$\{ \wp_{v \leftarrow e} \}$ v := e $\{ \wp \}$

Axiomatic semantics

Points

- The assignment statement 62
- Statement composition 64
- The “if-then-else” statement 66
- The “while” statement 69
- Narrowing and widening 73
- Termination 75
- Two diversions 77
- The greatest common divisor 79
- The “if-then” statement 84

« ‡ ◊ ¤ Δ ¤ ◊ ‡ »

Program verification includes two steps.

1. Associate a formula with every meaningful step of the computation.
2. Show that the final formula logically follows from the initial one through all intermediate steps and formulae.

Axiomatic semantics of assignments, compound statements, conditional statements, iterative statements has been developed by Professor C. A. R. Hoare.

The formulae for assignments and conditions are the elementary building blocks.

The effects of other statements are described by **inference rules** that combine formulae for assignments (just as statements themselves are combinations of assignments and conditions).

The assignment statement

$\wp$ is a formula that contains variable *v*

$\wp_{v \leftarrow e}$ is a formula produced from $\wp$ by replacing all occurrences of variable *v* with expression *e*

☒ $\wp \equiv h \geq 0 \;\&\; h \leq n \;\&\; n > 0$

$\wp_{h \leftarrow 0} \equiv 0 \geq 0 \;\&\; 0 \leq n \;\&\; n > 0$

☒ $\wp \equiv$
$m = \min(1 \leq i \leq k\text{-}1: a_i) \;\&\; k\text{-}1 \neq N$

$\wp_{k \leftarrow k+1} \equiv$
$m = \min(1 \leq i \leq (k+1)\text{-}1: a_i) \;\&\; (k+1)\text{-}1 \neq N$
$\equiv$
$m = \min(1 \leq i \leq k: a_i) \;\&\; k \neq N$

The **axiom** for the assignment statement:

$$\boxed{\{ \wp_{v \leftarrow e} \} \ \texttt{v := e} \ \{ \wp \}}$$

☒ $\{ \underline{\mathbf{0}} \ge 0 \ \& \ \underline{\mathbf{0}} \le n \ \& \ n > 0 \}$

`x := 0`

$\{ \underline{\mathbf{x}} \ge 0 \ \& \ \underline{\mathbf{x}} \le n \ \& \ n > 0 \}$

Two small puzzles

$\{ \text{??????} \}$ `z := z + 1` $\{ z \le N \}$

$\{ a > b \}$ `a := a - b` $\{ \text{??????} \}$

Statement composition

ASSUME THAT
$\{ \wp' \} \ S' \ \{ \wp'' \}$
and $\{ \wp'' \} \ S'' \ \{ \wp''' \}$

CONCLUDE THAT
$\{ \wp' \} \ S' \ S'' \ \{ \wp''' \}$

In other words:
$\{ \wp' \} \ S' \ \{ \wp'' \} \ S'' \ \{ \wp''' \}$

☒ A more complicated example

```
x := 0;  f := 1;
while x ≠ n do begin
    x := x + 1;
    f := f * x;
end;
```

We want to prove that
$\{ f = x! \}$ `x := x + 1; f := f * x;` $\{ f = x! \}$

Let's apply the inference rule for composition.

$\wp'$	is	$f = x!$
$\wp'''$	is	$f = x!$
S'	is	`x := x + 1;`
S''	is	`f := f * x;`

We need to find such a $\wp''$ that we can prove:
$\{ f = x! \}$ `x := x + 1;` $\{ \wp'' \}$
`f := f * x;` $\{ f = x! \}$

Observe that $f = x! \ \equiv \ f = ((x + 1) - 1)!$
and therefore $f = (x - 1)!_{\,x \leftarrow x + 1} \ \equiv \ f = x!$
That is, $\underbrace{\{ f = x! \}}_{\wp'}$ $\underbrace{\texttt{x := x + 1;}}_{S'}$ $\underbrace{\{ f = (x - 1)! \}}_{\wp''}$

