Programming and Hoare Logic

COMP SCI 2LC3

Ryszard Janicki

Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada

- We discuss some applications of predicate logic in computing:
  1. the formal specification of imperative programs
  2. the proof and development of sequences of assignments
  3. the calculation of parts of assignments (to avoid guess)
- We discuss the conditional statement and conditional expression

Specification of programs

- Execution of the assignment statement $x := E$ evaluates expression E and stores the result in variable x
- Assignment $x := E$ is read as "x becomes E"
- Execution of $x := E$ in a state stores in x the value of E in that state

Example

Suppose the state $s = (v, w, x) = (5, 4, 8)$. Consider the assignment $v := v + w$
The value of $v + w$ in the state is 9, so executing $v := v + w$ stores 9 in v, changing the state to $s' = (9, 4, 8)$.

Specification of programs

- A precondition of a statement is an assertion about the program variables in a state in which the statement may be executed

- A postcondition is an assertion about the states in which it may terminate

Example

Suppose the state $s = (v, w, x) = (5, 4, 8)$. Consider the assignment $v := v + w$

- The precondition could be $v = 5 \ \wedge \ w = 4 \ \wedge \ x = 2w$

- The postcondition could be $v' = v + w \ \wedge \ w' = w \ \wedge \ x' = x$

Some important questions:

- From a precondition for an assignment, how can we determine a corresponding postcondition?

- From a postcondition, can we determine a suitable precondition?

The conventional way of indicating a precondition and a postcondition for a statement S is $\{P\}\ S\ \{Q\}$ where

- P is the precondition
- Q is the postcondition

$\{P\}\ S\ \{Q\}$ is known as a Hoare triple

Specification of programs

- $\{Q\}\ S\ \{R\}$ has the interpretation

  Execution of S begun in any state in which Q is true is guaranteed to terminate, and R is true in the final state.

- $\{\text{true}\}\ S\ \{x = y\}$ says that the execution of S begun in any state is guaranteed to terminate at a state satisfying $x = y$
- If precondition Q is false, then the Hoare-triple guarantees nothing about execution, so execution of S can do anything!

Example

- $\{x = 0\}\ x := x + 1\ \{x > 0\}$ is a Hoare triple that is valid iff execution of $x := x + 1$ in any state in which $x = 0$ terminates in a state in which $x > 0$
- $\{x > 5\}\ x := x + 1\ \{x > 0\}$ VALID
- $\{x + 1 > 0\}\ x := x + 1\ \{x > 0\}$ VALID
- $\{x = 5\}\ x := x + 1\ \{x = 7\}$ NOT VALID

Definition (Assignment)

$\{R[x := E]\}\ x := E\ \{R\}$

Example

- $\{x + 1 > 5\}\ x := x + 1\ \{x > 5\}$
- $\{5 \neq 5\}\ x := 5\ \{x \neq 5\}$
- $\{x^2 > x^2 \cdot y\}\ x := x^2\ \{x > x \cdot y\}$

Note that R and $R[x := E]$ are exactly the same except that where R has an occurrence of x while $R[x := E]$ has an occurrence of "E"

- In some programming languages, the assignment statement is extended to the multiple assignment
  $x_I, x_2, \cdots, x_n := E_1, E_2, \cdots, E_n$, where the x_i are distinct variables and the E_i are expressions

- The multiple assignment is executed as follows:
  1. Evaluate all the expressions E_i to obtain a value v_i
  2. Assign v_i to $x_i, \cdots$, and finally v_n to x_n.

- Note that all expressions are evaluated before any assignments are performed

Specification of programs

Example

- $x, y \;:=\; y, x$
- $x, i \;:=\; 0, 0$
- $x, i \;:=\; x+1, i+1$
- $i, x \;:=\; i+1, x+1$
- $x, y \;:=\; x+y, x+y$
- $x \;:=\; x+y;\; y \;:=\; x+y$

Specification of programs

- Multiple assignment is defined in terms of simultaneous textual substitution

- $\{R[x, y := E, F]\}\ x, y := E, F\ \{R\}$

- $\{R[y := F][x := E]\}\ x := E; y := F\ \{R\}$

Example

- $\{y > x\}\ x, y := y, x\ \{x > y\}$

Problem (and Solution)

