Section 8.7
Loop Invariants

Interpreting Predicate Logic Statements

- Consider the following statement in predicate logic where the domain of discourse is the natural numbers:

$$(x = y) \lor (x + 1 = y)$$

- What is needed in order to determine the truth value of the statement?
- We need the values of each variable that occurs in the statement

Environment Functions

- In order to know the values of variables, we use a function that takes variable names and returns values in our domain of discourse, the natural numbers, $\boldsymbol{N}$. If V is the set of variables, then the function

$$\eta: V \rightarrow \boldsymbol{N}$$

So if $\eta(x) = 3$, then the variable x has the value 3

Such functions that map variables to values in the domain of discourse are called <u>environments</u>

Interpreting $(x = y) \vee (x + 1 = y)$

- In order to interpret $(x = y) \vee (x + 1 = y)$, we need an environment
- Suppose that
  - $\eta(x) = 3$
  - $\eta(y) = 4$
- Then the $(x = y) \vee (x + 1 = y)$ when evaluated with η is true

Interpreting $(x = y) \vee (x + 1 = y)$

- However, if
  - $\eta_2(x) = 3$
  - $\eta_2\ (y) = 0$
- Then the $(x = y) \vee (x + 1 = y)$ when evaluated with $\eta_2$is false

Program State

- A similar concept applies to computer programs
- When we hand trace a program, we write down the values of variables that are in computer memory

```
x := 1
y := 5
x := y * 10
```

x	y

Program State

- A similar concept applies to computer programs
- When we hand trace a program, we write down the values of variables that are in computer memory

```
x := 1
y := 5
x := y * 10
```

x	y
1	

Program State

- A similar concept applies to computer programs
- When we hand trace a program, we write down the values of variables that are in computer memory

```
x := 1
y := 5
x := y * 10
```

x	y
1	5

Program State

- A similar concept applies to computer programs
- When we hand trace a program, we write down the values of variables that are in computer memory

```
x := 1
y := 5
x := y * 10
```

x	y
~~1~~	5
50	

Program State and Environments

- The computer memory used by a program is referred to as the program's state.
- The environment function η and the computer memory symbolized by the table created when we create a hand-trace fill the same role: they store variable values

Programs as State Transformers

- We can think of a program or a program fragment as something that transforms its state

`x := y * 10` transforms

x	y
1	5

into

x	y
50	5

The program state before executing `x := y * 10`

The program state after executing `x := y * 10`

Programs as State Transformers

- Since environment functions and program state serve the same role, we can also think of a program or even a single program statement as transforming one environment into another

`x := y * 10` transforms η_1 into η_2

$\eta_1(x) = 1$
$\eta_1(y) = 5$

$\eta_2(x) = 50$
$\eta_2(y) = 5$

Program Verification

- Let p and q be statements in predicate logic and let S be a program, then S is partially correct with respect to pre-condition p and post-condition q when:

  For any environment η_1 in which p is true:

  If S transforms η_1 to η_2 then q is true in η_2

  $p\{S\}q$ denotes that S is partially correct with respect to p and q

  $p\{S\}q$ is called a partial correctness assertion

Program Verification

- Note that $p\{S\}q$ does not require S to terminate when started with η_1. It only requires that if S does terminate when started with a η_1 that makes p true, then the resulting η_2 makes q true

Program Verification

- Example:

$$x = 1 \{\texttt{x = x+1}\} x = 2$$

is a true partial correctness assertion

Program Verification

- Example:

$$y = 3 \{\texttt{x = 2*y}\} x = 6$$

  is a true partial correctness assertion

Building Programs

- Every assignment statement is a program.
- Larger programs can be built from smaller programs in 3 ways