Now, $f = (x - 1)! \ \equiv \ f * x = (x - 1)! * x = x!$
so, $f = x!_{\,f \leftarrow f * x} \ \equiv \ f = (x - 1)!$
That is, $\underbrace{\{ f = (x - 1)! \}}_{\wp''}$ $\underbrace{\texttt{f := f * x;}}_{S''}$ $\underbrace{\{ f = x! \}}_{\wp'''}$

QED

The "if-then-else" statement

ASSUME THAT
$\{ \wp \ \& \ \beta \} \ S' \ \{ \Re \}$
and $\{ \wp \ \& \ \neg \beta \} \ S'' \ \{ \Re \}$

CONCLUDE THAT
$\{ \wp \}$ **if** β **then** S' **else** S'' $\{ \Re \}$

Both paths through the if-then-else statement establish the same fact $\Re$, so the whole conditional statement establishes this fact.

☒ The statement
if `a < 0` **then** `b := -a` **else** `b := a`
makes the formula **b = abs(a)** true.

Specifically, the following fact holds:

{ true}
if `a < 0` **then** `b := -a` **else** `b := a`
{ b = abs(a) }

Here, $\wp$ is true
$\Re$ is b = abs(a)
β is a < 0
Also, S′ is `b := -a`
S′′ is `b := a`

We will consider cases. First, we assume that β is true:

true & a < 0 $\equiv$ a < 0 $\Rightarrow$ -a = abs(a)

Therefore, by the assignment axiom:

{ -a = abs(a) } `b := -a` { b = abs(a) }

Similarly, when we assume $\neg\beta$, we get this:

true & $\neg$ a < 0 $\equiv$ a $\geq$ 0 $\equiv$ a = abs(a)

Therefore:

{ a = abs(a) } `b := a` { b = abs(a) }

This shows that both S′ and S′′ establish the same condition:

b = abs(a)

Our fact has been proven:

{ true}
if a < 0 **then** b := -a **else** b := a
{ b = abs(a) }

In other words,
our conditional statement computes abs(a).
It does it without any preconditions: "true" means that there are no restrictions on the initial values of a and b.

The "while" statement

A loop invariant is a condition true immediately before entering the loop, maintained during its execution, and true after the loop has terminated.

ASSUME THAT
{ $\wp$ & β } S { $\wp$ }
[That is, S preserves $\wp$.]

CONCLUDE THAT
{ $\wp$ } **while** β **do** S { $\wp$ & $\neg\beta$ }
provided that the loop terminates

☒ The factorial again...

```
x := 0;  f := 1;
while x ≠ n do begin
   x := x + 1;
   f := f * x;
end;
```


After executing

`x := 0;  f := 1;`

we have f = x! — because actually 1 = 0!

We have shown earlier that

{ f = x! } `x := x + 1; f := f * x;` { f = x! }

Now, $\wp$ is f = x!
β is x $\neq$ n
$\neg\beta$ is x = n

Using the inference rule for **while** loops:

{ f = x! }
```
while x ≠ n do begin
   x := x + 1;
   f := f * x;
end;
```
{ f = x! & x = n}

Notice that $f = x!$ & $x = n \Rightarrow f = n!$

This means the following:

- { true } `x := 0;  f := 1` { $f = x!$ }
- { $f = x!$ } **while** `x` $\neq$ `n` **do begin**

```
         x := x + 1;  f := f * x;
      end
```

{ $f = n!$ }

In other words, the program establishes $f = n!$ without any preconditions on the initial values of f and n.

The axiom for statement composition gives us:

```
{ true } x := 0;  f := 1;
      while x ≠ n do begin
         x := x + 1;  f := f * x;
      end
{ f = n! }
```

Yes! This program computes the factorial of n.

Our reasoning agrees with the intuition of loop invariants: adjusting the relevant variables may make the invariant temporarily <u>false</u>, but we re-establish it by adjusting some other variables.

