Part II. Hoare Logic and Program Verification

Dilian Gurov

Part II. Hoare Logic and Program Verification

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

2

Specification and Verification

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

3

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

4

Why verify programs?

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

5

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

6

Semantics of Programming Languages

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

7

Semantics of Programming Languages

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

8

A Core Programming Language

- A simple program:

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

9

Formal Syntax

- Arithmetic expressions

  $E \ ::= \ n \ | \ x \ | \ (E+E) \ | \ (E-E) \ | \ (E*E)$
- Boolean expressions

  $B \ ::= \ \mathtt{true} \ | \ \mathtt{false} \ | \ (E<E) \ | \ (!B) \ | \ (B \,\&\, B) \ | \ (B \,||\, B)$
- Commands

  $C \ ::= \ x=E \ | \ C\,;C \ | \ \mathtt{if}\ B\ \{C\}\ \mathtt{else}\ \{C\} \ | \ \mathtt{while}\ B\ \{C\}$

10

States and Configurations

- State (in this context)
  - captures the values of the variables in the program
  - formally, a mapping $\mathrm{Var} \rightarrow \mathrm{Int}$
- Configuration
  - describes where we are in the execution
  - consists of a control point and a state

11

State Properties

- State properties
  - can be expressed as logic formulas in predicate calculus (over program variables)
  - are called assertions

    t.ex. $x > y$ eller $\exists z\,(x = 2 * z)$
- We associate assertions with control points in the program
  - could be seen as control point properties

12

Program Specification

- Can be accomplished with two assertions:
  - precondition
  - postcondition
- Formal notation: Hoare tripples

$$\langle\phi\rangle\ P\ \langle\psi\rangle$$

read (roughly): if execution of program P starts in a state where precondition ϕ holds, then the execution ends in a state where postcondition ψ holds

13

Specification Example

- The factorial program *Fac1*

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

can be specified with the Hoare tripple

$$\langle x \geq 0\rangle\ Fac1\ \langle y = x!\rangle$$

14

Partial Correctness

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

if the execution of P starts in a state where precondition ϕ holds,
and the execution terminates,
then postcondition ψ holds in the final state

15

Total Correctness

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

if the execution of P starts in a state where precondition ϕ holds,
then the execution terminates,
and postcondition ψ holds in the final state

16

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

17

Example

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

never!

18

Logic Variables

- Can the factorial program *Fac2*

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

be specified with the Hoare tripple

$$\langle x \geq 0 \rangle \; Fac2 \; \langle y = x! \rangle$$

Logic Variables

- 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 *Fac2*

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

can be specified with the Hoare tripple

$$\langle x \geq 0 \wedge x = x_0 \rangle \; Fac2 \; \langle y = x_0! \rangle$$

Program Specification

- It should be clear from the specification how the program can be used - *without* knowing the code itself!

$$\langle x \geq 0 \wedge x = x_0 \rangle \; Fac2 \; \langle y = x_0! \rangle$$

$x \geq 0$ 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

$$\frac{-}{\langle \psi[E/x] \rangle \; x = E \; \langle \psi \rangle}$$

- "propagates" the postcondition backwards

Implied Rules

$$\frac{\vdash \phi' \rightarrow \phi \qquad \langle\phi\rangle\, C\, \langle\psi\rangle}{\langle\phi'\rangle\, C\, \langle\psi\rangle}$$

$$\frac{\langle\phi\rangle\, C\, \langle\psi\rangle \qquad \vdash \psi \rightarrow \psi'}{\langle\phi\rangle\, C\, \langle\psi'\rangle}$$

Proof Example

$$\frac{\dfrac{\square}{\vdash x > 0 \rightarrow x + 1 > 0} \qquad \dfrac{-}{\langle x + 1 > 0\rangle\, x = x + 1\, \langle x > 0\rangle}}{\langle x > 0\rangle\, x = x + 1\, \langle x > 0\rangle}$$

- One proof obligation: $\vdash x > 0 \rightarrow x + 1 > 0$
  to be proved in an "external" proof system

Sequential Composition Rule

$$\frac{\langle\phi\rangle\, C_1\, \langle\eta\rangle \qquad \langle\eta\rangle\, C_2\, \langle\psi\rangle}{\langle\phi\rangle\, C_1 ; C_2\, \langle\psi\rangle}$$

- introduce an intermediate assertion η 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?

Example

- Program *Swap*

```
z = x;
x = y;
y = z;
```

can be specified with

$$\langle x = x_0 \wedge y = y_0\rangle\; Swap\; \langle x = y_0 \wedge y = x_0\rangle$$

The Proof Tree

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

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

31

The Proof in Tableau Form

$\langle x = x_0 \wedge y = y_0 \rangle$	Precondition
$\langle y = y_0 \wedge x = x_0 \rangle$	Implied (□)
`z = x;`	
$\langle y = y_0 \wedge z = x_0 \rangle$	Assignment
`x = y;`	
$\langle x = y_0 \wedge z = x_0 \rangle$	Assignment
`y = z;`	
$\langle x = y_0 \wedge y = x_0 \rangle$	Assignment

32

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

33

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

34

If Rule

$$\frac{\langle \phi \wedge B \rangle\ C_1\ \langle \psi \rangle \qquad \langle \phi \wedge \neg B \rangle\ C_2\ \langle \psi \rangle}{\langle \phi \rangle\ \mathbf{if}\ B\ \{C_1\}\ \mathbf{else}\ \{C_2\}\ \langle \psi \rangle}$$

35

Example

- What does the program

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

and how can this be specified?

