Using Logic on Computation Systems

Proofs of Performance

Specification of a Computational System

- By computational system, we could mean any of the following:
  - Abstract model (FSA, TM, PDA, …)
  - Single program
  - Interacting collection of programs
  - “Reactive” system, such as an operating system
  - Multi-agent systems, such as in AI and robotics.
- The language of logic can be used to specify desired **properties** of such systems.
- Aspects of logic such as predicates and functions can be used to **specify the system itself**.
- The **proof** aspect of logic can be used to prove that the properties hold.

Why Bother?

- Failure of a computational system can result in:
  - Loss of lives
  - Loss of money
  - Loss of time
  - General chaos

- While proving that a system works according to spec is time-consuming and requires extra effort and knowledge, it may be worth it compared to the value placed on not having the above items.

How Logic Enters (1)

- Model-checking approach (H&R, chapters 3 & 5):
  - Logic formulas are used to specify properties of a system.
  - A fixed logical interpretation (“model”) is used to capture the characteristics of the system itself.
  - The properties are verified with respect to the interpretation.
  - Most workable when the interpretation has a finite domain.
  - Some infinite domains can be accomodated (e.g. Petri net models).

How Logic Enters (2)

- Proof-checking approach (H&R, chapter 4):
  - Logic formulas are used to specify properties of a system.
  - Intended interpretation is characterized by axioms.
  - New rules of inference are added to represent program construction.
  - The properties are proved in the proof system relative to the axioms.
  - Works for arbitrary interepretations, not just ones with finite domain.
  - Generally undecidable.

Perspective

- We won't be able to cover H&R chapters 3&5. Some aspects these have been covered in a past offering of CS 156 (Parallel and Real-Time Computation).
- Chapter 5 discusses **temporal logic**, which is of interest in its own right. It also discusses **Kripke models**, which are essentially finite-state machines.
- There is the possibility of a future course dedicated to applications of logic. Let me know if you are interested.

Hoare Logic

Proofs of Programs

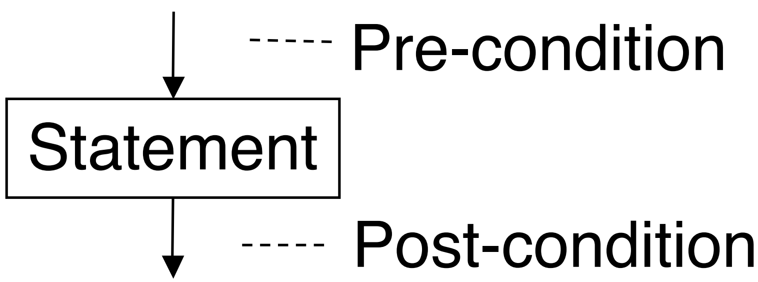
Essentially this is what Chapter 4 of H&R discusses.

Assumptions

- CS 60 talked about **partial** and **total correctness** with respect to an input/output specification.
- These traits were verified by insertion of **assertions** between statements and tests.
- A **loop invariant** is an assertion that goes before the test in a while loop.

Triple Notation
(C.A.R. (Tony) Hoare)

Instead of drawing



write

{Pre-condition} $\boxed{\text{Statement}}$ {Post-condition}

Tony did not use the boxes, but we will, for better clarity.

Composition of Triples
Expressible as *Inference Rules*

(from)

$$\{P\}\ \boxed{\text{Statement 1}}\ \{Q\}$$

$$\{Q\}\ \boxed{\text{Statement 2}}\ \{R\}$$

(infer)

$$\{P\}\ \boxed{\text{Statement 1; Statement 2}}\ \{R\}$$

Composition Rule Example

$$\{x+y > 0\}\ \boxed{\texttt{z = x+y;}}\ \{z > 0\}$$

$$\{z > 0\}\ \boxed{\texttt{x = z;}}\ \{x > 0\}$$

$$\{x+y > 0\}\ \boxed{\texttt{z = x+y; x = z;}}\ \{x > 0\}$$

Note:

The composition rule itself does not entail **justification** of the antecedents (formulas above the line). These are done as separate steps.

Other Inference Rules

- Implication Rule
- Conditional Rule
- One-armed Conditional Rule
- While Rule
- Assignment Rule

Implication Rule

$$\frac{\begin{array}{c}\{P\}\ \boxed{\text{Stmt 1}}\ \{Q\} \\ P' \Rightarrow P \\ Q \Rightarrow Q'\end{array}}{\{P'\}\ \boxed{\text{Stmt 1}}\ \{Q'\}}$$

In other words, in a forward derivation:

Pre-conditions can always be strengthened;

Post-conditions can always be weakened.

