and Total Correctness

3 Proof Calculus for Partial Correctness
Program Verification

**Specification**  Documenting and formalizing how a program should behave
Program Verification

**Specification**  Documenting and formalizing how a program should behave

**Proof**  Demonstrating that a program behaves as specified
Reasons for Program Verification

**Documentation.** Program properties formulated as theorems can serve as concise documentation.
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
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
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 $\phi_R$. 
Framework for Software Verification

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

Write program $P$ that meets $\phi_R$. 
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 $\phi_R$. 
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.
1 Core Programming Language
2 Hoare Triples; Partial and Total Correctness
3 Proof Calculus for Partial Correctness
Motivation of Core Language

- Real-world languages are quite large; many features and constructs
Real-world languages are quite large; many features and constructs

Verification framework would exceed time we have in CS3234
Motivation of Core Language

- Real-world languages are quite large; many features and constructs
- Verification framework would exceed time we have in CS3234
- Theoretical constructions such as Turing machines or lambda calculus are too far from actual applications; too low-level
Motivation of Core Language

- Real-world languages are quite large; many features and constructs
- Verification framework would exceed time we have in CS3234
- 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
Motivation of Core Language

- Real-world languages are quite large; many features and constructs
- Verification framework would exceed time we have in CS3234
- 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 $E$:

$$E ::= z | x | (E + E) | (E - E) | (E * E)$$
Expressions in Core Language

Expressions come as arithmetic expressions $E$:

$$E ::= z \mid x \mid (E + E) \mid (E - E) \mid (E \times E)$$

and boolean expressions $B$:

$$B ::= (E \leq E) \mid (!B) \mid (B || B)$$
Expressions in Core Language

Expressions come as arithmetic expressions $E$:

$$E ::= z \mid x \mid (E + E) \mid (E - E) \mid (E * E)$$

and boolean expressions $B$:

$$B ::= (E \leq E) \mid (!B) \mid (B \mid\mid B)$$

What about other kinds of boolean expressions (e.g., conjunction)?
Commands cover some common programming idioms. Expressions are components of commands.

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

Consider the factorial function:

\[
\begin{align*}
0! & \overset{\text{def}}{=} 1 \\
(n + 1)! & \overset{\text{def}}{=} (n + 1) \cdot n!
\end{align*}
\]

