

Logic and Modelling

— Program Logic —

Jörg Endrullis

VU University Amsterdam

Program Logic

Program Sum

Computing the sum of the first x natural numbers.

Program Sum

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

Note that:

$$\begin{aligned}\sum_{i=1}^x i &= 1 + 2 + 3 + \dots + x \\ &= \frac{x(x+1)}{2}\end{aligned}$$

Program Sum

Computing the sum of the first x natural numbers.

Program Sum

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

We would like to express, for example:

- ▶ started with input $x \geq 0$, Sum yields output $\sum_{i=1}^x i$ for z
- ▶ on every input value, Sum terminates

These are **program specifications** (satisfied by Sum).

Simple Core Language (Syntax)

Simple Core Language (Syntax)

Arithmetic expressions:

$$E ::= n \mid x \mid (-E) \mid (E + E) \mid (E - E) \mid (E * E)$$

Boolean expressions:

$$B ::= \text{true} \mid \text{false} \mid (!B) \mid (B \& B) \mid (B \parallel B) \mid (E < E)$$

Commands:

$$C ::= x = E \mid C; C \mid \text{if } B \{C\} \text{ else } \{C\} \mid \text{while } B \{C\}$$

Simple Core Language (Syntax)

Arithmetic expressions:

$$E ::= n \mid x \mid (-E) \mid (E + E) \mid (E - E) \mid (E * E)$$

where

- ▶ n is any integer numeral in $\{\dots, -2, -1, 0, 1, 2, \dots\}$
- ▶ x any variable
- ▶ $*$ indicates multiplication

- ▶ 42
- ▶ y
- ▶ $4 + (x - 3)$
- ▶ $x + (x * (y - (5 + z)))$

Simple Core Language (Syntax)

Boolean expressions:

$$B ::= \text{true} \mid \text{false} \mid (!B) \mid (B \& B) \mid (B \parallel B) \mid (E < E)$$

- ▶ false
- ▶ $x < 0$
- ▶ $(y * y < x)$
- ▶ $(x < 0) \parallel (!(x + 1 < 0))$

Abbreviations

- ▶ $E_1 == E_2$ is short for: $!(E_1 < E_2) \& !(E_2 < E_1)$
- ▶ $E_1 != E_2$ is short for: $!(E_1 == E_2)$

Binding priorities

same as for logical operators in predicate logic

Simple Core Language (Syntax, Operational Meaning)

Commands:

$$C ::= \underbrace{x = E}_{\text{atomic command}} \quad | \quad \underbrace{C;C \quad | \quad \text{if } B\{C\} \text{ else } \{C\} \quad | \quad \text{while } B\{C\}}_{\text{control structures}}$$

Assignment statement $x = E$:

- (a) evaluates integer expression E in current state
- (b) overwrites current value stored in x by that result

Sequential composition $C_1; C_2$:

- (a) begins with executing C_1
- (b) if this does not terminate,
then the run of $C_1; C_2$ does not terminate
- (c) otherwise the run continues by executing C_2
in the (storage) state resulting from the execution of C_1

Simple Core Language (Syntax, Operational Meaning)

Commands:

$$C ::= \underbrace{x = E}_{\text{atomic command}} \quad | \quad \underbrace{C; C \quad | \quad \text{if } B \{C\} \text{ else } \{C\} \quad | \quad \text{while } B \{C\}}_{\text{control structures}}$$

If-then-else statement `if B { C_1 } else { C_2 }`:

- (a) evaluates boolean expression B in the current state
- (b) if the result is true, then C_1 is executed;
- (c) otherwise C_2 is executed.