☒ { $f = x!$ } `x := x + 1` { $f = (x - 1)!$ }

the invariant is "almost true"

{ $f = (x - 1)!$ } `f := f * x` { $f = x!$ }

the invariant is back to normal

This reasoning is <u>not valid</u> for infinite loops: the terminating condition $\wp$ & $\neg \beta$ is never reached, and we know nothing the situation following the loop.

Narrowing and widening

<u>ASSUME THAT</u>

$\wp' \Rightarrow \wp$

and { $\wp$ } S { $\Re$ }

<u>CONCLUDE THAT</u>

{ $\wp'$ } S { $\Re$ }

<u>ASSUME THAT</u>

{ $\wp$ } S { $\Re$ }

and $\Re \Rightarrow \Re'$

<u>CONCLUDE THAT</u>

{ $\wp$ } S { $\Re'$ }

These rules can be used to <u>narrow</u> a precondition, or to <u>widen</u> a postcondition.

☒ **n!** is computed with **true** as the precondition (it is <u>always</u> computed successfully); so **n!** will be computed successfully if initially $n = 5$.

A larger example (in a more concise notation):

{ $N \geq 1$ } $\Rightarrow$

{ $N \geq 1$ & $1 = 1$ & $a_1 = a_1$ }

`i := 1;  s := a1`

{ $N \geq 1$ & $i = 1$ & $s = a_1$ } $\Rightarrow$

{ $N \geq 1$ & $s = a_1 + ... + a_i$ } $\longleftarrow$ <u>INVARIANT</u>

while `i` $\neq$ `N` **do begin**

{ $N \geq 1$ & $s = a_1 + ... + a_i$ & $i \neq N$ }

`i := i + 1;`

{ $N \geq 1$ & $s = a_1 + ... + a_{i-1}$ & $i-1 \neq N$ }

`s := s + ai`

{ $N \geq 1$ & $s = a_1 + ... + a_i$ }

end;

{ $N \geq 1$ & $s = a_1 + ... + a_i$ & $i = N$ } $\Rightarrow$

{ $N \geq 1$ & $s = a_1 + ... + a_N$ }

We have shown that this program computes the sum of $a_1, ..., a_N$.

$N \geq 1$ is only necessary to prove termination.

Termination

Proofs like this only show partial correctness:
- everything is fine if the loop stops,
- otherwise we don't know (but the program may well be correct for a large class of other data).

A reliable proof must show that all loops in the program are finite.

We can prove termination by showing how each step brings us closer to the final condition.

☒ Once again, the factorial.
Initially, x = 0.
Every step increases x by 1, so we go through the numbers 0, 1, 2, ...
$n \geq 0$ must be found among these numbers.
Notice that this reasoning will not work for $n < 0$: the program loops.

A loop terminates when the value of some function of program variables goes down to 0. For the factorial program, such a function could be **n-x**. Its value starts at **n** and decreases by 1 at every step. For summation, take **N-i**.

☒ Multiplication by successive additions.

{ B ≥ 0 & B = B & 0 = 0} ← FOR TERMINATION

```
b := B;  p := 0;
```

{ b = B & p = 0 } ⇒ { p = A * (B - b) }
↑ INVARIANT

```
while  b ≠ 0  do begin
  p := p + A;
```

{ p = A * (B - (b - 1)) }

```
  b := b - 1
```

{ p = A * (B - b) }

```
end;
```

{ p = A * (B - b) & b = 0} ⇒
{ p = A * B }

The loop terminates, because the value of **b** goes down to zero.

Two diversions

☒ Prove that the sequence

```
p := a;
a := b;
b := p;
```

exchanges the values of `a` and `b`:

{ a = A & b = B }

```
p := a;
a := b;
b := p;
```

{ b = A & a = B }

The highlights of a proof:

{ a = A & b = B (& p = P) } `p := a;`
{ p = A & b = B (& a = A) } `a := b;`
{ p = A & a = B (& b = B) } `b := p;`
{ b = A & a = B (& p = A) }

☒ Discover and PROVE the behaviour of the following sequence of statements:

```
x := x + y;
y := x - y;
x := x - y;
```

