CISC 5835: Algorithms for Data Analyitcs

An interlude: Loop invariants and Hoare axiomatics

Arthur G. Werschulz

Fordham University
Department of Computer and Information Sciences

Fall, 2019

Program verification

- When is a program correct?
- Why not simply test?
  - Exhaustive testing? Too many possibilities: a program that simply takes two 64-bit numbers as input requires $2^{128} \doteq 3.5 \times 10^{38}$ tests.
    If we could do 10^{12} tests per second, this would be about 10^{31} years.
  - Test all execution paths? If program has 20 **if**/**else** statements, there will be 2^{20} possibilities to check.
  - "Testing reveals the presence of bugs, not their absence." (Edsger Dikstra, 1969).
- What do you mean by "correctness" in the first place?

Criterion for correctness

Definition

Let Pre and Post be Boolean formulas (propositions), called *preconditions* and *postconditions*, respectively. For an algorithm A, we write

$$\{\text{Pre}\}A\{\text{Post}\} \tag{1}$$

to mean "if Pre is true before we execute A, then Post will be true after A terminates."

1. If (1) is true whenever A terminates, we say that A is *partially correct* with respect to Pre and Post.
2. If (1) is always true, we say that A is *totally correct* with respect to Pre and Post.

Partial vs. total correctness

- Total correctness = Partial correctness + termination.
- To see the difference, consider:

  $\{n \in \mathbb{N}_0\}$
  while $n \neq 1$
      if n is even
          $n \leftarrow n/2$
      else
          $n \leftarrow 3n+1$
  $\{n = 1\}$
- Partially correct (why?)
- Unknown whether this always terminates.
  (Known to terminate for $n \leq 87 \cdot 2^{60} \doteq 10^{20}$.)
- So unknown whether this is totally correct.

Partial vs. total correctness (cont'd)

- Why not simply write a program to check correctness?
- You can't!

Definition

The *halting function* $h : \{algorithms\} \to \{\text{true}, \text{false}\}$ is given by

$$h(A) = \begin{cases} \text{true} & \text{if } A \text{ halts after finitely-many steps,} \\ \text{false} & \text{otherwise.} \end{cases}$$

Theorem (Turing, 1936)

There is no algorithm that implements the halting function.

- So must use ad hoc techniques to prove that a given algorithm halts.

Hoare's axioms

Main ideas:

- Express algorithms using a small number of basic constructs
- Most basic: assignment statement
- Recursively defined (in terms of other statements):
  - A sequence of statements
  - Selection statement (**if**/**else** and **if**)
  - Iteration statement (**while**/**do**)
- Can reduce other control statements (e.g., **case**, **repeat**/**until**, **for**/**do**) to the ones above.
- Associate a transformation rule $\{P\}S\{Q\}$ (meaning: if P is true before executing S, then Q will be true afterwards) for each statement type S, where P and Q are Boolean conditions
- Only use transformation rules to reason about an algorithm.

Hoare's axioms: the assignment rule

Definition

Assignment rule: Let $P(x)$ be a predicate having free variable x. Let v be a variable and let e be an expression. Then

$$\{P(e)\}\, v \leftarrow e \,\{P(v)\} \tag{2}$$

Moreover:

- If $P(v)$ is known to be true after $v \leftarrow e$, then $P(e)$ is the weakest precondition such that (2) holds.
- If $P(e)$ is known to be true before $v \leftarrow e$, then $P(v)$ is the strongest postcondition such that (2) holds.

Assignment rule examples

Example

What's the weakest precondition such that $y = 1$ after executing $y \leftarrow x + 2$? **Answer**: $x + 2 = 1$, i.e., $x = -1$.

Example

What's the weakest precondition such that $y = z + 12$ after executing $y \leftarrow w + x$?
Answer: $z + 12 = w + x$.

Using the assignment rule

In the following running example, we'll assume that all variablesare integer-valued.

Example (extended)

What's the weakest precondition such that $(p = i \cdot n) \wedge (i \leq m)$ after executing $i \leftarrow i + 1$?