36

Example

- Program *Abs*

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

can be specified with

$\langle x = x_0 \rangle \; Abs \; \langle y = |x_0| \rangle$

37

Proof Tableau

$\langle x = x_0 \rangle$	Precondition
`if (x > 0) {`	
$\langle x = x_0 \wedge x > 0 \rangle$	If
$\langle x = \mid x_0 \mid \rangle$	Implied (□)
`y = x;`	
$\langle y = \mid x_0 \mid \rangle$	Assignment
`} else {`	
$\langle x = x_0 \wedge \neg(x > 0) \rangle$	If
$\langle -x = \mid x_0 \mid \rangle$	Implied (□)
`y = -x;`	
$\langle y = \mid x_0 \mid \rangle$	Assignment
`}`	
$\langle y = \mid x_0 \mid \rangle$	Postcondition

38

Partial-while Rule

$$\frac{\langle \eta \wedge B \rangle \; C \; \langle \eta \rangle}{\langle \eta \rangle \; \texttt{while} \; B \; \{C\} \; \langle \eta \wedge \neg B \rangle}$$

loop invariant

39

Example

- Proof
  $\langle x < 0 \rangle$ `while (x!=0) {x = x - 1;}` $\langle$false$\rangle$

- Proof
  $\langle$true$\rangle$ `while (x!=0) {x = x - 1;}` $\langle x = 0 \rangle$

40

Loop Invariants

- A loop invariant to

  $\texttt{while} \; B \; \{C\}$

  is an assertion η for which

  $\models_{\mathrm{par}} \langle \eta \wedge B \rangle \; C \; \langle \eta \rangle$

  holds
- Big choice, e.g.: false and true

41

Loop Invariants

- To the Hoare tripple

  $\langle \phi \rangle \; \texttt{while} \; B \; \{C\} \; \langle \psi \rangle$

  we need a loop invariant η such that:

  $\vdash \phi \rightarrow \eta$ and

  $\vdash \eta \wedge \neg B \rightarrow \psi$ $\quad (\vdash \eta \rightarrow \psi \vee B)$

42

Loop Invariants

- A whole "assertion interval" to choose from:

$$\vdash \phi \rightarrow \eta \rightarrow \psi \vee B$$

- Two immediate candidates:
  - ϕ
  - $\psi \vee B$

  in case they are loop invariants!

Example

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

- One answer: the preconditions!

Proof Tableau 1

$\langle \mathbf{x < 0} \rangle$	Precondition
`while (x!=0) {`	
$\langle \mathbf{x < 0} \wedge x \neq 0 \rangle$	Partial-while
$\langle x - 1 < 0 \rangle$	Implied ($\Box$)
`x = x - 1;`	
$\langle \mathbf{x < 0} \rangle$	Assignment
`}`	
$\langle \mathbf{x < 0} \wedge \neg(x \neq 0) \rangle$	Partial-while
$\langle \text{false} \rangle$	Implied ($\Box$)

Proof Tableau 2

$\langle \textbf{true} \rangle$	Precondition
`while (x!=0) {`	
$\langle \textbf{true} \wedge x \neq 0 \rangle$	Partial-while
$\langle \text{true} \rangle$	Implied ($\Box$)
`x = x - 1;`	
$\langle \textbf{true} \rangle$	Assignment
`}`	
$\langle \textbf{true} \wedge \neg(x \neq 0) \rangle$	Partial-while
$\langle x = 0 \rangle$	Implied ($\Box$)

Example

- Verify the factorial program $Fac1$

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

specified with the Hoare tripple

$$\langle x \geq 0 \wedge x = x_0 \rangle \; Fac1 \; \langle y = x_0! \rangle$$

Proof Tableau

$\langle x \geq 0 \wedge x = x_0 \rangle$	Precondition
$\langle 1 = 0! \wedge x = x_0 \wedge 0 \geq 0 \rangle$	Implied ($\Box$)
`y = 1;`	
$\langle y = 0! \wedge x = x_0 \wedge 0 \geq 0 \rangle$	Assignment
`z = 0;`	
$\langle \mathbf{y = z! \wedge x = x_0 \wedge z \geq 0} \rangle$	Assignment
`while (z != x) {`	
$\langle \mathbf{y = z! \wedge x = x_0 \wedge z \geq 0} \wedge z \neq x \rangle$	Partial-while
$\langle y \cdot (z + 1) = (z + 1)! \wedge x = x_0 \wedge z+1 \geq 0 \rangle$	Implied ($\Box$)
`z = z + 1;`	
$\langle y \cdot z = z! \wedge x = x_0 \wedge z \geq 0 \rangle$	Assignment
`y = y * z;`	
$\langle \mathbf{y = z! \wedge x = x_0 \wedge z \geq 0} \rangle$	Assignment
`}`	
$\langle \mathbf{y = z! \wedge x = x_0 \wedge z \geq 0} \wedge \neg(z \neq x) \rangle$	Partial-while
$\langle y = x_0! \rangle$	Implied ($\Box$)