Implication Rule Example

$\{x+y > 0\}$ `z = x+y;` $\{z > 0\}$

$(x > 0 \wedge y > 0) \Rightarrow x+y > 0$

$z > 0 \Rightarrow z+5 > 0$

$\{x > 0 \wedge y > 0\}$ `z = x+y;` $\{z+5 > 0\}$

Conditional Rule

$$\{Q \wedge P\}\ \boxed{\text{Stmt 1}}\ \{R\}$$

$$\{Q \wedge \neg P\}\ \boxed{\text{Stmt 2}}\ \{R\}$$

$$\{Q\}\ \boxed{\text{if(P) Stmt 1 else Stmt 2}}\ \{R\}$$

Conditional Rule Example

$$\{z == x \wedge x \geq 0\}\ \boxed{\texttt{y = x;}}\ \{z == x \wedge y \geq 0\}$$

$$\{z == x \wedge x < 0\}\ \boxed{\texttt{y = -x;}}\ \{z == x \wedge y \geq 0\}$$

$$\{z == x\}$$

$$\boxed{\texttt{if(x} \geq \texttt{0) y = x; else y = -x;}}$$

$$\{z == x \wedge y \geq 0\}$$

One-Armed Conditional Rule

$$\frac{\begin{array}{c}\{Q \wedge P\}\ \boxed{\text{Stmt 1}}\ \{R\} \\ (Q \wedge \neg P) \Rightarrow R\end{array}}{\{Q\}\ \boxed{\text{if(P) Stmt 1}}\ \{R\}}$$

While Rule

$$\frac{\{Q \wedge P\}\ \boxed{\text{Stmt 1}}\ \{Q\}}{\{Q\}\ \boxed{\text{while(P) Stmt 1}}\ \{Q \wedge \neg P\}}$$

Q is the “loop invariant”

While Rule Example

$$\frac{\{\overbrace{x \geq 0}^{Q} \wedge \overbrace{x > 0}^{P}\}\ \boxed{\texttt{x = x-1;}}\ \{\overbrace{x \geq 0}^{Q}\}}{\{\underbrace{x \geq 0}_{Q}\}\ \boxed{\texttt{while(}\ \underbrace{\texttt{x > 0}}_{P}\ \texttt{) x = x-1;}}\ \{\underbrace{x \geq 0}_{Q} \wedge \underbrace{x \leq 0}_{\neg P}\}}$$

Assignment Rule

$$\overline{\{Q[\mathcal{E}/x]\}\ \boxed{x = \mathcal{E};}\ \{Q\}}$$

read “Q with
each occurrence of
x replaced with $\mathcal{E}$”

Assignment Rule Example

(no antecedent)

$\{x+y > 5\}$ $\boxed{\texttt{x = x+y;}}$ $\{x > 5\}$

$Q[x \leftarrow \mathcal{E}]$ $\qquad$ $x = \mathcal{E};$ $\qquad$ Q

Typically, we use the assignment rule by starting with a **desired post-condition**, which then determines the pre-condition mechanically.

Weakest Precondition

- Given a block B with a post-condition Q, the **weakest-precondition** for block B with post-condition Q is the pre-condition P that is **implied-by** any other pre-condition:

  {P} B {Q}

- This formula is sometimes written wp(B, Q).
- The given formula for the assignment rule in fact **constructs** the WP for Q.

Exercises
WP for Assignment Statements

- Examples:
  - {??} x = x+y; {y > x}
  - {??} y = 2*y; {y < 5}
  - {??} y = 2*y; {even(y)}

Composition Rule Example

- Consider

  $\{??\}\ x = z+1;\ y = x+y;\ \{y > 5\}$

- $wp(y = x+y;, y > 5) = x+y > 5$
- $wp(x = z+1;, x+y > 5) = z+1+y > 5$
- So ?? is

  $z+1+y > 5$

Derivation Example

Derive the following triple (where n! == n*(n-1)*…*1 and 0! == 1):

$\{ f == 1 \land k == n \land n \geq 0 \}$ `while( k > 0 ) { f = k*f; k = k-1;}` $\{ f == n! \}$

Approach:

Work backword.
Use the while rule, the composition rule, and the assignment rule (twice), and the implication rule.

While rule: What is the invariant Q?