While-do statement `while B { C }`:

- (a) boolean expression B is evaluated in the current state
- (b) if B evaluates to false, then the command terminates
- (c) otherwise, command C will be executed.
- (d) if that execution terminates, then we resume at (a)

Programs Fac1, Fac2

The **factorial** n of a natural number n is defined inductively by:

$$0! \stackrel{\text{def}}{=} 1$$

$$(n+1)! \stackrel{\text{def}}{=} (n+1) \cdot n!$$

Program Fac1

```
y = 1;
z = 0;
while (z != x) {
    z = z + 1;
    y = y * z;
}
```

Program Fac2

```
y = 1;
while (x != 0) {
    y = x * y;
    x = x - 1;
}
```

Hoare Triples (Examples)

Factorial computing program `Fac1`

```
y = 1;  
z = 0;  
while (z != x) {  
    z = z + 1;  
    y = y * z;  
}
```

Does `Fac1` always terminate? — **Not for $x < 0$!**

Therefore `Fac1`:

- ▶ cannot (totally) satisfy the specification (**Hoare triple**)

$$\langle \top \rangle P \langle y = x! \rangle$$

- ▶ but it can satisfy:

$$\langle x \geq 0 \rangle P \langle y = x! \rangle$$

Hoare Triples

A **Hoare triple** is a program specification $\langle \phi \rangle P \langle \psi \rangle$ where:

- ▶ ϕ is the **precondition**,
- ▶ P a core program,
- ▶ ψ is the **postcondition**.

Quantifiers in ϕ and ψ **only** bind variables **not in** P !

A **state** (or **store**) of core programs is a function ℓ that to each variable x assigns an integer $\ell(x)$.

The formulas ϕ , ψ are predicate logic formulas over

$$\mathcal{F} = \{ 0/0, 1/0, -/2, +/2, */2 \} \qquad \mathcal{P} = \{ </2, =/2 \}$$

we define for all states ℓ :

$$\ell \models \phi \iff \mathcal{Z} \models_{\ell} \phi$$

where ℓ is viewed as environment, and \mathcal{Z} has domain \mathbb{Z} , and interprets function and predicate symbols in standard manner.

Tony Hoare



Tony Hoare (* 1934)

Evaluation of Conditions in a State

Example

Let ℓ a state with

$$\ell(x) = -2 \qquad \ell(y) = 5 \qquad \ell(z) = -1$$

Then we find:

- ▶ $\ell \models \neg(x + y < z)$ since $(-2) + 5 = 3 \geq (-1)$.
- ▶ $\ell \not\models y - x * z < z$ since $5 - (-2) \cdot (-1) = 3 \geq -1$.
- ▶ $\ell \not\models \forall u (y < u \rightarrow y * z < u * z)$
since: $\ell[u \mapsto 6] \not\models (y < u \rightarrow y * z < u * z)$
because: $5 < 6$, but $5 \cdot (-1) = -5 \geq -6 = 6 \cdot (-1)$.

Partial Correctness and Total Correctness

Definition (\models_{par})

The triple $(\langle \phi \rangle \ P \ \langle \psi \rangle)$ is satisfied under **partial correctness** if for all states ℓ that satisfy the pre-condition ϕ :

- ▶ **if** P terminates when started on ℓ ,
- ▶ **then** the reached state ℓ' satisfies the postcondition ψ .

In this case we write: $\models_{\text{par}} (\langle \phi \rangle \ P \ \langle \psi \rangle)$.

Definition (\models_{tot})

The triple $(\langle \phi \rangle \ P \ \langle \psi \rangle)$ is satisfied under **total correctness** if for all states ℓ that satisfy the pre-condition ϕ :

- ▶ P **terminates** when started on ℓ , and
- ▶ the reached state ℓ' satisfies the postcondition ψ .

In this case we write: $\models_{\text{tot}} (\langle \phi \rangle \ P \ \langle \psi \rangle)$.

Partial Correctness

Definition (\models_{par})

The triple $(\langle \phi \rangle \ P \ \langle \psi \rangle)$ is satisfied under **partial correctness** if for all states ℓ that satisfy the pre-condition ϕ :

- ▶ **if** P terminates when started on ℓ ,
- ▶ **then** the reached state ℓ' satisfies the postcondition ψ .

In this case we write: $\models_{\text{par}} (\langle \phi \rangle \ P \ \langle \psi \rangle)$.