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

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

2. Hoare Triples; Partial and Total Correctness

3. Proof Calculus for Partial Correctness
Example

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

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

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

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

Do we need pre-conditions, too?
```
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: \( x > 0 \)
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: $x > 0$
- 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

\[ \{ \phi \} \; P \; \{ \psi \} \]
## Assertions on Programs

### Shape of assertions

\[ \{ \phi \} \; P \; \{ \psi \} \]

### 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 $x$. 

(Slightly Trivial) Example

Informal specification
Given a positive number $x$, the program $P$ calculates a number $y$ whose square is less than $x$.

Assertion
\[
\{x > 0\} \ P \ \{y \cdot y < x\}\]
Informal specification

Given a positive number $x$, the program $P$ calculates a number $y$ whose square is less than $x$.

Assertion

$$\{x > 0\} \ P \ \{y \cdot y < x\}$$

Example for $P$

$y = 0$
(Slightly Trivial) Example

Informal specification

Given a positive number $x$, the program $P$ calculates a number $y$ whose square is less than $x$.

Assertion

$$\{ x > 0 \} \ P \ \{ y \cdot y < x \}$$

Example for $P$

$y = 0$

Our first Hoare triple

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

Same assertion

\{ x > 0 \} \ P \ \{ y \cdot y < x \} 

Another example for \( P \)

\begin{verbatim}
  y = 0;
  while (y * y < x) {
    y = y + 1;
  }
  y = y - 1;
\end{verbatim}
**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 \( \rho \) that assigns each variable \( x \) in \( P \) to an integer \( l(x) \).
- A state \( \rho \) satisfies \( \phi \) if \( \rho \models \phi \)—that is, we have a modal logic where the truth of \( \phi \) depends on the current state.
Let $\rho(x) = -2$, $\rho(y) = 5$ and $\rho(z) = -1$. We have:

$$\rho \vdash \neg(x + y < z)$$
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.
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 \} \text{ while } \text{true} \ \{ \ x = 0; \ \} \ \{ \psi \}

holds for all $\phi$ and $\psi$. 
Total Correctness

Definition

We say that the triple $\{\phi\} \ P \ {\psi}$ is satisfied under total correctness if, for all states which satisfy $\phi$, $P$ is guaranteed to terminate and the resulting state satisfies $\psi$.

Notation

We write $\models_{\text{tot}} \{\phi\} \ P \ {\psi}$.
Consider \texttt{Fac1}:

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

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

\[
\models_{\text{tot}} \{ x \geq 0 \} \texttt{Fac1} \{ y = x! \}
\]
Consider \texttt{Fac1}:

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

\[
\begin{align*}
\models_{\text{tot}} \{ x \geq 0 \} \texttt{Fac1} \{ y = x! \} \\
\not\models_{\text{tot}} \{ \top \} \texttt{Fac1} \{ y = x! \}
\end{align*}
\]
Back to Factorial

Consider $\text{Fac1}$:

\begin{align*}
y &= 1; \\
z &= 0; \\
\text{while } (z \neq x) \{ z &= z + 1; y = y \times z; \}
\end{align*}

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

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

- $\models_{\text{tot}} \{ x \geq 0 \} \ \text{Fac1} \ \{ y = x! \}$
- $\not\models_{\text{tot}} \{ \top \} \ \text{Fac1} \ \{ y = x! \}$
- $\models_{\text{par}} \{ x \geq 0 \} \ \text{Fac1} \ \{ y = x! \}$
- $\models_{\text{par}} \{ \top \} \ \text{Fac1} \ \{ y = x! \}$
1. Core Programming Language
2. Hoare Triples; Partial and Total Correctness
3. Proof Calculus for Partial Correctness
Strategy

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

$$\vdash_{\text{par}} \{ \phi \} P \{ \psi \}$$
Strategy

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

\[ \vdash \text{par} \{ \phi \} P \{ \psi \} \]

where

\[ \models \text{par} \{ \phi \} P \{ \psi \} \text{ holds whenever } \vdash \text{par} \{ \phi \} P \{ \psi \} \]

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

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

where

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

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

\[
\begin{align*}
\{[[x \rightarrow E] \psi \} & \quad x = E \quad \{\psi\} \\
\text{[Assignment]} & 
\end{align*}
\]
Rules for Partial Correctness (continued)

\[
\begin{align*}
\{ \phi \land B \} & \quad C_1 \quad \{ \psi \} \\
\{ \phi \land \neg B \} & \quad C_2 \quad \{ \psi \}
\end{align*}
\]

\[\text{[If-statement]}\]

\[
\{ \phi \}\quad \text{if}\quad B\quad \{ \ C_1 \} \quad \text{else}\quad \{ \ C_2 \} \quad \{ \psi \}
\]
Rules for Partial Correctness (continued)

\[
\begin{align*}
\{\phi \land B\} & \quad C_1 \quad \{\psi\} \\
\{\phi \land \neg B\} & \quad C_2 \quad \{\psi\}
\end{align*}
\]

\[\text{[If-statement]}\]
\[\{\phi\} \text{ if } B \{ C_1 \} \text{ else } \{ C_2 \} \{\psi\}\]

\[
\begin{align*}
\{\psi \land B\} & \quad C \quad \{\psi\}
\end{align*}
\]

\[\text{[Partial-while]}\]
\[\{\psi\} \text{ while } B \{ C \} \{\psi \land \neg B\}\]
Rules for Partial Correctness (continued)

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

\[\frac{}{\{\phi'\} \ C \ \{\psi'\}}[^{\text{Consequence}}]\]
Proof Tableaux

Proofs have tree shape

All rules have the structure

\[
\text{something} \\
\hline
\text{something else}
\]

As a result, all proofs can be written as a tree.

Practical concern

These trees tend to be very wide when written out on paper. Thus we are using a linear format, called \textit{proof tableaux}.
Interleave Formulas with Code

\[
\{\phi\} \ C_1 \ \{\eta\} \quad \{\eta\} \ C_2 \ \{\psi\} \\
\hline
\{\phi\} \ C_1; \ C_2 \ \{\psi\} \\
\text{[Composition]}
\]

Shape of rule suggests format for proof of \(C_1; C_2; \ldots; C_n:\)

\[
\{\phi_0\} \\
C_1; \\
\{\phi_1\} \quad \text{justification} \\
C_2; \\
\vdots \\
\{\phi_{n-1}\} \quad \text{justification} \\
C_n; \\
\{\phi_n\} \quad \text{justification}
\]
Overall goal

Find a proof that at the end of executing a program $P$, some condition $\psi$ holds.
Working Backwards

Overall goal
Find a proof that at the end of executing a program $P$, some condition $\psi$ holds.

Common situation
If $P$ has the shape $C_1; \ldots; C_n$, we need to find the weakest formula $\psi'$ such that

$$\{ \psi' \} \ C_n \ \{ \psi \}$$
## Working Backwards

### Overall goal

Find a proof that at the end of executing a program $P$, some condition $\psi$ holds.

### Common situation

If $P$ has the shape $C_1; \ldots; C_n$, we need to find the weakest formula $\psi'$ such that

$$\{\psi'\} C_n \{\psi\}$$

### Terminology

The weakest formula $\psi'$ is called *weakest precondition*. 

Example

\[ \{ y < 3 \} \]
\[ \{ y + 1 < 4 \} \quad \text{Implied} \]
y = y + 1;
\[ \{ y < 4 \} \quad \text{Assignment} \]
Another Example

Can we claim $u = x + y$ after $z = x; z = z + y; u = z; \ ?$
Another Example

Can we claim \( u = x + y \) after \( z = x; \ z = z + y; \ u = z; \)?

\[
\begin{align*}
\{ \top \} \\
\{ x + y = x + y \} & \quad \text{Implied} \\
z = x; \\
\{ z + y = x + y \} & \quad \text{Assignment} \\
z = z + y; \\
\{ z = x + y \} & \quad \text{Assignment} \\
u = z; \\
\{ u = x + y \} & \quad \text{Assignment}
\end{align*}
\]
An Alternative Rule for If

We have:

\[
\{ \phi \land B \} \quad C_1 \quad \{ \psi \}
\]
\[
\{ \phi \land \neg B \} \quad C_2 \quad \{ \psi \}
\]

\[
\frac{\{ \phi \} \quad \text{if} \quad B \quad \{ C_1 \} \quad \text{else} \quad \{ C_2 \} \quad \{ \psi \}}{\{ (B \rightarrow \phi_1) \land (\neg B \rightarrow \phi_2) \} \quad \text{if} \quad B \quad \{ C_1 \} \quad \text{else} \quad \{ C_2 \} \quad \{ \psi \}}
\]

Sometimes, the following derived rule is more suitable:
Consider this implementation of \texttt{Succ}:

\begin{verbatim}
  a = x + 1;
  if (a = 1 == 0) {
    y = 1;
  } else {
    y = a;
  }
\end{verbatim}

Can we prove \{\top\} \texttt{Succ} \{y = x + 1\}?
Another Example

