Logic of Programs

Robert Keller
14 February 2005

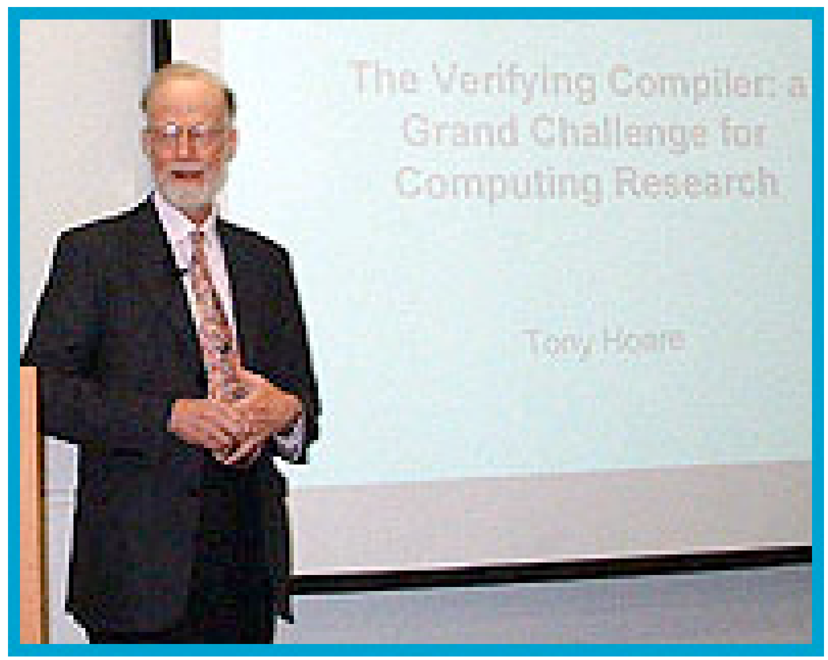
Programs with Proofs

- For many reasons, it is desirable to accompany programs with a **proof** that the program meets a certain specification.

- One way to do this is to derive the proof along with the program.

Hoare Logic

- C.A.R. (“Tony”) Hoare was the first to express program construction along with proofs of correctness as a single **unified logic**.



Sir Prof. Tony Hoare (FRS}
Microsoft Research Laboratory, Cambridge

Program "Dynamics"

- You may be accustomed to thinking of a program as something with "dynamic" behavior.

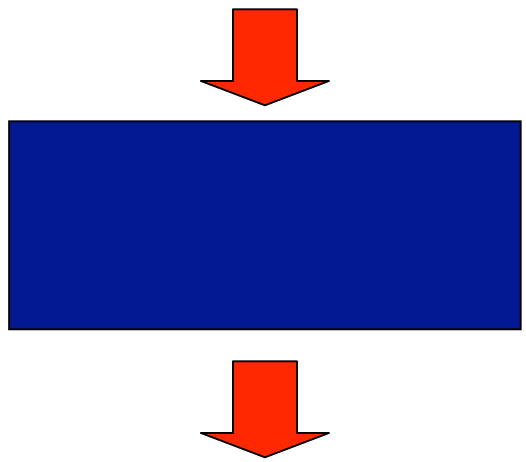
- The mathematical view is that a program's behavior is just one of many slices of a (generally-infinite) **static** structure, which can be analyzed with mathematics.

Programs States

- Programs work with **states**.
- Each state is a mapping from program variables into appropriate domains.
- A state is much like an assignment in our discussion of semantics of predicate calculus.

A Program is a State Transformer

starting state



ending state

Note about Files

- To deal with input streams and files, we will consider the entire file or stream, along with the current position of the reader or writer, to be part of the state.

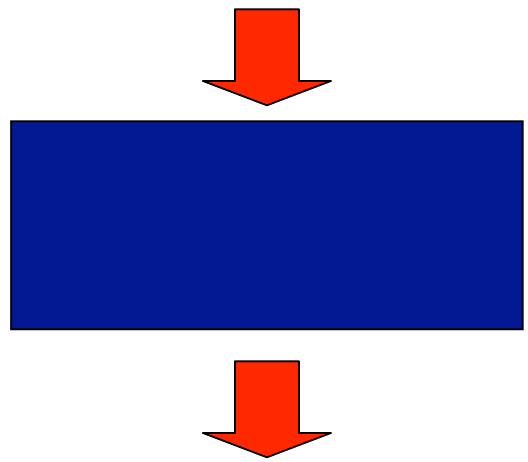
Programs with Assertions

- An **assertion** is a predicate-logic expression about the variables in the program.

- Assertions expression two kinds of things:
  - An **assumption** about the state.
  - An **expectation** about the state.

A **Program Specification** consists of

assumption about the starting state



expectation about the ending state

What ifs

- What if the assumption about the starting state doesn’t hold?
  - We don’t care about the result in this case.
  - However, the assumption can be made very stringent, e.g. identically TRUE, in which case we will always care.

