11—Program Verification (Part II)

CS 3234: Logic and Formal Systems

Martin Henz and Aquinas Hobor

October 28, 2010

Generated on Thursday 28th October, 2010, 11:44

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.

Expressions in Core Language

Expressions come as arithmetic expressions E:

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

Expressions in Core Language

Expressions come as arithmetic expressions E:

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

and boolean expressions B:

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

Commands in Core Language

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

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

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 Core program, we have $y = x!$.

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

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.

Example

What program P meets this tiple

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

One correct answer: $P =$

```
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 l that assigns each variable x in P to an integer $l(x)$.
- A state l satisfies ϕ if $\mathcal{M} \models_l \phi$, where $\mathcal{M}$ contains integers and gives the usual meaning to the arithmetic operations.
- Quantifiers in ϕ and ψ bind only variables that do *not* occur in the program P.

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

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.

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* $\{\psi\}$.

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

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

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

Rules for Partial Correctness (continued)

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

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

Rules for Partial Correctness (continued)

$$\frac{\vdash_{\mathrm{AR}} \phi' \to \phi \qquad \{\phi\}\ C\ \{\psi\} \qquad \vdash_{\mathrm{AR}} \psi \to \psi'}{\{\phi'\}\ C\ \{\psi'\}}\ [\text{Implied}]$$

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

How To Discover an Invariant?

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

How To Discover an Invariant?

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

To be proven: $\{\phi\}\ \texttt{while}\ B\ \{\ C\ \}\ \{\psi\}$

How To Discover an Invariant?

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

To be proven: $\{\phi\}\ \texttt{while}\ B\ \{\ C\ \}\ \{\psi\}$

1. $\vdash_{\mathrm{AR}} \phi \rightarrow \eta$

How To Discover an Invariant?

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

To be proven: $\{\phi\}\ \texttt{while}\ B\ \{\ C\ \}\ \{\psi\}$

1. $\vdash_{\text{AR}} \phi \rightarrow \eta$
2. $\vdash_{\text{AR}} \eta \wedge \neg B \rightarrow \psi$

How To Discover an Invariant?

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

To be proven: $\{\phi\}\ \texttt{while}\ B\ \{\ C\ \}\ \{\psi\}$

1. $\vdash_{\mathrm{AR}} \phi \rightarrow \eta$
2. $\vdash_{\mathrm{AR}} \eta \wedge \neg B \rightarrow \psi$
3. $\{\eta \wedge B\}\ C\ \{\eta\}$

Partial Correctness of `Fac1`

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

1. Review: Partial Correctness
2. Proof Calculus for Total Correctness
3. Programming by Contract

Ideas for Total Correctness

- The only source of non-termination is the `while` command.
- If we can show that the value of an integer expression decreases in each iteration, but never becomes negative, we have proven termination.

Ideas for Total Correctness

- The only source of non-termination is the `while` command.
- If we can show that the value of an integer expression decreases in each iteration, but never becomes negative, we have proven termination.
  Why?

Ideas for Total Correctness

- The only source of non-termination is the `while` command.
- If we can show that the value of an integer expression decreases in each iteration, but never becomes negative, we have proven termination.
  Why? Well-foundedness of natural numbers

Ideas for Total Correctness

- The only source of non-termination is the `while` command.
- If we can show that the value of an integer expression decreases in each iteration, but never becomes negative, we have proven termination.
  Why? Well-foundedness of natural numbers
- We shall include this argument in a new version of the `while` rule.

Rules for Partial Correctness (continued)

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

$$\frac{\{\psi \land B \land 0 \leq E = E_0\}\ C\ \{\psi \land 0 \leq E < E_0\}}{\{\psi \land 0 \leq E\}\ \texttt{while}\ B\ \{\ C\ \}\ \{\psi \land \neg B\}}\ \text{[Total-while]}$$

Factorial Example (Again!)

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

What could be a good variant E?

Factorial Example (Again!)

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

What could be a good variant E?

E must strictly decrease in the loop, but not become negative.

Factorial Example (Again!)

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

What could be a good variant E?

E must strictly decrease in the loop, but not become negative.

Answer:

$$x - z$$

Total Correctness of Fac1

$\vdots$

$\{y = z! \land 0 \leq x - z\}$

while (z != x) {

$\{y = z! \land z \neq x \land 0 \leq x - z = E_0\}$	Invariant
$\{y \cdot (z+1) = (z+1)! \land 0 \leq x - (z+1) < E_0\}$	Implied
z = z + 1;	
$\{y \cdot z = z! \land 0 \leq x - z < E_0\}$	Assignment
y = y $*$ z;	
$\{y = z! \land 0 \leq x - z < E_0\}$	Assignment

}

$\{y = z! \land \neg(z \neq x)\}$	Total-while
$\{y = x!\}$	Implied

Total Correctness of Fac1

$\{x \leq 0\}$	
$\{(1 = 0! \wedge 0 \leq x - 0\}$	Implied
`y = 1;`	
$\{y = 0! \wedge 0 \leq x - 0\}$	Assignment
`z = 0;`	
$\{y = z! \wedge 0 \leq x - z\}$	Assignment
`while ( z != x ) {`	
$\vdots$	
$\}$	
$\{y = z! \wedge \neg(z \neq x)\}$	Total-while
$\{y = x!\}$	Implied

Programming by Contract

Consider

$$\{\phi\}\ P\ \{\psi\}$$

Obligation for consumer of *P*

Only run *P* when ϕ is met.

Obligation for producer of *P*

Make sure ψ is met after every run of *P*, assuming that ϕ is met before the run.

Contracts as Documentation

```
int factorial (x: int) { ... return y; }
```

Method name:	`factorial`
Input:	`x of type int`
Assumes:	$0 \leq x$
Guarantees:	$y = x!$
Output:	y
Modifies only:	y