Part II. Hoare Logic and Program Verification

Dilian Gurov

Specs: safety of data manipulation
Models: source code
Specs: logic assertions
Method: Hoare logic, VCG
Tool: VeriFast

Specification and Verification

• Program specification
  - states program correctness, formally
  - relates properties of states
    before and after the execution
• Program verification
  - proves program correctness, formally
  - relative to a specification

Why specify programs?

• Good for documentation: capture unambiguously what the program should do (and not how)
• Programs annotated with specs can be fed into static checkers
• However, specifications:
  - require expertise and time
  - can get large and difficult to handle

Why verify programs?

• Testing can only find errors, but cannot prove their absence
• However, verification is expensive:
  - requires formal specs
  - requires expertise and time
  - faces decidability and complexity issues
• Therefore:
  - use light-weight tools for critical parts

Code Verification

• Code verification
  - the code itself is the model!
  - i.e., no abstraction...
  - ...but must still be based on a formal description of execution: a formal semantics of the programming language
Semantics of Programming Languages

• Semantics
  - a formal definition of how programs execute
  - can be given in various ways
  - see course DD2457

Semantics of Programming Languages

• Natural semantics
  - relates states before and after the execution
  - is defined inductively on the program structure
  - here: informal understanding

A Core Programming Language

• A simple program:
  ```
  y = 1;
  z = 0;
  while (z != x) {
    z = z + 1;
    y = y * z;
  }
  ```

Formal Syntax

• Arithmetic expressions
  ```
  E ::= n | x | (E + E) | (E - E) | (E * E)
  ```

• Boolean expressions
  ```
  B ::= true | false | (E < E) |
      !B | (B & B) | (B || B)
  ```

• Commands
  ```
  C ::= x = E | C ; C | if B { C } else { C } |
       while B { C }
  ```

States and Configurations

• State (in this context)
  - captures the values of the variables in the program
  - formally, a mapping $\text{Var} \rightarrow \text{Int}$

• Configuration
  - describes where we are in the execution
  - consists of a control point and a state

State Properties

• State properties
  - can be expressed as logic formulas in predicate calculus (over program variables)
  - are called assertions
    - e.g., $x > y$ eller $\exists z (x = 2 * z)$
  - We associate assertions with control points in the program
    - could be seen as control point properties
Program Specification

- Can be accomplished with two assertions:
  - precondition
  - postcondition
- Formal notation: Hoare triples
  \[ \langle \phi \rangle P \langle \psi \rangle \]
  read (roughly): if execution of program \( P \) starts in a state where precondition \( \phi \) holds, then the execution ends in a state where postcondition \( \psi \) holds.

Specification Example

- The factorial program \( \text{Fac}1 \)
  
  ```
  y = 1;
  z = 0;
  while (z != x) {
    z = z + 1;
    y = y * z;
  }
  ```

  can be specified with the Hoare triple
  \[ \langle x \geq 0 \rangle \text{Fac}1 \langle y = x! \rangle \]

Partial Correctness

\[ |=_{\text{par}} \langle \phi \rangle P \langle \psi \rangle \]
holds if:

- if the execution of \( P \) starts in a state where precondition \( \phi \) holds,
- and the execution terminates,
- then postcondition \( \psi \) holds in the final state.

Total Correctness

\[ |=_{\text{tot}} \langle \phi \rangle P \langle \psi \rangle \]
holds if:

- if the execution of \( P \) starts in a state where precondition \( \phi \) holds,
- then the execution terminates,
- and postcondition \( \psi \) holds in the final state.

Example

- When does \( |=_{\text{par}} \langle \text{true} \rangle P \langle \text{false} \rangle \) hold?

  when execution of \( P \) does not terminate, regardless of the start state.

Example

- When does \( |=_{\text{tot}} \langle \text{true} \rangle P \langle \text{false} \rangle \) hold?

  never!
Logic Variables

• Can the factorial program \texttt{Fac2}
  
  \begin{verbatim}
  y = 1;
  while (x != 0) {
    y = y * x;
    x = x - 1;
  }
  \end{verbatim}

  be specified with the Hoare tripple

  \[ \langle x \geq 0 \rangle \texttt{Fac2} \langle y = x! \rangle \]

• We need additional variables, so-called logic variables, to capture how the final values of the variables relate to the initial values.

• These variables are considered universally quantified in a Hoare tripple.