What ifs

- What if the assumption about the starting state holds, but the expectation doesn’t hold when the program terminates?
  - The program is incorrect.

Logic Triples

- Consider endowing a program to be designed with its assumption and expectation:

{assumption} code {expectation}

- This is known as a “triple”.

Example of a Triple

{assumption} code {expectation}

$\{x \leq y \wedge x \leq z\}$ *TBD* $\{x \leq y \wedge y \leq z\}$

Design then becomes the process of filling in the TBD.

Some triples are more stringent than others

{assumption} code {expectation}

$\{x \leq y \wedge x \leq z\}$ *TBD* $\{x \leq y \wedge y \leq z\}$

$\{x \leq y\}$ *TBD* $\{x \leq y \wedge y \leq z\}$

$\{true\}$ *TBD* $\{x \leq y \wedge y \leq z\}$

$\{true\}$ *TBD* $\{x \leq y \wedge y \leq z \wedge z \leq w\}$

Advantages of Using Logic

- End up with a program that is proven to meet its specification (something that no amount of testing can do}.

- Certain aspects of programming can be automated (program synthesis}.

Composition of Triples

- Suppose we have a triple:

  {Assumption} Code {Expectation}
- To develop the code, we can break it into two parts:

  {Assumption 1} Code 1 {Expectation 1}

  {Assumption 2} Code 2 {Expectation 2}

  We want Code = Code 1; Code 2.

  What else do we need for this to work?

Composition Rule

$$\frac{\{A\}\ S1\ \{B\} \qquad \{B\}\ S2\ \{C\}}{\{A\}\ S1;S2\ \{C\}}$$

Example of Composition Rule

1. $\{true\}\ S1\ \{x \leq y\}$
2. $\{x \leq y\}\ S2\ \{x \leq y \wedge y \leq z\}$
3. $\{true\}\ S1;\ S2\ \{x \leq y \wedge y \leq z\}$
   Comp. 1, 2

What if Conditions don’t Match

- Sometimes we need to compose segments of code, but the expectation of the first doesn’t match the assumption of the second.

- The next best thing is for it the expectation to **imply** the assumption.

- We can express this by allowing the assumption to be strengthened or the expectation to be weakened, **without changing the code**.

Expectation Weakening Rule

$$\frac{\{\text{Assumption}\}\ \text{Code}\ \{\text{Expectation}\} \qquad \text{Expectation} \rightarrow \text{Expectation}'}{\{\text{Assumption}\}\ \text{Code}\ \{\text{Expectation}'\}}$$

Note: The weakest possible expectation is ______.

Assumption Strengthening Rule

{Assumption} Code {Expectation}

Assumption′ $\rightarrow$ Assumption   (logical implication)

{Assumption′} Code {Expectation}

Note: The strongest possible assumption is ______.

Example of Weakening/Strengthing

1. $\{\text{true}\}\ S1\ \{x < y\}$

2. $\{x \leq y\}\ S2\ \{x \leq y \wedge y \leq z\}$

3. To compose these we can either use the assumption strengthening rule to get:
   $\{x < y\}\ S2\ \{x \leq y \wedge y \leq z\}$ since $x < y \rightarrow x \leq y$

4. or we could use the expectation weakening rule to get:
   $\{\text{true}\}\ S1\ \{x \leq y\}$

Generalized Composition Rule

$$\frac{\{A\}\ S1\ \{B\} \qquad\qquad B \rightarrow C \qquad\qquad \{C\}\ S2\ \{D\}}{\{A\}\ S1;S2\ \{D\}}$$

This avoids the introduction of extra steps by the strengthening and weakening rules.

Conditional Rule

$$\frac{\{A \wedge P\}\ S1\ \{B\} \qquad \{A \wedge \neg P\}\ S2\ \{B\}}{\{A\}\ \ \textbf{if(}\ P\ \textbf{)}\ S1\ \textbf{else}\ S2\ \ \{B\}}$$

Example of Conditional Rule

1. $\{x \leq y \wedge (y > z)\}$ *S1* $\{x \leq y \wedge y \leq z\}$
2. $\{x \leq y \wedge \neg(y > z)\}$ *S2* $\{x \leq y \wedge y \leq z\}$
3. $\{x \leq y\}$

   if(y > z) S1 **else** S2

   $\{x \leq y \wedge y \leq z\}$ Cond. 1, 2

One-Sided Conditional Rule

$$\frac{\{A \wedge P\}\ S1\ \{B\} \qquad (A \wedge \neg P) \rightarrow B}{\{A\}\ \textbf{if(}\ P\ \textbf{)}\ S1\ \{B\}}$$

Example of One-Sided Conditional Rule

1. $\{x \leq y \wedge y > z\}$ *S1* $\{x \leq y \wedge y \leq z\}$

2. $((x \leq y) \wedge \neg(y > z)) \rightarrow (x \leq y \wedge y \leq z)$

3. $\{x \leq y\}$

   if(y > z) S1

   $\{x \leq y \wedge y \leq z\}$ One-SidedCond. 1, 2

While Rule

$$\frac{\{I \wedge P\}\ S\ \{I\}}{\{I\}\ \textbf{while(P)}\ S\ \{I \wedge \neg P\}}$$