- $\{\ \ ?\ \ \}\ x, i := x + i, i + 1\ \{x = 1 + 2 + \cdots + (i - 1)\}$
  $?$ *is* $x + i = 1 + 2 + \cdots + (i + 1 - 1)$
- $\{\ \ ?\ \ \}\ i, x := i + 1, x + i\ \{x = 1 + 2 + \cdots + (i - 1)\}$
  $?$ *is* $x + i = 1 + 2 + \cdots + (i + 1 - 1)$

Specification of programs

Example

1. $\{\text{true}\}\ x := 2\ \{\text{true}\}$
2. $\{\text{false}\}\ x := 2\ \{\text{false}\}$
3. $\{\text{true}\}\ x, y := 1, 2\ \{\text{true}\}$
4. $\{\text{false}\}\ x, y := 1, 2\ \{\text{false}\}$
5. $\{x + 7 + y > 20\}\ x := x + 7\ \{x + y > 20\}$, or
   $\{x + y > 13\}\ x := x + 7\ \{x + y > 20\}$
6. $\{(x-1)^2 + 2(x-1) = 3\}\ x := x - 1\ \{x^2 + 2x = 3\}$, or
   $\{x^2 = 4\}\ x := x - 1\ \{x^2 + 2x = 3\}$
7. $\{((x-1)+1)((x-1)-1) = 0\}\ x := x - 1\ \{(x+1)(x-1) = 0\}$, or
   $\{x(x-2) = 0\}\ x := x - 1\ \{(x+1)(x-1) = 0\}$
8. $\{x + y = x\}\ y := x + y\ \{y = x\}$, or
   $\{y = 0\}\ y := x + y\ \{y = x\}$
9. $\{x + y = x + x + y\}\ y := x + y\ \{y = x + y\}$, or
   $\{x = 0\}\ y := x + y\ \{y = x + y\}$

Specification of programs

A specification of a program should give:

1. a precondition Q
2. a list x of variables that may be assigned to
3. a postcondition R

We denote such a specification by $\{Q\}\ x\ :=\ ?\ \{R\}$

Example

- $\{\mathrm{true}\}\ x\ :=\ ?\ \{x^2 = 9\}$
- $\{x < 0\}\ x\ :=\ ?\ \{x > 0\}$

Specification of programs

- A specification can be non-deterministic
- $\{\text{true}\}\ b\ := ?\ \{b^2 = 25\}$ specifies a program that in any initial state stores a value in b so that $b^2 = 25$
- A program that satisfies this specification can assign either -5 or 5 to b
- To formalize an English description of a program we have to define a precondition and a postcondition
- Some of the restrictions are implicit in the English specification ($\Longrightarrow$ we may have to invent variables)

Specification of programs

- Consider the following English specification:

  Find an integer approximation to the square root of integer n.

- It is implicit that $0 \leq n$

- The integer approximation has to be stored in some variable (we call it d)

- We must precisely define what is acceptable as an approximation

- We have derived the formal specification

$$\{0 \leq n\}\ d\ := ?\ \{d^2 \leq n < (d+1)^2\}$$

Specification of programs

We want to specify a program that finds the index of a value x in an array $b[0, \cdots, n-1]$

- Informal specification: "Find x in b"

- A precise definition must give conditions on b and n
  - Can the array segment be empty ($n = 0$)?
  - How should the index of x in b be indicated?
  - Can we assume that x is actually in the array segment?
  - If not, how should its absence be indicated?

Formal specification: Four acceptable specifications

1. $\{x \in b[0, \cdots, n-1]\}\ i\ := ?\ \{0 \leq i < n\ \wedge\ x = b[i]\}$

2. $$\{0 \leq n\}\ i\ := ?\ \{(0 \leq i < n\ \wedge\ x = b[i]) \vee (i = n\ \wedge\ x \notin b[0, \cdots, n-1])\ \}$$

3. $$\{0 \leq n\}\ i\ := ?\ \{(0 \leq i < n\ \wedge\ x \notin b[0, \cdots, i-1]) \wedge\ (x = b[i] \vee i = n)\ \}$$

4. $$\{0 \leq n\}\ i, c\ := ?\ \{(c \iff x \in b[0, \cdots, n-1]) \wedge\ (c \implies x = b[i])\ \}$$

