$\llparenthesis \psi[E/x] \rrparenthesis$

`x = ` E

$\llparenthesis \psi \rrparenthesis$

Assignment

1. We show that $\vdash_{\text{par}} (\!|y = 5|\!)$ `x = y + 1` $(\!|x = 6|\!)$ is valid:

$$(\!|y = 5|\!)$$
$$(\!|y + 1 = 6|\!) \qquad \text{Implied}$$
`x = y + 1`
$$(\!|x = 6|\!) \qquad \text{Assignment}$$

The proof is constructed from the bottom upwards. We start with $(\!|x = 6|\!)$ and, using the assignment axiom, we push it upwards through `x = y + 1`. This means substituting $y + 1$ for all occurrences of x, resulting in $(\!|y + 1 = 6|\!)$. Now, we compare this with the given precondition $(\!|y = 5|\!)$. The given precondition and the arithmetic fact $5 + 1 = 6$ imply it, so we have finished the proof.

Although the proof is constructed bottom-up, its justifications make sense when read top-down: the second line is implied by the first and the fourth follows from the second by the intervening assignment.

$(y = 5) \longrightarrow (y + 1 = 6)$

is true

For the sequential composition of assignment statements

P

```
z = x;
z = z + y;
u = z;
```

our goal is to show that u stores the sum of x and y after this s
assignments terminates. Let us write P for the code above. Thus, v
prove $\vdash_{\mathsf{par}} (\!|\top|\!)\ P\ (\!|u = x + y|\!)$.

We construct the proof by starting with the postcondition $u =$
pushing it up through the assignments, in reverse order, using the
rule.

– Pushing it up through `u = z` involves replacing all occurrences
  resulting in $z = x + y$. We thus have the proof fragment

$$(\!|z = x + y|\!)$$

```
u = z;
```

$$(\!|u = x + y|\!) \qquad \text{Assignment}$$

– Pushing $z = x + y$ upwards through `z = z + y` involves replacing z by $z + y$,
  resulting in $z + y = x + y$.

$(\!|\top|\!)$

$(\!|x + y = x + y|\!)$ Implied

```
z = x;
```

$(\!|z + y = x + y|\!)$ Assignment

```
z = z + y;
```

$(\!|z = x + y|\!)$ Assignment

```
u = z;
```

$(\!|u = x + y|\!)$ Assignment

$x = E$

- Consider the example 'proof'

INCORRECT PROOF

$(\!|x + 1 = x + 1|\!)$ $\phi(E/x)$

```
x = x + 1;
```

$(\!|x = x + 1|\!)$ Assignment

$x = E$

ϕ

$E : x + 1$

CORRECT PROOF

$(\!|x + 1 = x + 1 + 1|\!)$

```
x = x + 1;
```

$(\!|x = x + 1|\!)$ Assignment

If-statements. We now consider how to push a postcondition upwards through an if-statement. Suppose we are given a condition ψ and a program fragment $\texttt{if (}B\texttt{) }\{C_1\}\texttt{ else }\{C_2\}$. We wish to calculate the weakest ϕ such that

$$(\!|\phi|\!)\ \texttt{if (}B\texttt{) }\{C_1\}\texttt{ else }\{C_2\}\ (\!|\psi|\!).$$

This ϕ may be calculated as follows.

1. Push ψ upwards through C_1; let's call the result ϕ_1. (Note that, since C_1 may be a sequence of other commands, this will involve appealing to other rules. If C_1 contains another if-statement, then this step will involve a 'recursive call' to the rule for if-statements.)
2. Similarly, push ψ upwards through C_2; call the result ϕ_2.
3. Set ϕ to be $(B \to \phi_1) \wedge (\neg B \to \phi_2)$.

Succ?

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

C_1: y = 1; C_2: y = a;

$(\!|\, y = x+1 \,|\!)$

We want to show that $\vdash_{\mathsf{par}} (\!|\top|\!)\ \mathtt{Succ}\ (\!|y = x + 1|\!)$ is valid. Note that this program is the sequential composition of an assignment and an if-statement. Thus, we need to obtain a suitable midcondition to put between the if-statement and the assignment.