Building Programs

1. Sequencing: If S_1 and S_2 are programs, then $S_1\ S_2$ is a program

Example: Since `x := 0` and `y := 1` are each programs, then

```
x := 0    y := 1
```

is a program

Building Programs

2. Conditional Statements: If S is a program and $condition$ is a program test, then

```
if condition then
  S
end-if
```

is a program

Building Programs

2. Conditional Statements example:

```
if x > 0 then
  x := x+1
end-if
```

is a program

Building Programs

3. While loop: If S is a program and $condition$ is a program test, then

   while $condition$
     S
   end-while

   is a program

Building Programs

3. While Loop example:

```
while x > 0
  y := y + x
  x := x - 1
end-while
```

is a program

Rules of Inference

- For each type of program, there is a rule that guides us in creating partial correctness assertions from simpler partial correctness assertions

Rules of Inference

1. Sequencing

$$\frac{p\ \{S_1\}\ q \qquad q\ \{S_2\}\ r}{p\ \{S_1\ \ S_2\}\ r}$$

If $p\ \{S_1\}\ q$ and $q\ \{S_2\}\ r$ are true partial correctness assertions, then $p\ \{S_1\ \ S_2\}\ r$ is a true partial correctness assertion

Rules of Inference

1. Sequencing example:

$$\frac{y = 2\ \{\texttt{x = y+1}\}\ x = 3 \qquad x = 3\ \{\texttt{y = x+1}\}\ y = 4}{y = 2\ \{\texttt{x = y+1} \quad \texttt{y = x+1}\}\ y = 4}$$

Rules of Inference

2. Conditional Statement

$$\frac{p \land condition\ \{S_1\}\ q \qquad (p \land \neg condition) \to q}{p\ \{\texttt{if}\ condition\ \texttt{then}\ S_1\ \texttt{end-if}\}\ q}$$

Rules of Inference

2. Conditional Statement

$$\frac{p \land condition\ \{S_1\}\ q \qquad (p \land \neg condition) \to q}{p\ \{\texttt{if}\ condition\ \texttt{then}\ S_1\ \texttt{end-if}\}\ q}$$

If $p \land condition\ \{S_1\}\ q$ is a true partial correctness assertions and $(p \land \neg condition) \to q$ is a true in all environments η, then $p\ \{\texttt{if}\ condition\ \texttt{then}\ S_1\}\ q$ is a true partial correctness assertion

Rules of Inference

2. Conditional Statement example

$$\frac{\mathbf{True} \land x < 0 \; \texttt{\{x = -x\}} \; x \geq 0 \qquad (\mathbf{True} \land \neg x < 0) \rightarrow x \geq 0}{\mathbf{True} \; \{\texttt{if x < 0 then x = -x end-if}\} \; x \geq 0}$$

Rules of Inference

2. Conditional Statement with Else

$$\frac{p \wedge condition\ \{S_1\}\ q \qquad (p \wedge \neg condition)\ \{S_2\}\ q}{p\ \{\texttt{if}\ condition\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \texttt{end-if}\}\ q}$$

Rules of Inference

2. Conditional Statement with Else

$$\frac{p \wedge condition\ \{S_1\}\ q \qquad (p \wedge \neg condition)\ \{S_2\}\ q}{p\ \{\texttt{if}\ condition\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \texttt{end-if}\}\ q}$$

If $p \wedge condition\ \{S_1\}\ q$ and $(p \wedge \neg condition)\ \{S_2\}\ q$ are true partial correctness assertions, then

$p\ \{\texttt{if}\ condition\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \texttt{end-if}\}\ q$ is a true partial correctness assertion

Rules of Inference

3. While Loop

$$\frac{p \wedge condition\ \{S_1\}\ p}{p\ \{\texttt{while}\ condition\ S_1\ \texttt{end-while}\}\ (\neg condition \wedge p)}$$

p is called a loop invariant

Rules of Inference

3. While Loop

$$\frac{p \land condition\ \{S_1\}\ p}{p\ \{\texttt{while}\ condition\ S_1\ \texttt{end-while}\}\ (\neg condition \land p)}$$

