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

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 ϕ_R.
Write	program P that meets ϕ_R.
Prove	that P satisfies ϕ_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
- 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:

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

and boolean expressions B:

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

Where are the other comparisons, for example ==?

Commands in Core Language

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

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

Example

Consider the factorial function:

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

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

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

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

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

$$(\!|\phi|\!)\ P\ (\!|\psi|\!)$$

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.

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;
```

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 $f \in F$ 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 $\mathcal{M}$ satisfies ϕ with respect to environment l, written $\mathcal{M} \models_l \phi$:

- in case ϕ 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 $P^{\mathcal{M}}$;
- in case ϕ has the form $\forall x \psi$, if the $\mathcal{M} \models_{l[x \mapsto a]} \psi$ holds for all $a \in A$;
- in case ϕ has the form $\exists x \psi$, if the $\mathcal{M} \models_{l[x \mapsto a]} \psi$ holds for some $a \in A$;

Recall: Satisfaction Relation (continued)

- in case ϕ has the form $\neg\psi$, if $\mathcal{M} \models_I \psi$ does not hold;
- in case ϕ has the form $\psi_1 \vee \psi_2$, if $\mathcal{M} \models_I \psi_1$ holds or $\mathcal{M} \models_I \psi_2$ holds;
- in case ϕ has the form $\psi_1 \wedge \psi_2$, if $\mathcal{M} \models_I \psi_1$ holds and $\mathcal{M} \models_I \psi_2$ holds; and
- in case ϕ has the form $\psi_1 \rightarrow \psi_2$, if $\mathcal{M} \models_I \psi_1$ holds whenever $\mathcal{M} \models_I \psi_2$ holds.

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.

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

Notation

We write $\models_{\text{par}} (\!|\phi|\!)$ *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 $\models_{\text{tot}} (\!|\phi|\!)$ *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; }
```

- $\models_{\mathrm{tot}} (\!|x \geq 0|\!)$ Fac1 $(\!|y = x!|\!)$
- $\not\models_{\mathrm{tot}} (\!|\top|\!)$ Fac1 $(\!|y = x!|\!)$
- $\models_{\mathrm{par}} (\!|x \geq 0|\!)$ Fac1 $(\!|y = x!|\!)$

Back to Factorial

Consider `Fac1`:

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

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

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

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)

$$\frac{}{(\!|[x \rightarrow E]\psi|\!)\ x = E\ (\!|\psi|\!)}\ [\text{Assignment}]$$

Examples

Let P be the program x = 2.
Using

$$\frac{}{(\![x \to E]\psi]\!)\ x = E\ (\!|\psi|\!)}\ [\text{Assignment}]$$

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

$$\frac{}{(\!|[x \rightarrow E]\psi|\!)\ x = E\ (\!|\psi|\!)}\,[\text{Assignment}]$$

we can prove:

- $(\!|x + 1 = 2|\!)\ P\ (\!|x = 2|\!)$
- $(\!|x + 1 = y|\!)\ P\ (\!|x = y|\!)$

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)

$$\frac{\vdash_{AR} \phi' \to \phi \qquad (\!|\phi|\!)\ C\ (\!|\psi|\!) \qquad \vdash_{AR} \psi \to \psi'}{(\!|\phi'|\!)\ C\ (\!|\psi'|\!)}\,[\text{Implied}]$$

Next Week

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