{ x = X & y = Y } ⇒
{ x + y = X + Y & y = Y }   `x := x + y;`

{ x = X + Y & y = Y } ⇒
{ x = X + Y & x - y = X }   `y := x - y;`

{ x = X + Y & y = X } ⇒
{ x - y = Y & y = X }   `x := x - y;`

{ x = Y & y = X }

The greatest common divisor

$\{ X > 0 \ \& \ Y > 0 \}$
```
a := X;   b := Y;
```
$\{ \Im \}$ ⟵ what should be the invariant?

while a ≠ b $\{ \Im \ \& \ a \neq b \}$ **do**
  if a > b **then** $\{ \Im \ \& \ a \neq b \ \& \ a > b \}$
    `a := a - b`
  else $\{ \Im \ \& \ a \neq b \ \& \ \neg (a > b) \}$
    `b := b - a`

$\{ \Im \ \& \ \neg (a \neq b) \}$

$\{ a = GCD(X, Y) \}$

We will need only a few properties of greatest common divisors:

$GCD(n, n) = n$
$GCD(n + m, m) = GCD(n, m)$
$GCD(n, m + n) = GCD(n, m)$

The first step (**very** formally):

$\{ X > 0 \ \& \ Y > 0 \} \Rightarrow$
$\{ X > 0 \ \& \ Y > 0 \ \& \ X = X \ \& \ Y = Y \}$
```
a := X;  b := Y
```
$\{ a > 0 \ \& \ b > 0 \ \& \ a = X \ \& \ b = Y \}$

We want to get $GCD(X, Y) = a = GCD(a, a)$, when the loop stops with $a = b$:

$GCD(X, Y) = GCD(a, b) \ \& \ a = b$

The invariant $\Im$ could include this equality:

$GCD(X, Y) = GCD(a, b)$

At the beginning of the loop, we have:

$\{ a > 0 \ \& \ b > 0 \ \& \ a = X \ \& \ b = Y \} \Rightarrow$
$\{ a > 0 \ \& \ b > 0 \ \& \ GCD(X, Y) = GCD(a, b) \}$

That is, the invariant could be this:

$a > 0 \ \& \ b > 0 \ \& \ GCD(X, Y) = GCD(a, b)$

We should be able to prove that

$\{a > 0 \ \& \ b > 0 \ \& \ GCD(X, Y) = GCD(a, b) \ \& \ a \neq b\}$
  while ......
$\{a > 0 \ \& \ b > 0 \ \& \ GCD(X, Y) = GCD(a, b)\}$

The final condition will be

$a > 0 \ \& \ b > 0 \ \& \ GCD(X, Y) = GCD(a, b) \ \& \ a = b$

and this will imply $GCD(X, Y) = a$

The loop consists of one conditional statement.

Our proof will be complete if we show that

$\{a > 0 \ \& \ b > 0 \ \& \ GCD(X, Y) = GCD(a, b) \ \& \ a \neq b\}$
```
  if  a > b  then a := a - b
           else b := b - a
```
$\{a > 0 \ \& \ b > 0 \ \& \ GCD(X, Y) = GCD(a, b)\}$

Consider first the case of a > b.

$\{a > 0 \ \& \ b > 0 \ \& \ GCD(X, Y) = GCD(a, b) \ \& \ a \neq b \ \& \ a > b \} \Rightarrow$
$\{a\text{-}b > 0 \ \& \ b > 0 \ \& \ GCD(X, Y) = GCD(a\text{-}b, b)\}$
```
a := a - b
```
$\{a > 0 \ \& \ b > 0 \ \& \ GCD(X, Y) = GCD(a, b)\}$

Now, the case of ¬ a > b.

$\{a > 0 \ \& \ b > 0 \ \& \ GCD(X, Y) = GCD(a, b) \ \& \ a \neq b \ \& \ \neg (a > b) \} \Rightarrow$
$\{a > 0 \ \& \ b\text{-}a > 0 \ \& \ GCD(X, Y) = GCD(a, b\text{-}a)\}$
```
b := b - a
```
$\{a > 0 \ \& \ b > 0 \ \& \ GCD(X, Y) = GCD(a, b)\}$

Both branches of the loop give the same final condition. We will complete the correctness proof when must show that the loop terminates.