- ▶ $\models_{\text{par}} (\langle x > 0 \rangle \ \text{while } (x > 0) \{x = x - 1\} \ \langle x = 0 \rangle)$
- ▶ $\models_{\text{par}} (\langle x = 0 \rangle \ \text{while } (x > 0) \{x = x - 1\} \ \langle x = 0 \rangle)$
- ▶ $\models_{\text{par}} (\langle x < 0 \rangle \ \text{while } (x > 0) \{x = x - 1\} \ \langle x = 0 \rangle)$

Partial Correctness

Definition (\models_{par})

The triple $(\phi) P (\psi)$ is satisfied under **partial correctness** if for all states ℓ that satisfy the pre-condition ϕ :

- ▶ **if** P terminates when started on ℓ ,
- ▶ **then** the reached state ℓ' satisfies the postcondition ψ .

In this case we write: $\models_{\text{par}} (\phi) P (\psi)$.

- ▶ $\models_{\text{par}} (x > 0) \text{ while } (x > 0) \{x = x - 1\} (x = 0)$
- ▶ $\models_{\text{par}} (x = 0) \text{ while } (x > 0) \{x = x - 1\} (x = 0)$
- ▶ $\not\models_{\text{par}} (x < 0) \text{ while } (x > 0) \{x = x - 1\} (x = 0)$
- ▶ $\models_{\text{par}} (\top) \text{ while } (x > 0) \{x = x - 1\} (x = 0)$

Partial Correctness

Definition (\models_{par})

The triple $(\phi) P (\psi)$ is satisfied under **partial correctness** if for all states ℓ that satisfy the pre-condition ϕ :

- ▶ **if** P terminates when started on ℓ ,
- ▶ **then** the reached state ℓ' satisfies the postcondition ψ .

In this case we write: $\models_{\text{par}} (\phi) P (\psi)$.

- ▶ $\models_{\text{par}} (x > 0) \text{ while } (x > 0) \{x = x - 1\} (x = 0)$
- ▶ $\models_{\text{par}} (x = 0) \text{ while } (x > 0) \{x = x - 1\} (x = 0)$
- ▶ $\not\models_{\text{par}} (x < 0) \text{ while } (x > 0) \{x = x - 1\} (x = 0)$
- ▶ $\not\models_{\text{par}} (\top) \text{ while } (x > 0) \{x = x - 1\} (x = 0)$
- ▶ $\models_{\text{par}} (\top) \text{ while false } \{x = x - 1\} (x = 0)$

Partial Correctness

Definition (\models_{par})

The triple $(\langle \phi \rangle P \langle \psi \rangle)$ is satisfied under **partial correctness** if for all states ℓ that satisfy the pre-condition ϕ :

- ▶ **if** P terminates when started on ℓ ,
- ▶ **then** the reached state ℓ' satisfies the postcondition ψ .

In this case we write: $\models_{\text{par}} (\langle \phi \rangle P \langle \psi \rangle)$.

- ▶ $\models_{\text{par}} (\langle x > 0 \rangle \text{while } (x > 0) \{x = x - 1\} \langle x = 0 \rangle)$
- ▶ $\models_{\text{par}} (\langle x = 0 \rangle \text{while } (x > 0) \{x = x - 1\} \langle x = 0 \rangle)$
- ▶ $\not\models_{\text{par}} (\langle x < 0 \rangle \text{while } (x > 0) \{x = x - 1\} \langle x = 0 \rangle)$
- ▶ $\not\models_{\text{par}} (\langle \top \rangle \text{while } (x > 0) \{x = x - 1\} \langle x = 0 \rangle)$
- ▶ $\not\models_{\text{par}} (\langle \top \rangle \text{while false } \{x = x - 1\} \langle x = 0 \rangle)$
- ▶ $\models_{\text{par}} (\langle \top \rangle \text{while true } \{x = x - 1\} \langle x = 0 \rangle)$
- ▶ $\models_{\text{par}} (\langle \phi \rangle \text{while true } \{x = x - 1\} \langle \psi \rangle)$

Total Correctness

Definition (\models_{tot})

The triple $(\langle \phi \rangle \ P \ \langle \psi \rangle)$ is satisfied under **total correctness** if for all states ℓ that satisfy the pre-condition ϕ :

- ▶ P **terminates** when started on ℓ , and
- ▶ the reached state ℓ' satisfies the postcondition ψ .

In this case we write: $\models_{\text{tot}} (\langle \phi \rangle \ P \ \langle \psi \rangle)$.