It is necessary to provide some formulas for dealing with factorial (!).

The exit condition is of the form $(Q \land \neg P)$ where P is $k > 0$

suggesting that invariant Q might be: $f == n! / k! \land k \geq 0$

for then $(Q \land \neg P)$ is $f == n! / k! \land k \geq 0 \land \neg k > 0$

which is *equivalent to* $k == 0 \land f == n!$ which *implies* $f == n!$.

Derivation Example

The invariant is justified by the while rule, ***provided*** that we can derive:

$$\{f == n! / k! \land k \geq 0 \land k > 0\}\ \boxed{\texttt{f = k * f; k = k-1;}}\ \{f == n! / k! \land k \geq 0\}$$

$$(Q \land P)\ \{\text{Stmt } 1\}\ Q$$

But $k > 0 \Rightarrow k \geq 0$, so it is **sufficient** (by the implication rule) **to derive** the logically equivalent

$$\{ f == n! / k! \land k > 0 \}\ \boxed{\texttt{f = k * f; k = k-1;}}\ \{ f == n! / k! \land k \geq 0 \} \qquad (*)$$

Work backward from the post-condition using the **assignment rule**:

$$\{f == n!/(k-1)! \land (k-1) \geq 0 \}\ \boxed{\texttt{k = k-1;}}\ \{ f == n! / k! \land k \geq 0 \}$$

Derivation Example

Use the **assignment rule** again:

$$\{\underbrace{k * f == n!/(k-1)! \land (k-1) \geq 0}\}\ \boxed{\texttt{f = k * f;}}\ \{f == n!/(k-1)! \land (k-1) \geq 0\}$$

It is necessary to provide some formulas for dealing with factorial (!).

The left-hand formula **simplifies**, to get:

$$\{f == n!/k! \land k > 0\}\ \boxed{\texttt{f = k * f;}}\ \{f == n!/(k-1)! \land (k-1) \geq 0\}$$

since $n!/k! == n*(n-1)*(n-2)*\ldots(k+1)*k$

and, dividing by this by k gives $n*(n-1)*(n-2)*\ldots(k+1)$

which is $n! / k!$.

So now we have shown that (*) is derivable by the composition rule.

Derivation Example

All that is left to do is use the implication rule, with

$$\underbrace{f == 1 \wedge k == n \wedge n \geq 0}_{\text{overall pre-condition}} \Rightarrow \underbrace{f == n! \,/\, k! \wedge k \geq 0}_{\text{invariant}}$$

This is plausible, since from the left-hand side k == n, so k! == n!, and thus n! / k! == 1.

Derivation Exercise

Derive the following triple:

$\{x == x_0 \land y == y_0 \land x > 0 \land y > 0\}$

```
while( x != y )
    if( x > y )
      x = x - y;
    else
      y = y - x;
```

$\{ x == \gcd(x_0 , y_0) \}$

where gcd is the greatest-common-divisor function,

introducing and justifying any formulas you need for gcd.

Verifying Termination

- “Partial correctness” means that the program is correct, provided that it terminates.
- “Total correctness” is partial correctness and termination.
- Termination is often verified separately.

Verifying Termination

- The reason that termination is verified separately is that it requires coming up with a different sort of expression than an invariant.
- Such an expression is a “variant”. It describes a program’s inexorable movement toward a stopping point.

Variants

- Clearly the only cause for a (non-recursive) program's non-termination could lie in while-loops.
- A **variant** is some expression $\mathcal{E}$ such that:
  - $\mathcal{E} \geq 0$ is *invariant*, **and**
  - The value of $\mathcal{E}$ decreases at every iteration.
- If a loop has a variant, then the loop must terminate.

Variant Example

$\{x == x_0 \wedge y == y_0 \wedge x_0 \geq 0\}$

```
while( x > 0 )
   {
   y = y + k;
   x = x-1;
   }
```

$\{y == y_0 + k*x_0\}$

Here a **variant** for the loop would be x, since:

$x \geq 0$ is invariant, and

x decreases on each iteration.

Variant as a Triple

A sufficient condition for $\mathcal{E}$ to be a variant of

`while`(P) Stmt;

is that we be able to derive a triple:

$$\{\mathcal{E}_0 == \mathcal{E} \wedge \mathcal{E} > 0 \wedge \text{P}\}\ \text{Stmt}\ \{\mathcal{E}_0 > \mathcal{E} \wedge \mathcal{E} \geq 0\}$$

where $\mathcal{E}_0$ is a free variable.