We will show that the value of

max(a, b)

decreases at each turn of the loop.

Let a = A, b = B at the beginning of a step.
Assume first that a > b:

max(a, b) = A,
a - b < A, b < A,

therefore max(a - b, b) < A.

Now assume that a < b:

max(a, b) = B,
b - a < B, a < B,

therefore max(a, b - a) < B.

Since a > 0 and b > 0, max(a, b) > 0, and this means that decreasing it cannot go forever.

The "if-then" statement

ASSUME THAT

$\{ \wp \ \& \ \beta \} \ S \ \{ \Re \}$

and $\wp \ \& \ \neg\beta \Rightarrow \Re$

CONCLUDE THAT

$\{ \wp \} \ \textbf{if } \beta \textbf{ then } S \ \{ \Re \}$

We can show that

{ N > 0 }

```
k := 1;   m := a1;
while k ≠ N do begin
  k := k + 1;
  if ak < m then m := ak;
end
```

{ m = min($1 \le i \le N$: a_i) }

Termination is obvious:
the value of **N - k** goes down to zero.

Here is a good invariant: at the k-th turn of the loop, when we have already looked at a_1, ..., a_k, we know that m = min($1 \le i \le k$: a_i).

{ N > 0 } `k := 1; m := a1;`
{ k = 1 & m = a_1 } $\Rightarrow$
{ k = 1 & m = min($1 \le i \le k$: a_i) }

We must prove that

{ m = min($1 \le i \le k$: a_i) & $k \neq N$ }

```
k := k + 1;
if ak < m then m := ak;
```

{ m = min($1 \le i \le k$: a_i) }

{ m = min($1 \le i \le k$: a_i) & $k \neq N$ } $\Rightarrow$
{ m = min($1 \le i \le (k+1)-1$: a_i) &
$(k+1)-1 \neq N$ }

```
k := k + 1
```

{ m = min($1 \le i \le k-1$: a_i) & $k-1 \neq N$ }
Note that $k-1 \neq N$ ensures the existence of a_k.

This remains to be shown:

{ m = min($1 \le i \le k-1$: a_i) & $k-1 \neq N$ }

```
if ak < m then m := ak
```

{ m = min($1 \le i \le k$: a_i) }

The fact we will use is this:
min($1 \le i \le k$: a_i) = min2(min($1 \le i \le k-1$: a_i), a_k)

A conditional statement - two cases:

{m = min($1 \le i \le k-1$: a_i) & $k-1 \neq N$ & $\neg(a_k < m)$}
$\Rightarrow$ {m = min2(min($1 \le i \le k-1$: a_i), a_k)} $\Rightarrow$
{m = min($1 \le i \le k$: a_i)}

{m = min($1 \le i \le k-1$: a_i) & $k-1 \neq N$ & $a_k < m$}
$\Rightarrow$
{a_k = min2(min($1 \le i \le k-1$: a_i), a_k)} $\Rightarrow$
{a_k = min($1 \le i \le k$: a_i)}

```
m := ak
```

{m = min($1 \le i \le k$: a_i)}

The body of the loop preserves the condition

$m = \min(\ 1 \le i \le k\colon a_i\ )$

Now, the whole loop works as follows:

$\{\ m = \min(\ 1 \le i \le k\colon a_i\ )\ \}$

```
while k ≠ N do begin
    k := k + 1; if ak < m then ak := m
end;
```

$\{\ m = \min(\ 1 \le i \le k\colon a_i\ )\ \ \&\ \ k = N\ \}\ \Rightarrow$

$\{\ m = \min(\ 1 \le i \le N\colon a_i\ )\ \}$

All in all, we have shown that our program finds the minimum of N numbers, if only N > 0.

$\boxed{\text{QED}}$

Axiomatic semantics—summary

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....