- ▶ $\models_{\text{tot}} (\langle x > 0 \rangle \ \text{while } (x > 0) \{x = x - 1\} \ \langle x = 0 \rangle)$
- ▶ $\models_{\text{tot}} (\langle x = 0 \rangle \ \text{while } (x > 0) \{x = x - 1\} \ \langle x = 0 \rangle)$
- ▶ $\models_{\text{tot}} (\langle x < 0 \rangle \ \text{while } (x > 0) \{x = x - 1\} \ \langle x = 0 \rangle)$

Total Correctness

Definition (\models_{tot})

The triple $(\langle \phi \rangle \ P \ \langle \psi \rangle)$ is satisfied under **total correctness** if for all states ℓ that satisfy the pre-condition ϕ :

- ▶ P **terminates** when started on ℓ , and
- ▶ the reached state ℓ' satisfies the postcondition ψ .

In this case we write: $\models_{\text{tot}} (\langle \phi \rangle \ P \ \langle \psi \rangle)$.

- ▶ $\models_{\text{tot}} (\langle x > 0 \rangle \ \text{while } (x > 0) \{x = x - 1\} \ \langle x = 0 \rangle)$
- ▶ $\models_{\text{tot}} (\langle x = 0 \rangle \ \text{while } (x > 0) \{x = x - 1\} \ \langle x = 0 \rangle)$
- ▶ $\not\models_{\text{tot}} (\langle x < 0 \rangle \ \text{while } (x > 0) \{x = x - 1\} \ \langle x = 0 \rangle)$
- ▶ $\models_{\text{tot}} (\langle \top \rangle \ \text{while } (x > 0) \{x = x - 1\} \ \langle x = 0 \rangle)$

Total Correctness

Definition (\models_{tot})

The triple $\langle \phi \rangle P \langle \psi \rangle$ is satisfied under **total correctness** if for all states ℓ that satisfy the pre-condition ϕ :

- ▶ P **terminates** when started on ℓ , and
- ▶ the reached state ℓ' satisfies the postcondition ψ .

In this case we write: $\models_{\text{tot}} \langle \phi \rangle P \langle \psi \rangle$.

- ▶ $\models_{\text{tot}} \langle x > 0 \rangle \text{while } (x > 0) \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\models_{\text{tot}} \langle x = 0 \rangle \text{while } (x > 0) \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\not\models_{\text{tot}} \langle x < 0 \rangle \text{while } (x > 0) \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\not\models_{\text{tot}} \langle \top \rangle \text{while } (x > 0) \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\models_{\text{tot}} \langle \top \rangle \text{while false } \{x = x - 1\} \langle x = 0 \rangle$

Total Correctness

Definition (\models_{tot})

The triple $\langle \phi \rangle P \langle \psi \rangle$ is satisfied under **total correctness** if for all states ℓ that satisfy the pre-condition ϕ :

- ▶ P **terminates** when started on ℓ , and
- ▶ the reached state ℓ' satisfies the postcondition ψ .

In this case we write: $\models_{\text{tot}} \langle \phi \rangle P \langle \psi \rangle$.

- ▶ $\models_{\text{tot}} \langle x > 0 \rangle \text{while } (x > 0) \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\models_{\text{tot}} \langle x = 0 \rangle \text{while } (x > 0) \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\not\models_{\text{tot}} \langle x < 0 \rangle \text{while } (x > 0) \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\not\models_{\text{tot}} \langle \top \rangle \text{while } (x > 0) \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\not\models_{\text{tot}} \langle \top \rangle \text{while false } \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\models_{\text{tot}} \langle \top \rangle \text{while true } \{x = x - 1\} \langle x = 0 \rangle$

Total Correctness

Definition (\models_{tot})

The triple $\langle \phi \rangle P \langle \psi \rangle$ is satisfied under **total correctness** if for all states ℓ that satisfy the pre-condition ϕ :

- ▶ P **terminates** when started on ℓ , and
- ▶ the reached state ℓ' satisfies the postcondition ψ .

In this case we write: $\models_{\text{tot}} \langle \phi \rangle P \langle \psi \rangle$.