If $p \land condition\ \{S_1\}\ p$ is a true partial correctness assertions, then $p\ \{\texttt{while}$ $condition\ S_1\ \texttt{end-while}\}\ (\neg condition \land p)$is a true partial correctness assertion

Rules of Inference

3. While loop example

$$\frac{x + y = z \land \neg x = 0 \ \{\texttt{x=x-1; y=y+1}\} \ x + y = z}{x + y = z \ \{\texttt{while (}\neg\texttt{x=0) x:=x-1 y:=y+1 end-while}\} \ (\neg\neg x = 0 \land x + y = z)}$$

Rules of Inference

3. While loop example

$$\frac{x + y = z \land \neg x = 0 \ \{\texttt{x=x-1; y=y+1}\} \ x + y = z}{x + y = z \ \{\texttt{while}\ (\neg\texttt{x=0})\ \texttt{x:=x-1 y:=y+1 end-while}\} \ (\neg\neg x = 0 \land x + y = z)}$$

What happens if initially $x < 0$?

Loop Invariants

- A first attempt at creating a loop invariant
- Start with a hand trace and examine how the variables change

```
while ¬x=0
   x := x-1
   y := y+1
end-while
```

x	y
3	0
2	1
1	2
0	3

- In general, `x+y` is a constant, i.e. `x+y=c`

Loop Invariants

- Another example

```
x := 0;
i := 0;
while i < a
   x := x + m
   i := i + 1
end-while
```

x	i
0	0
m	1
m+m	2
m+m+m	3
⋮	⋮

  - In general, x = im

Loop Invariants and Mathematical Induction

- Prove $\forall n\ P(n)$ by mathematical induction on the natural numbers where $P(n)$ is

After n iterations of the loop, $x = im$

1. Base case: $n = 0$

   After 0 iterations, $x = 0$ and $i = 0$, hence $x = im$

```
x := 0;
i := 0;
while i < a
  x := x + m
  i := i + 1
end-while
```

Loop Invariants and Mathematical Induction

2. Induction step:

   Let x_k and i_k denote the values of program variables `x` and `i` after k iterations

```
x := 0;
i := 0;
while i < a
  x := x + m
  i := i + 1
end-while
```

Loop Invariants and Mathematical Induction

2. Induction step:

   Let x_k and i_k denote the values of program variables `x` and `i` after k iterations

   1. Assume after k iterations, $x_k = i_k m$

```
x := 0;
i := 0;
while i < a
  x := x + m
  i := i + 1
end-while
```

Loop Invariants and Mathematical Induction

2. Induction step:

   Let x_k and i_k denote the values of program variables `x` and `i` after k iterations

   1. Assume after k iterations, $x_k = i_k m$
   2. After the $k + 1$st iteration, $x_{k+1} = x_k + m$ and $i_{k+1} = i_k + 1$

```
x := 0;
i := 0;
while i < a
  x := x + m
  i := i + 1
end-while
```

Loop Invariants and Mathematical Induction

2. Induction step:

Let x_k and i_k denote the values of program variables `x` and `i` after k iterations

1. Assume after k iterations, $x_k = i_k m$
2. After the $k + 1$st iteration, $x_{k+1} = x_k + m$ and $i_{k+1} = i_k + 1$
3. $x_{k+1} = x_k + m = i_k m + m = (i_k + 1)m = i_{k+1} m$

```
x := 0;
i := 0;
while i < a
  x := x + m
  i := i + 1
end-while
```

Loop Invariants and Mathematical Induction

2. Induction step:

Let x_k and i_k denote the values of program variables `x` and `i` after k iterations

1. Assume after k iterations, $x_k = i_k m$
2. After the $k + 1$st iteration, $x_{k+1} = x_k + m$ and $i_{k+1} = i_k + 1$
3. $x_{k+1} = x_k + m = i_k m + m = (i_k + 1)m = i_{k+1} m$
4. After $k + 1$ iterations $x_{k+1} = i_{k+1} m$

```
x := 0;
i := 0;
while i < a
  x := x + m
  i := i + 1
end-while
```