Axiomatic Semantics

- Reasoning about imperative programs using Mathematical Logic
- Formulas have the form { P } S { Q } where S is a program and P and Q are formulas of predicate logic, called *assertions*.
- P is a *precondition* and Q is a *postcondition*.
- Q expresses a property that should hold about the program. It often expresses program correctness.
- For example:

  { n >= 0 } factorial(n) { fact = n! }

  { x > 10 } sum = 2 * x + 1 { sum > 1 }

Programs and Assertions

- Grammars for assertions, booleans, expressions, and program statements:

  P ::= B | P and P | P or P | not P | P => P

  B ::= true | false | E = E | E <> E | E > E | E < E |
  E <= E | E >= E | …

  E ::= x | n | E + E | E – E | E * E | E / E | E! | …

  S ::= x := E | S;S | if B then S else S |
  while B do S end

- Note that x represents a variable from an infinite set, and n represents a number. We assume that all numbers are integers.
- Note that arithmetic and boolean expressions have no side effects.

Weakest Precondition

- The *weakest precondition* is a precondition that is the least restrictive (contains the least amount of information) but still guarantees that the postcondition will be true.

- Example:

  { x > 10 } sum := 2 * x + 1 { sum > 1 }

- x > 10 is not the weakest precondition. Other preconditions that make the formula true include x > 50, x > 1000, x > 0. The last one is the weakest precondition.

  { x > 0 } sum := 2 * x + 1 { sum > 1 }

Hoare Logic

- Axiomatic Semantics is also called Hoare Logic (developed by Hoare, 1969). { P } S { Q } is often called a *Hoare triple*.
- Reasoning is done using *inference rules*, of the form

$$\frac{A_1 \quad A_2 \quad \ldots \quad A_n}{A}$$

- If A_1, A_2, … A_n are true, then A is also true.
- A_1, A_2, … A_n are *premises* (or antecedents) and A is the *conclusion* (or consequent)
- An *axiom* is a logical statement that is always true (an inference rule with only a conclusion and no premises).

The Assignment Rule

- There are 5 rules, one for each kind of statement, and a rule of consequence.

  S ::= x := E | S;S | if B then S else S |

  while B do S end

- The rule for assignment is an axiom.

  $\{ Q_{x \to E} \}$ x := E $\{ Q \}$

- $Q_{x \to E}$ denotes substitution; it is the notation used in this chapter for [E/x]Q, which means that we replace all occurrences of x in Q with E.

More Assignment Examples

$$\boxed{\{ Q_{x \to E} \}\ x := E\ \{ Q \}}$$

Example:

$\{ x - 3 > 0 \}\ x := x - 3\ \{ x > 0 \}$

$\{ x > 3 \}\ x := x - 3\ \{ x > 0 \}$

$\{ n > 0 \}$

$\{ 0 >= 0 \text{ and } 0 <= n \text{ and } n > 0 \}$

$x := 0$

$\{ x >= 0 \text{ and } x <= n \text{ and } n > 0 \}$

Understanding Hoare Triples

{ ??? } z := z + 1 { z <= n }

{ a > b } a := a – b { ??? }

Rule of Consequence

$$\frac{\{ P \}\ S\ \{ Q \} \qquad P' \Rightarrow P \qquad Q \Rightarrow Q'}{\{ P' \}\ S\ \{ Q' \}}$$

- The postcondition can always be *weakened* and the precondition can always be *strengthened*.

- Example:

$$\frac{\{ x > 3 \}\ x := x\text{-}3\ \{ x > 0 \} \qquad x > 5 \Rightarrow x > 3 \qquad x > 0 \Rightarrow x > 0}{\{ x > 5 \}\ x := x\text{-}3\ \{ x > 0 \}}$$

Sequences

$$\frac{\{ P_1 \}\ S_1\ \{ P_2 \} \qquad\qquad \{ P_2 \}\ S_2\ \{ P_3 \}}{\{ P_1 \}\ S_1\ ;\ S_2\ \{ P_3 \}}$$

- Example: Find the weakest precondition.

{ ? } y := 3*x+1 ; x := y+3 { x < 10 }

$$\frac{\{(3*x+1)+3<10\}\ y:=3*x+1\ \{y+3<10\} \qquad\qquad \{y+3<10\}\ x:=y+3\ \{x<10\}}{\{\ (3*x+1)+3<10\ \}\ y := 3*x+1\ ;\ x := y+3\ \{\ x < 10\ \}}$$

- Equivalently:

$$\frac{\{x<2\}\ y:=3*x+1\ \{y<7\} \qquad\qquad \{y<7\}\ x:=y+3\ \{x<10\}}{\{\ x<2\ \}\ y := 3*x+1\ ;\ x := y+3\ \{\ x < 10\ \}}$$

The Conditional Statement Rule

$$\frac{\{\ B \text{ and } P\ \}\ S_1\ \{\ Q\ \} \qquad \{\ (\text{not } B) \text{ and } P\ \}\ S_2\ \{\ Q\ \}}{\{\ P\ \}\ \text{if } B \text{ then } S_1 \text{ else } S_2\ \{\ Q\ \}}$$