- ▶ $\models_{\text{tot}} \langle x > 0 \rangle \text{while } (x > 0) \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\models_{\text{tot}} \langle x = 0 \rangle \text{while } (x > 0) \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\not\models_{\text{tot}} \langle x < 0 \rangle \text{while } (x > 0) \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\not\models_{\text{tot}} \langle \top \rangle \text{while } (x > 0) \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\not\models_{\text{tot}} \langle \top \rangle \text{while false } \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\not\models_{\text{tot}} \langle \top \rangle \text{while true } \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\models_{\text{tot}} \langle \phi \rangle \text{while true } \{x = x - 1\} \langle \psi \rangle$

Total Correctness

Definition (\models_{tot})

The triple $\langle \phi \rangle P \langle \psi \rangle$ is satisfied under **total correctness** if for all states ℓ that satisfy the pre-condition ϕ :

- ▶ P **terminates** when started on ℓ , and
- ▶ the reached state ℓ' satisfies the postcondition ψ .

In this case we write: $\models_{\text{tot}} \langle \phi \rangle P \langle \psi \rangle$.

- ▶ $\models_{\text{tot}} \langle x > 0 \rangle \text{while } (x > 0) \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\models_{\text{tot}} \langle x = 0 \rangle \text{while } (x > 0) \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\not\models_{\text{tot}} \langle x < 0 \rangle \text{while } (x > 0) \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\not\models_{\text{tot}} \langle \top \rangle \text{while } (x > 0) \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\not\models_{\text{tot}} \langle \top \rangle \text{while false } \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\not\models_{\text{tot}} \langle \top \rangle \text{while true } \{x = x - 1\} \langle x = 0 \rangle$
- ▶ $\not\models_{\text{tot}} \langle \phi \rangle \text{while true } \{x = x - 1\} \langle \psi \rangle$

Partial and Total Correctness

A roundabout implementation of the successor function:

Succ

```
a = x + 1;
if (a - 1 == 0) {
    y = 1;
} else {
    y = a;
}
```

We should be able to prove:

- ▶ $\models_{\text{par}} (\top) \text{ Succ } (y = x + 1)$
- ▶ $\models_{\text{tot}} (\top) \text{ Succ } (y = x + 1)$

Partial and Total Correctness

```
Fac1
```

```
y = 1;  
z = 0;  
while (z != x) {  
    z = z + 1;  
    y = y * z;  
}
```

Does not terminate for $x < 0$!

Therefore for total correctness, we expect to be able to prove:

- ▶ $\not\vdash_{\text{tot}} (\top) \text{Fac1 } (|y = x!|)$
- ▶ $\not\vdash_{\text{tot}} (|x \geq 0|) \text{Fac1 } (|y = x!|)$

And for partial correctness we expect:

- ▶ $\vdash_{\text{par}} (|x \geq 0|) \text{Fac1 } (|y = x!|)$
- ▶ $\vdash_{\text{par}} (\top) \text{Fac1 } (|y = x!|)$

Program Variables and Logical Variables

Fac2

```
y = 1;
while (x != 0) {
  y = y * x;
  x = x - 1;
}
```

Note that `Fac2` 'consumes' its input:

- ▶ If $x > 0$ before execution, we have $x = 0$ afterwards.

Although `Fac2` computes the factorial function, we have:

- ▶ $\not\vdash_{\text{par}} (x = n) \text{ Fac2 } (y = x!) \quad (\text{for } n = 2, 3, \dots).$
- ▶ $\vdash_{\text{par}} (x \geq 0) \text{ Fac2 } (y = x!)$

Program Variables and Logical Variables

Fac2

```
y = 1;
while (x != 0) {
  y = y * x;
  x = x - 1;
}
```

Note that `Fac2` 'consumes' its input:

- ▶ If $x > 0$ before execution, we have $x = 0$ afterwards.