```plaintext
:
if ( a − 1 == 0 ) {
    {1 = x + 1} If-Statement 2
    y = 1;
    {y = x + 1} Assignment
}
else {
    {a = x + 1} If-Statement 2
    y = a;
    {y = x + 1} Assignment
}
{y = x + 1} If-Statement 2
```
Another Example

\[
\{\top\}
\{(x + 1 - 1 = 0 \rightarrow 1 = x + 1) \land
(\neg(x + 1 - 1 = 0) \rightarrow x + 1 = x + 1)\}\]

Implied

\[
a = x + 1;
\{(a - 1 = 0 \rightarrow 1 = x + 1) \land
(\neg(a - 1 = 0) \rightarrow a = x + 1)\}\]

Assignment

if ( a - 1 == 0 ) {  
\{1 = x + 1\}
    y = 1;
\{y = x + 1\}
} else {  
\{a = x + 1\}
    y = a;
\{y = x + 1\}
}
Recall: Partial-while Rule

\[
\{\psi \land B\} \ C \ \{\psi\}
\]

\[
\{\psi\} \ \text{while} \ B \ \{\mathcal{C}\} \ \{\psi \land \neg B\}
\]
We shall show that the following Core program \texttt{Fac1} meets this specification:

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

Thus, to show:

\[
\{\top\} \texttt{Fac1} \{y = x!\}
\]
Partial Correctness of Fac1

: 
{ \( y = z! \) }
while ( \( z \neq x \) ) {
  { \( y = z! \land z \neq x \) } \hspace{1cm} \text{Invariant}
  { \( y \cdot (z + 1) = (z + 1)! \) } \hspace{1cm} \text{Implied}
  \( z = z + 1; \)
  { \( y \cdot z = z! \) } \hspace{1cm} \text{Assignment}
  \( y = y \ast z; \)
  { \( y = z! \) } \hspace{1cm} \text{Assignment}
}
{ \( y = z! \land \neg(z \neq x) \) } \hspace{1cm} \text{Partial-while}
{ \( y = x! \) } \hspace{1cm} \text{Implied}
Partial Correctness of \texttt{Fac1}

\[
\begin{align*}
\{ \top \} & \quad \text{Implied} \\
\{(1 = 0!)\} & \\
y = 1; & \\
\{y = 0!\} & \quad \text{Assignment} \\
z = 0; & \\
\{y = z!\} & \quad \text{Assignment} \\
\text{while ( } z \neq x \text{ ) } & \{ \\
& : \\
& } & \\
\{y = z! \land \neg(z \neq x)\} & \quad \text{Partial-while} \\
\{y = x!\} & \quad \text{Implied}
\end{align*}
\]
Next Week

Lecture 11: Total Correctness; Semantics of Hoare Logic