I is known as the “loop invariant”

Example of While Rule

1. $\{x \leq y \wedge y \geq z\}$ *S* $\{x \leq y\}$

2. $\{x \leq y\}$

   while($y \geq z$) S

   $\{x \leq y \wedge \neg(y \geq z)\}$ While 1

Assignment Statement Rule

$$\frac{}{\{A[\varepsilon/\nu]\}\ \nu := \varepsilon \quad \{A\}}$$

where, as in predicate logic, $A[\varepsilon/\nu]$ denotes the result of replacing free occurrences of variable ν in A with ε.

(This rule has an **empty** antecedent.)

- "Assignment" here should not be confused with assignment as in the interpretation of logic formulas.
- Those assignments are like program states.

Example of Assignment Rule

$$\{A[\varepsilon/\nu]\} \quad \nu := \varepsilon \qquad \{A\}$$

1. $\{x \leq z\}$ **y := z** $\{x \leq y\}$ Assignment

Here ν is identified with y

ε is identified with z

More Examples of Assignment Rule

$$\{A[\varepsilon/\nu]\} \quad \nu := \varepsilon \quad \{A\}$$

1. $\{x \leq y+1\}$ **y := y+1** $\{x \leq y\}$ Assignment
2. $\{x*y \leq n\}$ **y := x*y** $\{y \leq n\}$ Assignment
3. $\{x+1 \leq n+1\}$ **x := x+1** $\{x \leq n+1\}$ Assignment

Examples of Derivations of Small Programs: Exchange Program

To derive: A program that exchanges the values in variables x and y.

$\{x = x_0 \wedge y = y_0\}$ **z := x; x := y; y := z;** $\{y = x_0 \wedge x = y_0\}$

1. $\{z = x_0 \wedge x = y_0\}$ **y := z;** $\{y = x_0 \wedge x = y_0\}$ Assignment
2. $\{z = x_0 \wedge y = y_0\}$ **x := y;** $\{z = x_0 \wedge x = y_0\}$ Assignment
3. $\{x = x_0 \wedge y = y_0\}$ **z := x;** $\{z = x_0 \wedge y = y_0\}$ Assignment
4. $\{z = x_0 \wedge y = y_0\}$ **x := y; y := z;** $\{y = x_0 \wedge x = y_0\}$ Comp 2, 1
5. $\{x = x_0 \wedge y = y_0\}$ **z := x; x := y; y := z;** $\{y = x_0 \wedge x = y_0\}$ Comp 3, 4

Examples of Derivations of Small Programs: Ordering two numbers

- $\{x = x_0 \wedge y = y_0\}$
  if(x > y) {z := x; x := y; y := z;}
  $\{x \leq y \wedge ((x = x_0 \wedge y = y_0) \vee (y = x_0 \wedge x = y_0))\}$

- We'll obviously be needing the 1-sided conditional rule.
- We'll assume some things about the < and $\leq$ predicates:
- $\neg(x > y) \rightarrow (x \leq y)$
- $(y > x) \rightarrow (x \leq y)$
- Similar to the derivation on the previous page, we can derive:
- $\{x > y \wedge x = x_0 \wedge y = y_0\}$
  z := x; x := y; y := z;
  $\{y > x \wedge y = x_0 \wedge x = y_0\}$
  and using expectation weakening, we can replace the expectation with
  $\{x \leq y \wedge y = x_0 \wedge x = y_0\}$
- Then identify P in the 1-sided cond rule as: $x > y$

Examples of Derivations of Small Programs

- $\{x \leq n\}$ **while(x < n) x := x+1** $\{x = n\}$

- We can use the while rule, provided that we can rely on properties of **integer** arithmetic such as:

$$(x < n) \rightarrow ((x+1) \leq n)$$

$$((x \leq n) \wedge \neg(x < n)) \equiv (x = n)$$

Examples of Derivations of Small Programs

1. $((x \leq n) \wedge \neg(x < n)) \equiv (x = n)$ Premise
2. $(x < n) \rightarrow ((x+1) \leq n)$ Premise
3. $\{x+1 \leq n\}$ **x := x+1** $\{x \leq n\}$ Assignment
4. $\{x < n\}$ **x := x+1** $\{x \leq n\}$ Assumption strengthening 3, 2
5. $\{x \leq n \wedge x < n\}$ **x := x+1** $\{x \leq n\}$ Assumption strengthening 4
6. $\{x \leq n\}$ while($x < n$) **x := x+1** $\{x \leq n \wedge \neg(x < n)\}$ While 4
7. $\{x \leq n\}$ while($x < n$) **x := x+1**$\{x = n\}$ Expectation weakening 6