Although `Fac2` computes the factorial function, we have:

- ▶ $\not\vdash_{\text{par}} \langle x = n \rangle \text{Fac2} \langle y = x! \rangle$ (for $n = 2, 3, \dots$).
- ▶ $\not\vdash_{\text{par}} \langle x \geq 0 \rangle \text{Fac2} \langle y = x! \rangle$

Remedy: use of a **logical variable** x_0 . Then we expect:

- ▶ $\vdash_{\text{par}} \langle (x \geq 0) \wedge x_0 = x \rangle \text{Fac2} \langle y = x_0! \rangle$
- ▶ $\vdash_{\text{tot}} \langle (x \geq 0) \wedge x_0 = x \rangle \text{Fac2} \langle y = x_0! \rangle$

Program Variables and Logical Variables

`Sum` adds up the first x natural numbers and stores them in z :

`Sum`

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

During execution x is consumed. So while we have e.g.:

- ▶ $\text{F}_{\text{tot}} \langle x = 2 \rangle \text{Sum} \langle z = 3 \rangle$,
- ▶ $\text{F}_{\text{tot}} \langle x = 3 \rangle \text{Sum} \langle z = 6 \rangle$,

we find:

- ▶ ~~$\text{F}_{\text{tot}} \langle x \geq 0 \rangle$~~ $\text{Sum} \langle z = \frac{x(x+1)}{2} \rangle$, (note: $\sum_{i=1}^x i = \frac{x(x+1)}{2}$).

Again, we 'store' the initial value of x in a logical variable x_0 :

- ▶ $\text{F}_{\text{tot}} \langle x \geq 0 \wedge x_0 = x \rangle \text{Sum} \langle z = \frac{x_0(x_0+1)}{2} \rangle$

Program Variables and Logical Variables

Definition

For a Hoare triple $\langle \phi \rangle P \langle \psi \rangle$, its set of **logical variables** are those variables that are **free** in ϕ and ψ , but **do not occur** in P .

Examples

- ▶ $\models_{\text{tot}} \langle (x \geq 0) \wedge x_0 = x \rangle \text{Fac2}(x, y) \langle y = x_0! \rangle$
 - ▶ $\models_{\text{tot}} \langle x \geq 0 \wedge x' = x \rangle \text{Sum}(x, z) \langle z = \frac{x'(x'+1)}{2} \rangle$
- ▶ called **logical**: because occur only in logical formulas
 - ▶ state of program assigns a value to program variables, but not to logical variables

Proof Rules for Partial Correctness

$$\frac{(\phi) C_1 (\eta) \quad (\eta) C_2 (\psi)}{(\phi) C_1; C_2 (\psi)} \text{Composition}$$

$$\frac{}{(\psi[E/x]) x = E (\psi)} \text{Assignment}$$

$$\frac{(\phi \wedge B) C_1 (\psi) \quad (\phi \wedge \neg B) C_2 (\psi)}{(\phi) \text{if } B \{C_1\} \text{else } \{C_2\} (\psi)} \text{If-statement}$$

$$\frac{(\psi \wedge B) C (\psi)}{(\psi) \text{while } B \{C\} (\psi \wedge \neg B)} \text{Partial-while}$$

$$\frac{\vdash_{AR} \phi' \rightarrow \phi \quad (\phi) C (\psi) \quad \vdash_{AR} \psi \rightarrow \psi'}{(\phi') C (\psi')} \text{Implied}$$

Proof Rules for Partial Correctness

$$\frac{(\phi) C_1 (\eta) \quad (\eta) C_2 (\psi)}{(\phi) C_1; C_2 (\psi)} \text{Composition}$$

Convenient to read bottom up (also for the other rules):

In order to prove

$$(\phi) C_1; C_2 (\psi)$$

we need to find an **appropriate midcondition** η , and prove

$$(\phi) C_1 (\eta) \quad \text{and} \quad (\eta) C_2 (\psi)$$

We thereby split the goal into two subgoals:

- ▶ execution of C_1 in a state satisfying ϕ , we get to a state satisfying η
- ▶ execution of C_2 in a state satisfying η , we get to a state satisfying ψ

Proof Rules for Partial Correctness

$$\frac{(\phi \wedge B) \ C_1 \ (\psi) \quad (\phi \wedge \neg B) \ C_2 \ (\psi)}{(\phi) \ \text{if } B \{C_1\} \ \text{else } \{C_2\} \ (\psi)} \text{if-statement}$$

The goal (conclusion) is decomposed into two subgoals:

- ▶ if B evaluates to true, the then-part C_1 is executed
- ▶ if B evaluates to false, the else-part C_2 is executed

In both cases, ψ has to hold after the execution.