Specification of programs

- One can replace $x \in b[0, \cdots, n-1]$ by $\exists(k \mid 0 \leq k < n : x = b[k])$
- There may be many ways to formalize an English specification
- It takes experience, thought, and care to be able to do it well
- Developing a clear and rigorous (if not formal) specification is an important part of programming
- The more complicated the problem being tackled, the more important are good specifications

Reasoning about the assignment statement

- We defined the (multiple) assignment $x \;:=\; E$ by

$$\{R[x := E]\}\; x := E\; \{R\}$$

Assumption: E is total

- Many expressions are not total

- For each expression E, we define the predicate $\mathrm{dom}\,(E)$ to be satisfied in exactly those states in which E is defined

Example

$$\mathrm{dom}\,(\sqrt{\tfrac{x}{y}}) \iff y \neq 0 \;\wedge\; \tfrac{x}{y} \geq 0$$

Reasoning about the assignment statement

- We can use different definitions of dom for different purposes

Example

$x = x + y$ here E is $x + y$

- When first writing a program, we assume that $\mathbb{Z}$ contains all the integers and write $\text{dom}\,(x + y) \iff \text{true}$
- To prove that no overflow occurs on a computer whose range of integers is $-2^{16} + 1 \cdots 2^{16} - 1$

$$\text{dom}\,(x + y) \iff -2^{16} < x + y < 2^{16}$$

More general definition of assignment:
$\{\text{dom}\,(E) \;\wedge\; R[x := E]\}\; x := E\; \{R\}$

- We claim that $R[x := E]$ is the weakest precondition
- Another precondition Q satisfies $\{Q\}\ x := E\ \{R\}$ if and only if $Q \implies R[x := E]$ holds

Assignment introduction (proof method):

To show that $x := E$ is an implementation of $\{Q\}\ x := ?\ \{R\}$, prove

$$\boxed{Q \implies R[x := E]}$$

Reasoning about the assignment statement

Example

Specification	Implementation
$\{x > 0\}\ x\ := ?\ \{x > 1\}$	$x\ :=\ x + 1$
Let P be $0 \leq i \leq n\ \wedge$ $x =\ +(k\ \mid\ 0 \leq k < i\ :\ b[k])$	
$\{P\ \wedge\ I = i \neq n\}\ x, i\ := ?$ $\{P\ \wedge\ i = I + 1\}$	$x, i\ :=\ x + b[i], i + 1$

Reasoning about the assignment statement

- Suppose we want to find the weakest precondition such that execution of a sequence $x := E; y := F$ of assignments will terminate with R true:

$$\{?\}\ x := E; y := F\ \{R\}$$

- We know how to find the weakest precondition for one assignment

Example

1. $\{?\}\ x := E; y := F\ \{R\}$
2. $\{?\}\ x := E; \{R[y := F]\} y := F\ \{R\}$
3. $\{R[y := F][x := E]\}\ x := E; \{R[y := F]\} y := F\ \{R\}$

Generalisation:

$$\{R[x_n := E_n]\cdots[x_2 := E_2][x_1 := E_1]\}$$
$$x_1 := E_1; x_2 := E_2; \cdots x_n := E_n$$
$$\{R\}$$

Reasoning about the assignment statement

Example

1. Find the weakest precondition such that execution of $t := x; x := y; y := t$ leads to state that satisfies $x = X \land y = Y$
2. Calculate and simplify the weakest precondition for the following (where x and y are integers) $\{?\}\, x := x + y; y := x - y; x := x - y\, \{x = X \land y = Y\}$

Problem

Define a predicate $perm(b, c, n)$ *that means: array segment* $b[O \cdots n-1]$ *is a permutation of array segment* $c[O \cdots n-1]$*. (One array segment is a permutation of another if its values can be interchanged (swapped) so the two segments are equal. For example,* $(3, 5, 2, 5)$ *is a permutation of* $(2, 3, 5, 5)$*.*

Calculating parts of assignments

- Consider maintaining

$$P_1 : x = \; +(k \mid 0 \leq k < i \; : \; b[k])$$

using an assignment $i, x \; := \; i+1, e$, where we assume that e is unknown

- How e can be calculated, instead of guessed?
  - Hoare triple: $\{P_1\}\; i, x \; := \; i+1, e \; \{P_1\}$
  - Hoare triple is valid when $P_1 \implies P_1[i, x \; := \; i+1, e]$