Variant as a Triple: Example

$$\{\mathcal{E}_0 == \mathcal{E} \wedge \mathcal{E} > 0 \wedge \mathrm{P}\} \text{ Stmt } \{\mathcal{E}_0 > \mathcal{E} \wedge \mathcal{E} \geq 0\}$$

Consider the previous while program:

```
while( x > 0 )
   {
   y = y + k;
   x = x-1;
   }
```

$\{x_0 == x \wedge x > 0 \wedge x > 0\}$ `y = y + k; x = x-1;` $\{x_0 > x \wedge x \geq 0\}$
is the triple to be derived.

Variant as a Triple: Example

$\{x_0 == x \square x > 0 \square x > 0\}$ `y = y + k; x = x-1;` $\{x_0 > x \square x \geq 0\}$
is the triple to be derived.

Working backward from the post-condition, we need to derive:
$\{x_0 == x \square x > 0 \square x > 0\}$ `y = y + k;` $\{x_0 > x-1 \square x-1 \geq 0\}$

which follows from the implication rule if we can derive:

$$\{x_0 == x \square x > 0 \square x > 0\} \square \{x_0 > x-1 \square x-1 \geq 0\}$$

which follows directly (assuming x integer).

Exercise

- Derive a variant for the gcd program introduced earlier:

$\{x == x_0 \wedge y == y_0 \wedge x > 0 \wedge y > 0\}$

```
while( x != y )
    if( x > y )
      x = x - y;
    else
      y = y - x;
```

$\{ x == \gcd(x_0, y_0) \}$

Exercise

- Can a variant be derived for the similar triple:

$\{x == x_0 \land y == y_0 \land x \geq 0 \land y \geq 0\}$

```
while( x != y )
    if( x > y )
      x = x - y;
    else
      y = y - x;
```

$\{ x == gcd(x_0 , y_0) \}$

WP Calculus

- wp obeys some fairly obvious rules:
  - wp(x = $\mathcal{E}$;, Q) = Q [$\mathcal{E} / x$] as already stated
  - wp(B_1; B_2, Q) =
  - wp(if(P) B_1; else B_2 , Q) =
- wp for a loop is harder, because it generally requires an infinite formula (unwind the loop as an infinite nest of conditions).

Example: WP for a Test

- {??}
  if(x > y) x = x-y; else y = y-x;
  {gcd(x, y) == z}

- wp is

$$
\begin{aligned}
&(x > y) \wedge \overbrace{\text{gcd}(x\text{-}y,\ y) == z}^{\text{wp's of the assignment statements}} \vee \\
&\neg(x > y) \wedge \text{gcd}(x,\ y\text{-}x) == z
\end{aligned}
$$

Example 2: WP for a Test

- {??}
  if(x > y) z = x; else z = y;
  {z == max(x, y)}

- wp is

$$\begin{aligned}&(x > y) \land \overbrace{\text{max}(x, y) == x}^{\text{wp's of the assignment statements}} \lor \\ &\neg(x > y) \land \text{max}(x, y) == y\end{aligned}$$

  which simplifies to *true*.

When the *else* part is missing

- If the *else* part is missing, then T is effectively a “no-op”, “skip”, or trivial assignment $x = x$;
- Since wp($x = x$; Q) = Q
- the wp for

  if(P) S

  is then

  P □ wp(S, Q)

  ^ □P □ Q

Example: WP for a Test without else

- {??}
  if(x > y) y = x;
  {y = max(x, y)}

- wp is

$$
\begin{aligned}
&(x > y) \wedge \overbrace{x = \max(x, x)}^{\text{wp of the assignment statement}} \\
&\vee \neg(x > y) \wedge y = \max(x, y)
\end{aligned}
$$

- which simplifies to *true*.

Alternate WP for a Test

- wp(if(P) S else T, Q) =

  (P ^ wp(S, Q))
  (¬P ^ wp(T, Q))

- To see that this is equivalent to the previous version, let wp(S, Q) be A and wp(T, Q) be B. Then we are asking whether
  (P ^ A)   (¬P ^ B)
  is equivalent to
  (P ⇒ A) ^ (¬P ⇒ B)

Alternate WP for a Test

- (P ^ A) (□ P ^ B) =? (P □ A) ^ (□ P □ B)
- For P = true, this becomes A =? A.
- For P = false, this becomes B =? B
- Therefore the two forms are equivalent.