Logic Variables

• The factorial program \texttt{Fac2}
  
  \begin{verbatim}
  y = 1;
  while (x != 0) {
    y = y * x;
    x = x - 1;
  }
  \end{verbatim}

  can be specified with the Hoare tripple

  \[ \langle x \geq 0 \land x = x_0 \rangle \texttt{Fac2} \langle y = x_0! \rangle \]

Program Specification

• It should be clear from the specification how the program can be used - \textit{without} knowing the code itself!

  \[ \langle x \geq 0 \land x = x_0 \rangle \texttt{Fac2} \langle y = x_0! \rangle \]

  \( x \geq 0 \) then the program terminates

  \( x = x_0 \) binds the start value of \( x \)

  \( y = x_0! \) relates the final value of \( y \) to the initial value of \( x \)

Hoare Logic

• Consists of a set of rules
  - for reasoning over Hoare tripples
  - to verify programs (partial correctness)

• Proofs
  - in the form of proof trees
  - or so-called "tableaux"
  - reduce the validity of Hoare tripples to the validity of predicate logic formulas over arithmetic

Assignment Rule

\[ \langle \psi[E/x] \rangle x = E \langle \psi \rangle \]

• "propagates" the postcondition backwards
**Implied Rules**

\[
\begin{align*}
&\vdash \phi' \rightarrow \phi \quad \langle \phi \rangle C \langle \psi \rangle \\
&\Rightarrow \quad \langle \phi' \rangle C \langle \psi \rangle \\
&\langle \phi \rangle C \langle \psi \rangle \quad \vdash \psi \rightarrow \psi' \\
&\Rightarrow \quad \langle \phi \rangle C \langle \psi' \rangle 
\end{align*}
\]

**Proof Example**

\[
\begin{align*}
&\vdash x > 0 \rightarrow x + 1 > 0 \\
&\Leftrightarrow \quad \langle x + 1 > 0 \rangle x = x + 1 \langle x > 0 \rangle \\
&\Leftrightarrow \quad \langle x > 0 \rangle x = x + 1 \langle x > 0 \rangle \\
\end{align*}
\]

- One proof obligation: \( \vdash x > 0 \rightarrow x + 1 > 0 \)

  to be proved in an "external" proof system

**Sequential Composition Rule**

\[
\langle \phi \rangle C_1 \langle \eta \rangle \quad \langle \eta \rangle C_2 \langle \psi \rangle \\
\Rightarrow \quad \langle \phi \rangle C_1 ; C_2 \langle \psi \rangle \\
\]

- introduce an intermediate assertion \( \eta \) in the control point preceding \( C_2 \)

**Example**

- What does this program do
  \[
  z = x; \\
x = y; \\
y = z;
\]
  and how can this be specified?

**The Proof Tree**

- Too big to be shown here...
  - ...show on whiteboard instead

**Example**

- Program \( \text{Swap} \)
  \[
  z = x; \\
x = y; \\
y = z;
\]
  can be specified with
  \[
  \langle x = x_0 \land y = y_0 \rangle \text{Swap} \langle x = y_0 \land y = x_0 \rangle
  \]
Proof Tableaux

• Alternative presentation:
  - as tableau
  - correctness proofs can be presented as commented (or annotated) programs, where the comments are assertions associated with control points

The Proof in Tableau Form

\[ \langle x = x_0 \land y = y_0 \rangle \text{ Precondition} \]
\[ \langle y = y_0 \land x = x_0 \rangle \text{ Implied ( )} \]
\[ z = x; \text{ Assignment} \]
\[ \langle y = y_0 \land z = x_0 \rangle \]
\[ x = y; \text{ Assignment} \]
\[ \langle x = y_0 \land z = x_0 \rangle \]
\[ y = z; \text{ Assignment} \]
\[ \langle x = y_0 \land y = x_0 \rangle \]

Tableaux

• A tableau is an annotated program
• An operational interpretation:
  if execution begins in a state where the first annotation (precondition) holds, then every time the execution reaches a control point, all assertions associated with this control point hold

Proofs in Tableau Form

• A proof in tableau form is a program annotated with at least one assertion at every control point, where the annotations match the rules (patterns)
• The proof process can be seen as completion of the initial annotation “inwards”
• Assertions associated with the same control point give rise to proof obligations

If Rule

\[ \frac{\langle \phi \land B \rangle C_1 \langle \psi \rangle \quad \langle \phi \land \neg B \rangle C_2 \langle \psi \rangle}{\langle \phi \rangle \text{ if } B \{ C_1 \} \text{ else } \{ C_2 \} \langle \psi \rangle} \]

Example

• What does the program

```java
if (x > 0) {
    y = x;
} else {
    y = -x;
}
```

and how can this be specified?
Example

• Program \textit{Abs}

\[
\text{if } (x > 0) \{
    y = x;
\} \text{ else }
\]
\[
    y = -x;
\]

can be specified with
\[
\langle x = x_0 \rangle \text{Abs } \langle y = |x_0| \rangle
\]