We push the postcondition $y = x + 1$ upwards through the two branches of the if-statement, obtaining

- ϕ_1 is $1 = x + 1$;
- ϕ_2 is $a = x + 1$;

and obtain the midcondition $(a - 1 = 0 \to 1 = x + 1) \wedge (\neg(a - 1 = 0) \to a = x + 1)$ by appealing to a slightly different version of the rule If-statement:

$$\frac{(\!|\phi_1|\!)\ C_1\ (\!|\psi|\!) \qquad (\!|\phi_2|\!)\ C_2\ (\!|\psi|\!)}{(\!|(B \to \phi_1) \wedge (\neg B \to \phi_2)|\!)\ \mathtt{if}\ B\ \{C_1\}\ \mathtt{else}\ \{C_2\}\ (\!|\psi|\!)}\ \text{If-Statement} \quad (4.7)$$

ϕ_1 $(\!|1 = x + 1|\!)$

C_1: $y = 1$

$(\!|y = x + 1|\!)$

ϕ_2 $(\!|a = x + 1|\!)$

C_2: $y = a$

$(\!|y = x + 1|\!)$

$(\!\vert \top \vert\!)$	
$(\!\vert ? \vert\!)$	?
`a = x + 1;`	
$(\!\vert (a-1=0 \to 1=x+1) \wedge (\neg(a-1=0) \to a=x+1) \vert\!)$	?
`if (a - 1 == 0) {`	
$\quad(\!\vert 1=x+1 \vert\!)$	If-Statement
`    y = 1;`	
$\quad(\!\vert y=x+1 \vert\!)$	Assignment
`} else {`	
$\quad(\!\vert a=x+1 \vert\!)$	If-Statement
`    y = a;`	
$\quad(\!\vert y=x+1 \vert\!)$	Assignment
`}`	
$(\!\vert y=x+1 \vert\!)$	If-Statement

Continuing this example, we push the long formula above the if-statement through the assignment, to obtain

$$(x+1-1=0 \to 1=x+1) \wedge (\neg(x+1-1=0) \to x+1=x+1) \qquad (4.8)$$

$$(x = 0 \to 1 = x + 1) \land (\neg(x = 0) \to x + 1 = x + 1)$$

and both these conjuncts, and therefore their conjunction, are clearly valid implications. The above proof now is completed as:

$\llparenthesis \top \rrparenthesis$	
$\llparenthesis (x + 1 - 1 = 0 \to 1 = x + 1) \land (\neg(x + 1 - 1 = 0) \to x + 1 = x + 1) \rrparenthesis$	Implied
`a = x + 1;`	
$\llparenthesis (a - 1 = 0 \to 1 = x + 1) \land (\neg(a - 1 = 0) \to a = x + 1) \rrparenthesis$	Assignment
`if (a - 1 == 0) {`	
$\llparenthesis 1 = x + 1 \rrparenthesis$	If-Statement
`y = 1;`	
$\llparenthesis y = x + 1 \rrparenthesis$	Assignment
`} else {`	
$\llparenthesis a = x + 1 \rrparenthesis$	If-Statement
`y = a;`	
$\llparenthesis y = x + 1 \rrparenthesis$	Assignment
`}`	
$\llparenthesis y = x + 1 \rrparenthesis$	If-Statement

While-statements. Recall that the proof rule for partial correctness of while-statements was presented in the following form in Figure 4.1 – here we have written η instead of ψ:

$$\frac{(\!|\eta \wedge B|\!)\ C\ (\!|\eta|\!)}{(\!|\eta|\!)\ \texttt{while}\ B\ \{C\}\ (\!|\eta \wedge \neg B|\!)}\ \textsf{Partial-while}. \tag{4.9}$$

$$(\!|\phi|\!)\ \texttt{while (}B\texttt{)}\ \{\texttt{C}\}\ (\!|\psi|\!) \tag{4.10}$$

for some ϕ and ψ which are not related in that way. How can we use **Partial-while** in a situation like this?

The answer is that we must *discover* a suitable η, such that

loop-invariant

1. $\vdash_{\mathrm{AR}} \phi \to \eta$,
2. $\vdash_{\mathrm{AR}} \eta \wedge \neg B \to \psi$ and
3. $\vdash_{\mathsf{par}} (\!|\eta|\!)\ \texttt{while (}B\texttt{)}\ \{C\}\ (\!|\eta \wedge \neg B|\!)$

$z \geq 0$

Definition 4.15 An invariant of the while-statement `while (B) {C}` is a formula η such that $\vDash_{\mathsf{par}} (\!|\eta \wedge B|\!)\ C\ (\!|\eta|\!)$ holds; i.e. for all states l, if η and B are true in l and C is executed from state l and terminates, then η is again true in the resulting state.

