(Dynamic) Semantics of Programs

- Operational – what the program does
- Denotational – the function computed by the program.
- Axiomatic – what is true after execution

We want to prove theorems of the form:

\{P\} S \{Q\}  

where

\{P\} and \{Q\} are sets of assertions about the program variables.

S is a (possibly compound) statement

S could in fact be a whole program

The theorem \{P\} S \{Q\} says that

If P (the precondition) is true just prior to executing S,

then Q (the postcondition) is true just after executing S.

Normally, the desired result of statement S is expressed by the postcondition Q. So given Q, we need to deduce a precondition P that, when true, will guarantee that Q holds after running S. In particular, we want the weakest precondition.
The assignment statement

It turns out that given postcondition Q for an assignment statement
ident ← exp
the weakest precondition P that gives \{ P \} S \{ Q \}
can be determined immediately by the axiom:

\{ Q[ \text{exp} / \text{ident} ] \} \quad \text{ident} ← \text{exp} \quad \{ Q \}

In other words, P is the assertion we get by substituting the expression on the right of the assignment for the identifier on the left, throughout Q.

Example: Consider the statement \( n ← (2x) + 1 \)
Suppose we want \( n > 1 \) as a postcondition for this.
What is the weakest precondition that guarantees it?

By the above axiom, it is: \( (n > 1)[(2x) + 1 / n] \)
which is \( ((2x) + 1 > 1) \)
\[ = 2x > 0 = x > 0 \]

So we have proved

\{ x > 0 \} \quad n ← (2x) + 1 \quad \{ n > 1 \}
The assignment statement: reasoning forward

Given assignment statement \( \text{ident} \leftarrow \text{exp} \)

if we know a precondition \( P \) holds \textit{prior} to executing the statement, can we figure out what postcondition \( Q \) will hold \textit{after} executing the statement? Answer: SOMETIMES

Recall the assignment axiom:

\[
\{ Q[ \text{exp} / \text{ident} ] \} \quad \text{ident} \leftarrow \text{exp} \quad \{Q\}
\]

and consider again the statement \( n \leftarrow (2\times x)+1 \)

If we know \( \{ x > 0 \} \) holds prior to the assignment, can we try to apply the axiom in reverse?

Look for "\( (2\times x)+1 \)" in \( \{ x > 0 \} \) and replace them by \( n \).

But \( (2\times x)+1 \) doesn’t occur in \( \{ x > 0 \} \) – do some algebra.

\[
\{ x > 0 \} = \{ 2\times x > 0 \} = \{ (2\times x)+1 > 1 \}
\]

Now the replacement produces \( \{ n > 1 \} \)

So we suspect that

\[
\{ x > 0 \} \quad n \leftarrow (2\times x)+1 \quad \{ n > 1 \}
\]

THIS DOESN’T ALWAYS WORK

To be sure we have to check by using the axiom in its proper form:

Start with \( \{n > 1\} \) and derive the precondition \( \{ x > 0 \} \)
Consider the little theorem just proved:

\[
\{ x > 0 \} \; n \leftarrow (2 \cdot x) + 1 \; \{ n > 1 \}
\]

Suppose, prior to executing \(n \leftarrow (2 \cdot x) + 1\),
we knew that \(x > 9\)?
Well, \(x > 9 \implies x > 0\),
so the weakest precondition \(x > 0\) also holds.

Furthermore, suppose after executing \(n \leftarrow (2 \cdot x) + 1\),
we needed only that \(n \geq 0\)?
Well, \(n > 1 \implies n \geq 0\),
and since the postcondition \(n > 1\) holds,
the weaker postcondition \(n \geq 0\) also holds.

As a result, if \(Q\) and \(P\) are, respectively, the postcondition and
weakest precondition for statement \(S\), and if both \(P' \implies P\) and
\(Q \implies Q'\), then we have \(\{P'\} S \{Q'\}\).

Written as a rule of inference:

\[
\frac{\{P\} S \{Q\}, \; P' \implies P, \; Q \implies Q'}{\{P'\} S \{Q'\}}
\]
Sequences of Statements

How do we combine preconditions and postcondition for single statements into preconditions and postconditions for multiple statements and/or whole programs?

Consider a two statement sequence, S1; S2.
Given a postcondition for S2, use the weakest precondition of S2 as the postcondition for S1.

Expressed as a rule of inference:

\[
\begin{align*}
\{P\} \ S1 & \{Q\}, \ \{Q\} \ S2 \ {R} \\
\{P\} \ S1; \ S2 & \{R\}
\end{align*}
\]

Composition Rule

Consider the sequence: \( n \leftarrow 2 \times n; \ i \leftarrow i+1 \)
and the postcondition \( n = 2^i \)

What is the weakest precondition for the sequence?