Calculating parts of assignments - a proof

Proof.

$$
\begin{array}{ll}
 & P_1[i, x := i+1, e] \\
= & \langle \text{ Definition of } P_1 \quad \& \quad \text{Textual substitution } \rangle \\
 & e = +(k \mid 0 \le k < i+1 : b[k]) \\
= & \langle \text{ Split off term } \rangle \\
 & e = +(k \mid 0 \le k < i : b[k]) + b[i] \\
= & \langle \text{ Assumption i.e., } x = +(k \mid 0 \le k < i : b[k]) \rangle \\
 & e = x + b[i]
\end{array}
$$

∎

Calculating parts of assignments

Problem

1. *Consider solving for e in*
   $\{P_2 \ : \ x = \ +(k \mid i \le k < n \ : \ b[k])\} \ i, x \ := \ i-1, e \ \{P_2\}$
   (see page 186 of Gries-Schneider for a solution)

2. *Suppose the number of apples that Mary and John have (represented by m and j, respectively) are related by the formula (C is some constant)* $P \ : \ C = m + 2j$.
   Find a solution for e in $\{P \ \wedge \ \mathrm{even}(m)\} \ m, j \ := \ m/2, e \ \{P\}$.

- The conditional statement, call it IF, has the following form

$$\text{IF: if } B \text{ then } S_1 \text{ else } S_2,$$

  - B is a boolean expression
  - S_1 and S_2 are statements
- We can annotate IF to illustrate the execution of IF.

$$\begin{array}{l} \{Q\} \\ \text{if } B \text{ then } \{Q \wedge B\}\ S_1\ \{R\} \\ \qquad\quad \text{else } \{Q \wedge \neg B\}\ S_2\ \{R\} \\ \{R\} \end{array}$$

Conditional statements and expressions

Proof method for IF.: To prove $\{Q\}$ IF $\{R\}$, it suffices to prove

1. $\{Q \ \wedge \ B\}\ S_1\ \{R\}$, and
2. $\{Q \ \wedge \ \neg B\}\ S_2\ \{R\}$

- The statement S_2 can be a skip (it does nothing)
- skip satisfies $\{R\}$ skip $\{R\}$

Conditional statements and expressions

Example

To prove

$\{\text{true}\}$
if $x \leq y$ then skip else $x, y := y, x$
$\{x \leq y\}$

we prove the following,

1. $\{\text{true} \wedge x \leq y\}$ skip $\{x \leq y\}$
2. $\{\text{true} \wedge \neg(x \leq y)\}\ x, y := y, x\ \{x \leq y\}$

Conditional statements and expressions

- The statement if B then S_1 else S_2 can be written in the notation of guarded commands as the alternative statement:

$$
\begin{array}{ll}
\text{if} & B \longrightarrow S_1 \\
[] & \neg B \longrightarrow S_2 \\
\text{fi} &
\end{array}
$$

- In the guarded command notation, an alternative statement can be written with more than two possible choices

$$
\mathrm{IFG}: \quad
\begin{array}{ll}
\text{if} & B_1 \longrightarrow S_1 \\
[] & B_2 \longrightarrow S_2 \\
[] & B_3 \longrightarrow S_3 \\
\text{fi} &
\end{array}
$$

Conditional statements and expressions

$$
\begin{array}{lll}
\textcolor{red}{\mathrm{IFG}}: & \mathrm{if} & B_1 \longrightarrow S_1 \\
 & [] & B_2 \longrightarrow S_2 \\
 & [] & B_3 \longrightarrow S_3 \\
 & \mathrm{fi} &
\end{array}
$$

There are two key points with the alternative statement.