Proof Rules for Partial Correctness

$$\frac{}{(\psi[E/x]) \ x = E \ (\psi)} \text{Assignment}$$

For ψ is $x = 5$ we get:

$$(\ 6 = 5 \) \ x = 6 \ (\ x = 5 \)$$

For ψ' is $x = 6$ we get:

$$(\ 6 = 6 \) \ x = 6 \ (\ x = 6 \)$$

The reverse form with conclusion

$$(\ \psi \) \ x = E \ (\ \psi[E/x] \)$$

does not make sense! E.g. for ϕ is $x = 5$ it would entail:

$$(\ x = 5 \) \ x = 6 \ (\ 6 = 5 \) \quad \times$$

Proof Rules for Partial Correctness

$$\frac{}{\langle \psi[E/x] \rangle x = E \langle \psi \rangle} \text{Assignment}$$

- ▶ is best applied backwards
- ▶ **note** the necessity of **capture avoiding** in substitution!

Hint for understanding: if ϕ holds when replacing x by E , then after the assignment $x = E$, the formula ϕ holds.

- ▶ $\langle 2 = y \rangle x = 2 \langle x = y \rangle$
- ▶ $\langle 2 = 4 \rangle x = 2 \langle x = 4 \rangle$
- ▶ $\langle 2 > 2 \rangle x = 2 \langle 2 > x \rangle$
- ▶ $\langle x + 1 = y \rangle x = x + 1 \langle x = y \rangle$
- ▶ $\langle x + 1 + 5 = y \rangle x = x + 1 \langle x + 5 = y \rangle$
- ▶ $\langle x + 1 > 0 \wedge y > 0 \rangle x = x + 1 \langle x > 0 \wedge y > 0 \rangle$

Proof Rules for Partial Correctness

$$\frac{(\psi \wedge B) \ C \ (\psi)}{(\psi) \ \text{while } B \{C\} \ (\psi \wedge \neg B)} \text{ Partial-while}$$

The key part is the **loop invariant** ψ :

- ▶ the body C of the loop usually changes the variables
- ▶ ψ expresses a relationship between values that is preserved by any execution of C
- ▶ the **premise** expresses:
 - ▶ if ψ and B are true before executing the body C , then ψ will again be true afterwards
- ▶ the **conclusion** expresses:
 - ▶ if ψ is true before the execution, then no matter how often the while-loop is executed, ψ will also be true at the end
 - ▶ and since the while-statement only terminates if B does not hold any more, B will be false in the final state

Proof Rules for Partial Correctness

$$\frac{\vdash_{AR} \phi' \rightarrow \phi \quad (\langle \phi \rangle \text{ C } \langle \psi \rangle) \quad \vdash_{AR} \psi \rightarrow \psi'}{(\langle \phi' \rangle \text{ C } \langle \psi' \rangle)} \text{ Implied}$$

$\vdash_{AR} \phi$ is valid \iff

there is a natural deduction proof of ϕ where
standard laws of arithmetic may be used as premises

When **applied top-down**, this rule **allows** to:

- ▶ **strengthen** the precondition (assume more than we need)
- ▶ **weaken** the postcondition (conclude less than we could)

Proof Rules for Partial Correctness

$$\frac{(\phi) C_1 (\eta) \quad (\eta) C_2 (\psi)}{(\phi) C_1; C_2 (\psi)} \text{Composition}$$

$$\frac{}{(\psi[E/x]) x = E (\psi)} \text{Assignment}$$

$$\frac{(\phi \wedge B) C_1 (\psi) \quad (\phi \wedge \neg B) C_2 (\psi)}{(\phi) \text{if } B \{C_1\} \text{else } \{C_2\} (\psi)} \text{If-statement}$$

$$\frac{(\psi \wedge B) C (\psi)}{(\psi) \text{while } B \{C\} (\psi \wedge \neg B)} \text{Partial-while}$$

$$\frac{\vdash_{AR} \phi' \rightarrow \phi \quad (\phi) C (\psi) \quad \vdash_{AR} \psi \rightarrow \psi'}{(\phi') C (\psi')} \text{Implied}$$