\[
\begin{align*}
\{n = 2^i[i+1 / i]\} \ i & \leftarrow i+1 \ \{n = 2^i\} \quad \text{(assignment axiom)} \\
\{n = 2^{i+1}\} \ i & \leftarrow i+1 \ \{n = 2^i\} \\
\{n = 2^{i+1}[2 \times n / n]\} \ n & \leftarrow 2 \times n \ \{n = 2^{i+1}\} \quad \text{(assignment axiom)} \\
\{2 \times n = 2^{i+1}\} \ n & \leftarrow 2 \times n \ \{n = 2^{i+1}\} \\
\{2 \times n = 2^{i+1}\} \ n & \leftarrow 2 \times n; \ i \leftarrow i+1 \ \{n = 2^i\} \quad \text{(composition rule)} \\
\{n = 2^i\} \ n & \leftarrow 2 \times n; \ i \leftarrow i+1 \ \{n = 2^i\}
\end{align*}
\]
Given the conditional: \textbf{if B then S}
we would like to discover an
assertions P and Q such that
if P holds prior to the conditional,
then Q holds after the conditional, i.e.,
\{P\} if B then S {Q}
But S will execute exactly when B holds,
so we need to establish
\{P \land B\} S \{Q\}
But if \neg B holds, S \textbf{does not} execute,
so we also need to establish
\{P \land \neg B\} => \{Q\}
The formal rule of inference for conditionals accounts for all these things:

\[
\frac{\{P \land B\} S \{Q\}, \ P \land \neg B => Q}{\{P\} \textbf{ if B then S } \{Q\}}
\]
The extension to "\textbf{if then else}" is obvious:

\[
\frac{\{P \land B\} S_1 \{Q\}, \ \{P \land \neg B\} S_2 \{Q\}}{\{P\} \textbf{ if B then S}_1 \textbf{ else S}_2 \{Q\}}
\]
Given the loop: \textbf{while B do S}

we would like to discover a

\textit{loop invariant} \ P \ such that

\{P\} \ S \ \{P\}

But S will never execute unless B holds.

So if B and P are related, we need

\{P \land B\} \ S \ \{P\}

So if \{P \land B\} holds prior to the loop,

then \{P\} will hold after the loop.

But we know more than just \{P\} after the loop:

Since the loop terminated, B must be false.

So after the loop, \{P \land \neg B\} holds.

The formal rule of inference for while loops accounts for all these things:

\[ \frac{\{P \land B\} \ S \ \{P\}}{\{P\} \ \textbf{while B do S} \ \{P \land \neg B\}} \]
Recall that \[ \sum_{i=1}^{n} i = \frac{n(n+1)}{2} \]

input(n)

s ← 0

i ← 1

while \( i \neq n+1 \) \hspace{1cm} (i\neq n+1 \equiv B) \n
\[
\begin{align*}
  s &\leftarrow s+i \\
  i &\leftarrow i+1
\end{align*}
\]

end while

\[
\begin{align*}
\{ \quad s = \frac{(i(i-1))/2}{P} \quad \land \quad i = n+1 \quad \} \quad \Rightarrow \quad s &= \frac{(n(n+1))/2}{\neg B}
\end{align*}
\]
Recall that \( \sum_{i=1}^{n} i = \frac{n(n+1)}{2} \)

input(n)

\( \{ s = 0 \} [0/s] = \{ 0 = 0 \} = \text{true} \)

\( s \leftarrow 0 \)

\( \{ P[1/i] \} = \{ s = (i(i-1))/2 \} [1/i] = \{ s = (1(1-1))/2 \} = \{ s=0 \} \)

\( i \leftarrow 1 \)

\( \{ s = (i(i-1))/2 \} \equiv P \)

while \( i \neq n+1 \) \hspace{1cm} (i\neq n+1 \equiv B) \)

\( s \leftarrow s+i \)

\( i \leftarrow i+1 \)

end while

\( \{ \frac{s = (i(i-1))/2}{P} \land \frac{i=n+1}{\neg B} \} \Rightarrow \) \( s = \frac{n(n+1)}{2} \)
\{s = 0\}[0/s] = \{0 = 0\} = \text{true}

\{P[1/i]\} = \{s = (i(i-1))/2\}[1/i] = \{s = (1(1-1))/2\} = \{s=0\}

\{s = (i(i-1))/2\} \equiv P
Recall that $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

input(n)

\{s = 0\}[0/s] = \{0 = 0\} = \text{true}

s \leftarrow 0

\{P[1/i]\} = \{s = (i(i-1))/2\}[1/i] = \{s = (1(1-1))/2\} = \{s=0\}

i \leftarrow 1

\{s = (i(i-1))/2\} \equiv P

while $i \neq n+1$

\{P'[s+i/s]\} = \{s+i=(i(i+1))/2\} = \{s+i=(i^2+i)/2\} = \{s=(i(i-1))/2\} \equiv P

\quad s \leftarrow s+i

\{P[i+1/i]\} = \{s = ((i+1)((i+1)-1))/2\} = \{s=(i(i+1))/2\} \equiv P'

\quad i \leftarrow i+1

\{s = (i(i-1))/2\} \equiv P

end while

\{ \frac{s = (i(i-1))/2 \land i=n+1}{\text{P} \quad \neg \text{B}} \} \implies s = (n(n+1))/2
\( P'[s+i/s] \) \( = \) \( s+i=(i(i+1))/2 \) \( = \) \( s+(i^2+i)/2 \) \( = \) \( s=(i(i-1))/2 \) \( \equiv \) \( P \)

\( P[i+1/i] \) \( = \) \( s = ((i+1)((i+1)-1))/2 \) \( = \) \( s=(i(i+1))/2 \) \( \equiv \) \( P' \)

\( s = (i(i-1))/2 \) \( \equiv \) \( P \)