1. Execution aborts if no guard is true
2. If more than one guard is true, only one of them is chosen (arbitrarily) and its corresponding command is executed.

$$\begin{array}{llll} \text{IFG}: & \text{if} & B_1 \longrightarrow S_1 \\ & [] & B_2 \longrightarrow S_2 \\ & [] & B_3 \longrightarrow S_3 \\ & \text{fi} & \end{array}$$

Proof method for IFG: To prove $\{Q\}$ IFG $\{R\}$, it suffices to prove:

1. $Q \implies B_l \vee B_2 \vee B_3$
2. $\{Q \wedge B_1\}\ S_1\ \{R\}$
3. $\{Q \wedge B_2\}\ S_2\ \{R\}$
4. $\{Q \wedge B_3\}\ S_3\ \{R\}$

Example

$$
\begin{array}{lll}
\color{red}{S}: & \text{if} & w > x \longrightarrow w, x := x, w \\
 & [\!] & x > y \longrightarrow x, y := y, x \\
 & [\!] & y > z \longrightarrow y, z := z, y \\
 & \text{fi} &
\end{array}
$$

Prove or disprove $\{\neg(x > y \vee w > x)\}\ S\ \{w \leq x \leq y \leq z\}$.

Problem

Prove that the following annotated program is correct.

1. $\{x > 5\}$ if $x \leq y$ then skip else $x, y := y, x$ $\{x \leq y\}$
2. $\{x = X\}$ if $x < 0$ then $x := -x$ else skip $\{x = |X|\}$

Proof method for IF.: To prove $\{Q\}$ while B do S od $\{\neg B \wedge Q\}$, it suffices to prove

$$\{Q \ \wedge \ B\}\ S\ \{Q\}.$$

Example

To prove

$\{x \leq 10\}$
while $x < 10$ do $x := x + 1$ od
$\{\neg(x < 10) \wedge x \leq 10\}$

we prove the following,

- $\{x \leq 10 \ \wedge \ x < 10\}\ x := x + 1\ \{x \leq 10\}$, or simplified
- $\{x < 10\}\ x : +x + 1\ \{x \leq 10\}$,

which is easily obtained by the assignment rule. Finally, the postcondition $\{\neg(x < 10) \wedge x \leq 10\}$ can be simplified to $\{x = 10\}$.

- Finding loop invariants is difficult! It involves induction
- We will discuss this problem later.

The following inference rules create so called ‘Hoare Logic’

- Empty statement

$$\frac{}{\{P\}\ \text{skip}\ \{P\}}$$

- Assignment

$$\frac{}{\{P[x := E]\}\ x := E\ \{P\}}$$

- Composition

$$\frac{\{P\}\ S\ \{Q\}\ ,\ \{Q\}\ T\ \{R\}}{\{P\}\ S; T\ \{R\}}$$

- if-then-else

$$\frac{\{B \wedge P\}\ S\ \{Q\}\ ,\ \{\neg B \wedge P\}\ T\ \{Q\}}{\{P\}\ \text{if}\ B\ \text{then}\ S\ \text{else}\ T\ \{Q\}}$$

- while-do

$$\frac{P \wedge B}{\{P\}\ \text{while}\ \ B\ \ \text{do}\ \ S\ \ \text{od}\ \{\neg B \wedge P\}}$$

- Consequence (often omitted)

$$\frac{P_1 \implies P_2 \;,\; \{P_2\}\; S\; \{Q_2\} \;,\; Q_2 \implies Q_1}{\{P_1\}\; S\; \{Q_1\}}$$

Finding **loop invariants** is the most complicated task! In general we cannot calculate them! We have to guess them (induction!).

- Sometimes $\{Q\}$ while B do S od $\{R\}$ can be proven for (almost) arbitrary Q and R!

Example

$\{P \wedge x = 0\}$ while $i \leq 4$ do $x := x + 1; S$ od $\{R\}$

$\Longleftrightarrow$

$\{P \wedge x = 0\}$

$x := x + 1; S;$

$x := x + 1; S;$

$x := x + 1; S;$

$x := x + 1; S$

$\{R\}$

- For each *constant* n, we can prove

  $$\text{for } i = 1 \text{ to } n \text{ do } S \text{ od}$$

  by unfolding for and using composition inference rule $n-1$ times.

Example

$\{Q\}$ for $i = 1$ to 4 do S od $\{R\}$

$\Longleftrightarrow$

$\{Q\}\ S;\ S;\ S;\ S\ \{Q\}$

Similarly for

$$\text{for } i = 1 \text{ to } 10^{10^{10}} \text{ do } S \text{ od},$$

one just needs to use composition $10^{10^{10}} - 1$ times.