$\vdash_{\text{par}} (\top) \text{ while true } \{x = x - 1\} (x = 0)$

$$\frac{\frac{\frac{\vdash_{AR} \top \wedge \top \rightarrow \top \quad \overline{(\top) \ x = x - 1 \ (\top)}}{\text{Assign}}}{(\top \wedge \top) \ x = x - 1 \ (\top)} \text{ Implied}}{\frac{(\top) \ \text{while true } \{x = x - 1\} \ (\top \wedge \neg \top) \quad \vdash_{AR} \top \wedge \neg \top \rightarrow x = 0}{(\top) \ \text{while true } \{x = x - 1\} \ (x = 0)} \text{ Partial-while}} \text{ Implied}$$

Note: We ignored trivial premises $\vdash_{AR} \chi \rightarrow \chi$ of Implied

$$\frac{\vdash_{AR} \phi' \rightarrow \phi \quad (\phi) \ C \ (\psi) \quad \vdash_{AR} \psi \rightarrow \psi'}{(\phi') \ C \ (\psi')} \text{ Implied}$$

$$\frac{(\psi \wedge B) \ C \ (\psi)}{(\psi) \ \text{while } B \ \{C\} \ (\psi \wedge \neg B)} \text{ Partial-while}$$

$$\frac{}{(\psi[E/x]) \ x = E \ (\psi)} \text{ Assignment}$$

$\vdash_{\text{par}} (\langle \phi \rangle \text{ while true } \{x = x - 1\} (\langle \psi \rangle))$

$$\frac{\frac{\frac{\vdash_{AR} \top \wedge \top \rightarrow \top \quad \overline{(\langle \top \rangle x = x - 1 (\langle \top \rangle))}^a}{(\langle \top \wedge \top \rangle x = x - 1 (\langle \top \rangle))}^i}{\vdash_{AR} \phi \rightarrow \top \quad (\langle \top \rangle \text{ while true } \{x = x - 1\} (\langle \top \wedge \neg \top \rangle))}^w \quad \vdash_{AR} \top \wedge \neg \top \rightarrow \psi}{(\langle \phi \rangle \text{ while true } \{x = x - 1\} (\langle \psi \rangle))}^i$$

$$\frac{\vdash_{AR} \phi' \rightarrow \phi \quad (\langle \phi \rangle \text{ C } (\langle \psi \rangle)) \quad \vdash_{AR} \psi \rightarrow \psi'}{(\langle \phi' \rangle \text{ C } (\langle \psi' \rangle))} \text{ Implied}$$

Towards Proving Partial Correctness of Fac2

$$\frac{}{\langle \psi[E/x] \rangle \quad x = E \quad \langle \psi \rangle} \text{Assignment}$$

$$\frac{\vdash_{AR} \phi' \rightarrow \phi \quad \langle \phi \rangle \quad C \quad \langle \psi \rangle \quad \vdash_{AR} \psi \rightarrow \psi'}{\langle \phi' \rangle \quad C \quad \langle \psi' \rangle} \text{Implied}$$

$$\frac{\langle \phi \rangle \quad C_1 \quad \langle \eta \rangle \quad \langle \eta \rangle \quad C_2 \quad \langle \psi \rangle}{\langle \phi \rangle \quad C_1; C_2 \quad \langle \psi \rangle} \text{Composition}$$

$$\frac{\frac{\langle 1 = 1 \rangle \quad y = 1 \quad \langle y = 1 \rangle}{\langle \top \rangle \quad y = 1 \quad \langle y = 1 \rangle} \quad a \quad \frac{\langle y = 1 \wedge 0 = 0 \rangle \quad z = 0 \quad \langle y = 1 \wedge z = 0 \rangle}{\langle y = 1 \rangle \quad z = 0 \quad \langle y = 1 \wedge z = 0 \rangle} \quad a}{\langle \top \rangle \quad y = 1; z = 0 \quad \langle y = 1 \wedge z = 0 \rangle} \quad c$$