- Given an "if" statement and a postcondition, a strategy for finding a precondition and a proof is:
  - Find preconditions P_1 and P_2 for the "then" and "else" branches (using Q as the postcondition):

    $\{\ P_1\ \}\ S_1\ \{\ Q\ \}$ $\qquad$ $\{\ P_2\ \}\ S_2\ \{\ Q\ \}$

  - Find an assertion P such that (B and P) $\Rightarrow P_1$ and ((not B) and P) $\Rightarrow P_2$. For example, use the weakest precondition:

    $P \equiv (B \Rightarrow P_1)$ and $((\text{not } B) \Rightarrow P_2)$

Example

Complete the proof with the Assignment and Consequence rules

:

Complete the proof with the Assignment and Consequence rules

:

$$\frac{\{\ x>0 \text{ and } P\ \}\ y:=y-1\ \{\ y>0\ \} \qquad \{\ (\text{not } (x>0)) \text{ and } P\ \}\ y:=y+1\ \{\ y>0\ \}}{\{\ P\ \}\ \text{if } x>0 \text{ then } y:=y-1 \text{ else } y:=y+1\ \{\ y > 0\ \}}$$

One solution is the weakest precondition:

$P \equiv (x>0 \Rightarrow y>1) \text{ and } ((\text{not}(x>0)) \Rightarrow y > -1)$

Another solution is y>1.

The Complete Proof

Let $P_w \equiv (x>0 \Rightarrow y>1) \text{ and } ((\text{not}(x>0)) \Rightarrow y > -1)$

1. $\{ y > 1 \}\ y := y-1\ \{ y > 0 \}$ — by Assignment Rule
2. $(x > 0 \text{ and } P_w) \Rightarrow (y > 1)$ — by arithmetic/logic
3. $(y > 0) \Rightarrow (y > 0)$ — by logic
4. $\{x > 0 \text{ and } P_w \}\ y := y-1\ \{ y > 0 \}$ — by Consequence Rule (from 1,2,3)
5. $\{ y > -1 \}\ y := y+1\ \{ y > 0\}$ — by Assignment Rule
6. $(\text{not } (x > 0)) \text{ and } P_w \Rightarrow y > -1$ — by arithmetic/logic
7. $\{ (\text{not } (x > 0)) \text{ and } P_w \}\ y := y+1\ \{ y > 0 \}$ — by Consequence Rule (from 5,6,3)
8. $\{ P_w \}$ if $x>0$ then $y:=y-1$ else $y:=y+1$ $\{ y > 0 \}$ — by If Rule (from 4,7)

The While Rule

$$\frac{\{\ I \text{ and } B\ \}\ S\ \{\ I\ \}}{\{\ I\ \}\ \text{while } B \text{ do } S \text{ end}\ \{\ I \text{ and (not } B)\ \}}$$

- I is a *loop invariant*. It expresses a relationship between the values of the variables in the loop. The values change, but the relationship stays the same.
- Let Q be the postcondition of the while loop. Let P be the weakest precondition of the loop body S. Then the proof usually has the form:

$$\frac{\dfrac{\dfrac{\{\ P\}\ S\ \{\ I\ \} \qquad (I \text{ and } B) \Rightarrow P \qquad \dfrac{}{I \Rightarrow I}}{\{\ I \text{ and } B\ \}\ S\ \{\ I\ \}}}{\{\ I\ \}\ \text{while } B \text{ do } S \text{ end}\ \{\ I \text{ and (not } B)\ \}} \qquad \dfrac{}{I \Rightarrow I} \qquad I \text{ and (not } B) \Rightarrow Q}{\{\ I\ \}\ \text{while } B \text{ do } S \text{ end}\ \{\ Q\ \}}$$

Example: Factorial

$\{ n >= 0 \}$

S_1 : count := 0;

S_2 : fact := 1;

S_3 : while count <> n do

  S_4 : count := count+1;

  S_5 : fact := fact * count

  end

$\{ fact = n! \}$

$$
\begin{array}{l}
\{ P_1 \}\ S_1\ \{ P_2 \} \quad n >= 0 \Rightarrow P_1 \quad P_2 \Rightarrow P_2 \\
\hline
\{ n >= 0 \}\ S_1\ \{ P_2 \} \qquad\qquad\qquad \{ P_2 \}\ S_2\ \{ I \} \\
\hline
\qquad \{ P_1 \}\ S_1\ ;\ S_2\ \{ I \} \qquad\qquad\qquad \{ I \}\ S_3\ \{ fact = n! \} \\
\hline
\qquad\qquad \{ n >=0 \}\ S_1\ ;\ S_2\ ;\ S_3\ \{ fact = n! \}
\end{array}
$$

Proving Correctness of Factorial

$$\dfrac{\dfrac{\dfrac{\{\, P_1 \,\}\ S_1\ \{\, P_2 \,\} \quad n >= 0 \Rightarrow P_1 \quad P_2 \Rightarrow P_2}{\{\, n >= 0 \,\}\ S_1\ \{\, P_2 \,\}} \qquad \{\, P_2 \,\}\ S_2\ \{\, I \,\}}{\{\, P_1 \,\}\ S_1\ ;\ S_2\ \{\, I \,\}} \qquad \{\, I \,\}\ S_3\ \{\, fact = n! \,\}}{\{\, n >= 0 \,\}\ S_1\ ;\ S_2\ ;\ S_3\ \{\, fact = n! \,\}}$$

1. Find an invariant I of the while loop S_3.
2. Find the weakest precondition P_2 of $\{ P_2 \}\ S_2\ \{ I \}$ by applying the assignment rule.
3. Find the weakest precondition P_1 of $\{ P_1 \}\ S_1\ \{ P_2 \}$ by applying the assignment rule.
4. Make sure that $n >= 0 \Rightarrow P_1$.