Proof Tableau

| \{x = x_0\} | \text{Precondition} |
| \{x > 0\} | \text{If} |
| \{y = x\} | \text{Implied ( )} |
| \{y = |x|\} | \text{Assignment} |

| \{x = x_0 \land \neg (x > 0)\} | \text{If} |
| \{y = -x\} | \text{Implied ( )} |
| \{y = |x|\} | \text{Assignment} |

\langle y = |x_0| \rangle \text{Postcondition}

Partial-while Rule

\[
\frac{(\eta \land B) C \langle \eta \rangle}{\langle \eta \rangle \text{while } B \{C\} \langle \eta \land \neg B \rangle}
\]

\[\text{loop invariant}\]

Example

• Proof

\[\langle x < 0 \rangle \text{ while } (x! = 0) \{ x = x - 1; \} \langle \text{false} \rangle\]

• Proof

\[\langle \text{true} \rangle \text{ while } (x! = 0) \{ x = x - 1; \} \langle x = 0 \rangle\]

Loop Invariants

• A loop invariant to

\[\text{while } B \{C\}\]

is an assertion \(\eta\) for which
\[|=_{\text{par}} \langle \eta \land B \rangle \{C\} \langle \eta \rangle\]
holds

• Big choice, e.g.: false and true

Loop Invariants

• To the Hoare tripple

\[\langle \phi \rangle \text{ while } B \{C\} \langle \psi \rangle\]

we need a loop invariant \(\eta\) such that:
\[|= \phi \rightarrow \eta\]
\[|= \eta \land \neg B \rightarrow \psi\]
\[|= \eta \rightarrow \psi \lor B\]
Loop Invariants

• A whole “assertion interval” to choose from:
  \[ \neg \phi \rightarrow \eta \rightarrow \psi \lor B \]

• Two immediate candidates:
  - \( \phi \)
  - \( \psi \lor B \)
  in case they are loop invariants!

Example

• Find suitable loop invariants for:
  - \( \langle x < 0 \rangle \) while \( (\neg|x|=0) \) \( \{x = x - 1;\} \) \( \langle \text{false} \rangle \)
  - \( \langle \text{true} \rangle \) while \( (\neg|x|=0) \) \( \{x = x - 1;\} \) \( \langle x = 0 \rangle \)

• One answer: the preconditions!

Proof Tableau 1

\[
\begin{align*}
\langle x < 0 \rangle & \quad \text{Precondition} \\
\text{while} (\neg|x|=0) \{ & \\
\langle x < 0 \land \neg (x \neq 0) \rangle & \quad \text{Partial-while} \\
\langle x - 1 < 0 \rangle & \quad \text{Implied ( )} \\
x &= x - 1; & \quad \text{Assignment} \\
\langle x < 0 \rangle & \quad \text{Partial-while} \\
\langle \text{false} \rangle & \quad \text{Implied ( )}
\end{align*}
\]

Example

• Verify the factorial program Fac1

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

specified with the Hoare triple

\( \langle x \geq 0 \land x = x_0 \rangle \) Fac1 \( \langle y = x_0! \rangle \)

Proof Tableau 2

\[
\begin{align*}
\langle \text{true} \rangle & \quad \text{Precondition} \\
\text{while} (\neg|x|=0) \{ & \\
\langle \text{true} \land \neg (x \neq 0) \rangle & \quad \text{Partial-while} \\
\langle \text{true} \rangle & \quad \text{Implied ( )} \\
x &= x - 1; & \quad \text{Assignment} \\
\langle \text{true} \rangle & \quad \text{Implied ( )}
\end{align*}
\]

Proof Tableau

\[
\begin{align*}
\langle x \geq 0 \land x = x_0 \rangle & \quad \text{Precondition} \\
(1 = 0 \land x = x_0 \geq 0) & \quad \text{Implied ( )} \\
y &= 1; & \quad \text{Assignment} \\
(\neg x \neq x_0 \lor x = 0) & \quad \text{Assignment} \\
x &= 0; & \quad \text{Assignment} \\
(z = 0 \land x = x_0 \land \neg z) & \quad \text{Partial-while} \\
\text{while} (z != x) \{ & \\
\quad (z + 1 = x) & \quad \text{Implied ( )} \\
\quad y = y \times z; & \quad \text{Assignment} \\
\quad y &= y \times z; \\
\quad (y \neq z) & \quad \text{Assignment} \\
\}
\end{align*}
\]

\( \langle y \neq z \rangle \)