Answer: $(p = (i+1) \cdot n) \wedge (i + 1 \leq m)$, which is equivalent to $(p = i \cdot n + n) \wedge (i < m)$.

What's the weakest precondition such that $(p = i \cdot n + n) \wedge (i < m)$ holds after executing $p \leftarrow p + n$?

Answer: $(p + n = i \cdot n + n) \wedge (i < m)$, which is equivalent to $(p = i \cdot n) \wedge (i < m)$.)

A notational convention

To save space, we'll use the notation

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

to mean the following:
Let the condition R be true.

- If P is true before S executes, then Q will be true after S executes.
- If Q is true after S executes, then P was true before S executes.

Hoare's axioms: the composition rule

Definition

Composition rule: Let S_1 and S_2 be statements, and let $S_1; S_2$ denote their composition (i.e., S_1 followed by S_2). Then

$$\frac{\{P_0\}\, S_1\, \{P_1\} \wedge \{P_1\}\, S_2\, \{P_2\}}{\{P_0\}\, S_1; S_2\, \{P_2\}}$$

Once we accept this rule, we see that it extends to any number of statements:

Theorem

Let $S_1, S_2, \ldots, S_n$ be statements. Then

$$\frac{\{P_0\}\, S_1\, \{P_1\} \wedge \{P_1\}\, S_2\, \{P_2\} \wedge \cdots \wedge \{P_{n-1}\}\, S_n\, \{P_n\}}{\{P_0\}\, S_1; S_2; \ldots; S_n\, \{P_n\}}$$

Proof sketch.

Use mathematical induction on n. □

Using the composition rule

Example (extended, cont'd)

From an earlier slide, we know

$\{(p = i \cdot n) \wedge (i < m)\}$
$p \leftarrow p + n$
$\{(p = (i+1) \cdot n) \wedge (i < m)\}$

and

$\{(p = (i+1) \cdot n) \wedge (i < m)\}$
$i \leftarrow i + 1$
$\{(p = i \cdot n) \wedge (i \leq m)\}$

So by the composition rule, we have

$\{(p = i \cdot n) \wedge (i < m)\}$
$p \leftarrow p + n; i \leftarrow i + 1$
$\{(p = i \cdot n) \wedge (i \leq m)\}$

Using the composition rule

Example (extended, cont'd)

We had

$\{(p = i \cdot n) \wedge (i < m)\}$
$p \leftarrow p + n; i \leftarrow i + 1$
$\{(p = i \cdot n) \wedge (i \leq m)\}$

But $i < m \equiv (i \leq m) \wedge (i < m)$, and so

$\{[(p = i \cdot n) \wedge (i \leq m)] \wedge (i < m)\}$
$p \leftarrow p + n; i \leftarrow i + 1$
$\{(p = i \cdot n) \wedge (i \leq m)\}$

Hence $\{(p = i \cdot n) \wedge (i \leq m)\}$ is an *invariant* for the statement pair

$$p \leftarrow p + n;\ i \leftarrow i + 1$$

provided that $i < m$.

Hoare's axioms: the selection rule

Definition (Selection rule (**if**/**else** statement):)

Let S_1 and S_2 be statements. Then

$$\frac{(\{P \wedge B\} S_{\mathrm{T}} \{Q\}) \wedge (\{P \wedge \neg B\} S_{\mathrm{F}} \{Q\})}{\{P\} \textbf{ if } B \textbf{ then } S_{\mathrm{T}} \textbf{ else } S_{\mathrm{F}} \{Q\}}$$

Corollary (Selection rule (**if** statement):)

Let S be a statement. Then

$$\frac{(\{P \wedge B\} S \{Q\}) \wedge (\{P \wedge \neg B \Rightarrow Q\})}{\{P\} \textbf{\textit{ if }} B \textbf{\textit{ then }} S \{Q\}}$$

Proof.