Proving { I } S_3 { fact = n! }

$$
\dfrac{
  \dfrac{
    \dfrac{\{ P_4 \}\ S_4\ \{ P_5 \} \qquad (I \text{ and } B) \Rightarrow P_4 \qquad \overline{P_5 \Rightarrow P_5}}
          {\{ I \text{ and } B \}\ S_4\ \{ P_5 \}}
    \qquad \{ P_5 \}\ S_5\ \{ I \}
  }{\{ I \text{ and } B \}\ S_4 ;\ S_5\ \{ I \}}
}{\{ I \}\ \text{while } B \text{ do } S_4 ;\ S_5 \text{ end}\ \{ I \text{ and } (\text{not } B) \}}
$$

$$
\dfrac{\{ I \}\ \text{while } B \text{ do } S_4 ;\ S_5 \text{ end}\ \{ I \text{ and } (\text{not } B) \} \qquad \overline{I \Rightarrow I} \qquad (I \text{ and } (\text{not } B)) \Rightarrow \text{fact} = n!}
{\{ I \}\ S_3\text{: while } B \text{ do } S_4 ;\ S_5 \text{ end}\ \{ \text{fact} = n! \}}
$$

Note that B is (count <> n).

5. Find the weakest precondition P_5 in { P_5 } S_5 { I } by applying the assignment rule.

6. Find the weakest precondition P_4 in { P_4 } S_4 { P_5 } by applying the assignment rule.

7. Make sure that I is strong enough to prove (I and (not (count <> n))) $\Rightarrow$ fact = n!

8. Make sure that I is strong enough to prove (I and count <> n) $\Rightarrow$ P_4

The Complete Proof (1)

1. { 1 = 0! } count := 0 {1 = count! } by Assignment Rule
2. (n >= 0) ⇒ (1 = 0!) by arithmetic/logic
3. (1 = count!) ⇒ (1 = count!) by logic
4. { n >= 0 } count := 0 { 1 = count! } by Consequence Rule (from 1,2,3)
5. { 1 = count! } fact := 1 { fact = count! } by Assignment Rule
6. { n >= 0 } S_1 ; S_2 { fact = count! } by Sequence Rule (from 4,5)
7. { fact*(count+1)=(count+1)! } count := count + 1 { fact*count=count! }

   by Assignment Rule
8. (fact = count!) and (count <> n) ⇒ (fact*(count+1)=(count+1)!)

   by arithmetic/logic
9. (fact*count=count!) ⇒ (fact*count=count!) by logic
10. {(fact = count!) and (count <> n) } count := count + 1 { fact*count=count! }

    by Consequence Rule (from 7,8.9)

The Complete Proof (2)

11. { fact*count = count! } fact := fact * count { fact = count!}

by Assignment Rule

12. { (fact = count!) and (count <> n) } S_4 ; S_5 { fact = count! }

by Sequence Rule (from 10,11)

13. { (fact = count!) } while B do S_4 ; S_5 end

{ (fact = count!) and (not (count <> n)) }

by While Rule (from 12)

14. (fact = count!) $\Rightarrow$ (fact = count!) by logic

15. (fact = count!) and (not (count <> n)) $\Rightarrow$ fact = n!

by logic/arithmetic

16. { (fact = count!) } while B do S_4 ; S_5 end { fact = n!) }

by Consequence rule (from 13,14,15)

17. { n >=0 } S_1 ; S_2 ; S_3 { fact = n! } by Sequence Rule (from 6, 16)

Infinite Loops and Hoare Triples

~~{ n >= 0 }~~ { true }

count := 0;

fact := 1;

while count <> n do

    count := count+1;

    fact := fact * count

end

{ fact = n! }

- Note that if we replace the precondition with “true”, this Hoare triple is provable also.
- But if n < 0, the loop never terminates.
- The meaning of { P } S { Q } is that whenever P holds before execution, if S terminates, then Q holds. This is called partial correctness.

Proving Termination

```
{ n >= 0 }
count := 0;
fact := 1;
while count <> n do
        count := count+1;
        fact := fact * count
end
{ fact = n! }
```


- If the precondition holds, then the loop always terminates, because the value of the expression (n-count) goes down to 0.
- We can add termination to proofs of Hoare triples, but that is beyond the scope of this chapter.
- The meaning of { P } S { Q } becomes: whenever P holds before execution, then Q holds after, and S always terminates. This is called total correctness.