What is η ?

Example 4.16 Consider the program `Fac1` from page 262, annotated with location labels for our discussion:

Possible invariant η : $y = z!$

x : input

```
    y = 1;
    z = 0;
l1: while (z != x) {
      z = z + 1;
      y = y * z;
l2: }
```

after iteration	z at `l1`	y at `l1`	B at `l1`
0	0	1	true
1	1	1	true
2	2	2	true
3	3	6	true
4	4	24	true
5	5	120	true
6	6	720	false

We wish to establish

$\vDash_{par} (\!|\top|\!)\ \mathrm{Fac1}\ (\!|y = x!|\!)$

How should we use the while-rule in proof tableaux? We need to think about how to push an arbitrary postcondition ψ upwards through a while-statement to meet the precondition ϕ. The steps are:

1. Guess a formula η which you hope is a suitable invariant.
2. Try to prove that $\vdash_{\mathrm{AR}} \eta \wedge \neg B \rightarrow \psi$ and $\vdash_{\mathrm{AR}} \phi \rightarrow \eta$ are valid, where B is the boolean guard of the while-statement. If both proofs succeed, go to 3. Otherwise (if at least one proof fails), go back to 1.
3. Push η upwards through the body C of the while-statement; this involves applying other rules dictated by the form of C. Let us name the formula that emerges η'.
4. Try to prove that $\vdash_{\mathrm{AR}} \eta \wedge B \rightarrow \eta'$ is valid; this proves that η is indeed an invariant. If you succeed, go to 5. Otherwise, go back to 1.
5. Now write η above the while-statement and write ϕ above that η, annotating that η with an instance of Implied based on the successful proof of the validity of $\vdash_{\mathrm{AR}} \phi \rightarrow \eta$ in 2. Mission accomplished!

Example 4.17 We continue the example of the factorial. The partial proof obtained by pushing $y = x!$ upwards through the while-statement – thus checking the hypothesis that $y = z!$ is an invariant – is as follows:

$(y = z!) \wedge (z = x) \rightarrow (y = x!)$

```
y = 1;
z = 0;
```
$(\!|y = z!|\!)$?

```
while (z != x) {
```
$(\!|y = z! \wedge z \neq x|\!)$ Invariant Hyp. $\wedge$ guard

$(\!|y \cdot (z + 1) = (z + 1)!|\!)$ Implied

```
    z = z + 1;
```
$(\!|y \cdot z = z!|\!)$ Assignment

```
    y = y * z;
```
$(\!|y = z!|\!)$ Assignn

```
}
```
$(\!|y = x!|\!)$?

Whether $y = z!$ is a suitable invariant depends on three things:

- The ability to prove that it is indeed an invariant, i.e. that $y = z!$ implies $y \cdot (z + 1) = (z + 1)!$. This is the case, since we just multiply each side of $y = z!$ by $z + 1$ and appeal to the inductive definition of $(z + 1)!$ in Example 4.2.
- The ability to prove that η is strong enough that it and the negation of the boolean guard together imply the postcondition; this is also the case, for $y = z!$ and $x = z$ imply $y = x!$.
- The ability to prove that η is weak enough to be established by the code leading up to the while-statement. This is what we prove by continuing to push the result upwards through the code preceding the while-statement.

$(\!|\top|\!)$

$(\!|1 = 0!|\!)$ Implied

```
y = 1;
```

$(\!|y = 0!|\!)$ Assignment

```
z = 0;
```

$(\!|y = z!|\!)$ Assignment

```
while (z != x) {
```

$(\!|y = z! \wedge z \neq x|\!)$ Invariant Hyp. $\wedge$ guard

$(\!|y \cdot (z + 1) = (z + 1)!|\!)$ Implied

```
    z = z + 1;
```

$(\!|y \cdot z = z!|\!)$ Assignment

```
    y = y * z;
```

$(\!|y = z!|\!)$ Assignment

```
}
```

$(\!|y = z! \wedge \neg(z \neq x)|\!)$ Partial-while

$(\!|y = x!|\!)$ Implied