Use **if**/**else** rule with $S_{\mathrm{T}} = S$ and $S_{\mathrm{F}} =$ empty statement. □

Hoare's axioms: the selection rule

Example

The assignment rule tells us that

$$\{(x \in \mathbb{Z}) \wedge (x < 0)\}\ abs \leftarrow -x\ \{abs = |x|\}$$

and (of course)

$$\{(x \in \mathbb{Z}) \wedge (x \geq 0)\}\ abs \leftarrow x\ \{abs = |x|\}$$

So selection rule tells us

$\{x \in \mathbb{Z}\}$
if $x \geq 0$
then $abs \leftarrow x$
else $abs \leftarrow -x$
$\{abs = |x|\}$

Hoare's axioms: the iteration rule

Definition (Iteration rule (*while/do* statement))

Let S be a statement. Then

$$\frac{\{P \wedge B\}\, S\, \{P\}}{\{P\} \textbf{ while } B \textbf{ do } S\ \{P \wedge \neg B\}}$$

provided the loop terminates.

Using the iteration rule

Example (extended, cont'd)

We previously showed

$\{(p = i \cdot n) \wedge (i \leq m) \wedge (i < m)\}$
$p \leftarrow p + n$
$i \leftarrow i + 1$
$\{(p = i \cdot n) \wedge (i \leq m)\}$

So the iteration rule tells us that

$\{(p = i \cdot n) \wedge (i \leq m)\}$
while $i < m$ **do**
  $p \leftarrow p + n$
  $i \leftarrow i + 1$
$\{(p = i \cdot n) \wedge (i \leq m) \wedge \neg(i < m)\}$

Using the iteration rule (cont'd)

Example (extended, cont'd)

But

$$\begin{aligned}
&(p = i \cdot n) \wedge (i \leq m) \wedge \neg(i < m) \\
&\quad \equiv (p = i \cdot n) \wedge (i = m) \\
&\quad \Rightarrow p = m \cdot n
\end{aligned}$$

and so (weaken the post-condition)

$\{(p = i \cdot n) \wedge (i \leq m)\}$
while $i < m$ **do**
  $p \leftarrow p + n$
  $i \leftarrow i + 1$
$\{p = m \cdot n\}$

Using the iteration rule (cont'd)

Example (extended, cont'd)

We have

$\{(p = i \cdot n) \wedge (i \leq m)\}$
while $i < m$ **do**
  $p \leftarrow p + n$
  $i \leftarrow i + 1$
$\{p = m \cdot n\}$

Need initialization statements to force loop precondition.
Let's try

$$\begin{aligned}
&p \leftarrow 0 \\
&i \leftarrow 0
\end{aligned}$$

Example (extended, cont'd)

Assignment rule gives

$\{(p = 0 \cdot n) \wedge (0 \leq m)\}$
$i \leftarrow 0$
$\{(p = i \cdot n) \wedge (i \leq m)\}$

with the precondition simplifying to $(p = 0) \wedge (0 \leq m)$.
Once more with assignment rule:

$\{(0 = 0) \wedge (0 \leq m)\}$
$p \leftarrow 0$
$\{(p = 0) \wedge (0 \leq m)\}$

with the precondition simplifying to $0 \leq m$, or $m \geq 0$.

Example (extended, cont'd)

So by the composition rule, we have

$\{m \geq 0)\}$
$p \leftarrow 0$
$i \leftarrow 0$
$\{(p = i \cdot n) \wedge (i \leq m)\}$

Gluing this all together (via the composition rule), we get ...

Example (extended, cont'd)

$\{m \geq 0)\}$
$p \leftarrow 0$
$i \leftarrow 0$
$\{(p = i \cdot n) \wedge (i \leq m)\}$
while $i < m$ **do**
    $p \leftarrow p + n$
    $i \leftarrow i + 1$
$\{p = m \cdot n\}$

So if $m \geq 0$ before code is run, we'll have $p = m \cdot n$ when it's done.

Example (extended, cont'd)

- What have we overlooked?
- Don't know whether it terminates after finitely-many steps! (Only know partial corretness!)
- Not too hard in this case:
  - m never changes.
  - Before doing a loop iteration, we check that $i \leq m$.
  - After each loop iteration, we increment i.
  - So i is bounded from above by a fixed number and increases each time we do a loop iteration.
  - This can only happen finitely-many times.
  - So loop only executes finitely-many times.
- So algorithm is totally correct, as anotated.