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

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 ϕ_R.

Framework for Software Verification

Convert informal description R of *requirements* for an application domain into formula ϕ_R.

Write program P that meets ϕ_R.

Framework for Software Verification

Convert informal description R of *requirements* for an application domain into formula ϕ_R.

Write program P that meets ϕ_R.

Prove that P satisfies ϕ_R.

Framework for Software Verification

Convert informal description R of *requirements* for an application domain into formula ϕ_R.

Write program P that meets ϕ_R.

Prove that P satisfies ϕ_R.

Each step provides risks and opportunities.

Motivation of Core Language

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

Motivation of Core Language

- 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:

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

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:

$$B ::= (E <= E) \mid (!B) \mid (B \| \| B)$$

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:

$$B ::= (E <= E) \mid (!B) \mid (B \| \| B)$$

What about other kinds of boolean expressions (*e.g.*, conjunction)?

Commands in Language

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

$$C ::= \texttt{skip} \mid x = E \mid C; C \mid \texttt{if}\,(B)\,\{C\}\,\texttt{else}\,\{C\} \mid \texttt{while}\,(B)\,\{C\}$$

Example

Consider the factorial function:

$$\begin{aligned} 0! &\stackrel{\text{def}}{=} 1 \\ (n+1)! &\stackrel{\text{def}}{=} (n+1) \cdot n! \end{aligned}$$

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

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

Example

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

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

```
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

```
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 ϕ, then the state resulting from P's execution will satisfy ψ.

(Slightly Trivial) Example

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

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

(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
```

(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

$$\{x > 0\}\ \texttt{y = 0}\ \{y \cdot y < x\}$$

(Slightly Less Trivial) Example

Same assertion

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

Another example for P

```
y = 0;
while (y * y < x) {
    y = y + 1;
}
y = y - 1;
```

Hoare Triples

Definition

An assertion of the form $\{\phi\}\ P\ \{\psi\}$ is called a Hoare triple.

- ϕ is called the precondition, ψ is called the postcondition.
- A state of a Core program P is a function ρ that assigns each variable x in P to an integer $I(x)$.
- A state ρ satisfies ϕ if $\rho \Vdash \phi$—that is, we have a modal logic where the truth of ϕ depends on the current state.

Example

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 ϕ, the state resulting from *P*'s execution satisfies ψ, 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 ϕ, the state resulting from *P*'s execution satisfies ψ, provided that *P* terminates.

Notation

We write $\models_{\mathrm{par}} \{\phi\}\ P\ \{\psi\}$.

Extreme Example

$\{\phi\}$ while true { x = 0; } $\{\psi\}$

holds for all ϕ and ψ.

Total Correctness

Definition

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

Notation

We write $\models_{\mathrm{tot}} \{\phi\}\ P\ \{\psi\}$.

Back to Factorial

Consider Fac1:

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

Back to Factorial

Consider Fac1:

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

- $\models_{\text{tot}} \{x \geq 0\}$ Fac1 $\{y = x!\}$

Back to Factorial

Consider `Fac1`:

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

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

Back to Factorial

Consider Fac1:

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

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

Back to Factorial

Consider `Fac1`:

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

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

Strategy

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

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

Strategy

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

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

where

- $\models_{\text{par}} \{\phi\}\ P\ \{\psi\}$ holds whenever $\vdash_{\text{par}} \{\phi\}\ P\ \{\psi\}$ (correctness)

Strategy

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

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

where

- $\models_{\text{par}} \{\phi\}\ P\ \{\psi\}$ 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

$$\frac{\{\phi\}\ C_1\ \{\eta\} \qquad \{\eta\}\ C_2\ \{\psi\}}{\{\phi\}\ C_1; C_2\ \{\psi\}}\ [\text{Composition}]$$

Rules for Partial Correctness (continued)

$$\frac{}{\{[x \to E]\psi\}\ x = E\ \{\psi\}}\ [\text{Assignment}]$$

Rules for Partial Correctness (continued)

$$\frac{\{\phi \wedge B\}\ C_1\ \{\psi\} \qquad \{\phi \wedge \neg B\}\ C_2\ \{\psi\}}{\{\phi\}\ \texttt{if}\ B\ \{\ C_1\ \}\ \texttt{else}\ \{\ C_2\ \}\ \{\psi\}}\ \text{[If-statement]}$$

Rules for Partial Correctness (continued)

$$\frac{\{\phi \wedge B\}\ C_1\ \{\psi\} \qquad \{\phi \wedge \neg B\}\ C_2\ \{\psi\}}{\{\phi\}\ \texttt{if}\ B\ \{\ C_1\ \}\ \texttt{else}\ \{\ C_2\ \}\ \{\psi\}}\ \text{[If-statement]}$$

$$\frac{\{\psi \wedge B\}\ C\ \{\psi\}}{\{\psi\}\ \texttt{while}\ B\ \{\ C\ \}\ \{\psi \wedge \neg B\}}\ \text{[Partial-while]}$$

Rules for Partial Correctness (continued)

$$\frac{\vdash_{AR} \phi' \rightarrow \phi \qquad \{\phi\}\ C\ \{\psi\} \qquad \vdash_{AR} \psi \rightarrow \psi'}{\{\phi'\}\ C\ \{\psi'\}}\ [\text{Consequence}]$$

Proof Tableaux

Proofs have tree shape

All rules have the structure

$$\frac{\text{something}}{\text{something else}}$$

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 *proof tableaux*.

Interleave Formulas with Code

$$\frac{\{\phi\}\ C_1\ \{\eta\} \qquad \{\eta\}\ C_2\ \{\psi\}}{\{\phi\}\ C_1; C_2\ \{\psi\}}\text{[Composition]}$$

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

$\{\phi_0\}$
$C_1;$
$\{\phi_1\}$ justification
$C_2;$
$\vdots$
$\{\phi_{n-1}\}$ justification
$C_n;$
$\{\phi_n\}$ justification

Working Backwards

Overall goal

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

Working Backwards

Overall goal

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

Common situation

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

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

Working Backwards

Overall goal

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

Common situation

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

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

Terminology

The weakest formula ψ' is called *weakest precondition*.

Example

$\{y < 3\}$	
$\{y + 1 < 4\}$	Implied
y = y + 1;	
$\{y < 4\}$	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;` ?

$\{\top\}$	
$\{x + y = x + y\}$	Implied
`z = x;`	
$\{z + y = x + y\}$	Assignment
`z = z + y;`	
$\{z = x + y\}$	Assignment
`u = z;`	
$\{u = x + y\}$	Assignment

An Alternative Rule for If

We have:

$$\frac{\{\phi \land B\}\ C_1\ \{\psi\} \qquad \{\phi \land \neg B\}\ C_2\ \{\psi\}}{\{\phi\}\ \texttt{if}\ B\ \{\ C_1\ \}\ \texttt{else}\ \{\ C_2\ \}\ \{\psi\}}\ [\text{If-statement}]$$

Sometimes, the following *derived rule* is more suitable:

$$\frac{\{\phi_1\}\ C_1\ \{\psi\} \qquad \{\phi_2\}\ C_2\ \{\psi\}}{\{(B \to \phi_1) \land (\neg B \to \phi_2)\}\ \texttt{if}\ B\ \{\ C_1\ \}\ \texttt{else}\ \{\ C_2\ \}\ \{\psi\}}\ [\text{If-stmt 2}]$$

Example

Consider this implementation of Succ:

```
a = x + 1;
if (a - 1 == 0) {
    y = 1;
} else {
    y = a;
}
```

Can we prove $\{\top\}$ Succ $\{y = x + 1\}$?

Another Example

$$\vdots$$

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

$\{\top\}$	
$\{(x+1-1=0 \rightarrow 1=x+1) \wedge$	
$(\neg(x+1-1=0) \rightarrow x+1=x+1)\}$	Implied
a = x + 1;	
$\{(a-1=0 \rightarrow 1=x+1) \wedge$	
$(\neg(a-1=0) \rightarrow a=x+1)\}$	Assignment
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

Recall: Partial-while Rule

$$\frac{\{\psi \land B\}\ C\ \{\psi\}}{\{\psi\}\ \texttt{while}\ B\ \{\ C\ \}\ \{\psi \land \neg B\}}\ \text{[Partial-while]}$$

Factorial Example

We shall show that the following Core program `Fac1` meets this specification:

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

Thus, to show:

$$\{\top\}\ \texttt{Fac1}\ \{y = x!\}$$

Partial Correctness of `Fac1`

$\vdots$

$\{y = z!\}$	
`while ( z != x ) {`	
$\quad\{y = z! \land z \neq x\}$	Invariant
$\quad\{y \cdot (z+1) = (z+1)!\}$	Implied
`z = z + 1;`	
$\quad\{y \cdot z = z!\}$	Assignment
`y = y * z;`	
$\quad\{y = z!\}$	Assignment
`}`	
$\{y = z! \land \neg(z \neq x)\}$	Partial-while
$\{y = x!\}$	Implied

Partial Correctness of Fac1

$\{\top\}$	
$\{(1 = 0!)\}$	Implied
`y = 1;`	
$\{y = 0!\}$	Assignment
`z = 0;`	
$\{y = z!\}$	Assignment
`while ( z != x ) {`	
$\vdots$	
`}`	
$\{y = z! \land \neg(z \neq x)\}$	Partial-while
$\{y = x!\}$	Implied

Next Week

- Lecture 11: Total Correctness; Semantics of Hoare Logic