1. Another version of the factorial program might have been `Fac2`:

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

Unlike the previous version, it 'consumes' the input x. Nevertheless, it correctly calculates the factorial of x and stores the value in y; and we would like to express that as a Hoare triple. However, it is not a good idea to write $(\!|x \geq 0|\!)$ `Fac2` $(\!|y = x!|\!)$ because, if the program terminates, then x will be 0 and y will be the factorial of the initial value of x.

We need a way of remembering the initial value of x, to cope with the fact that it is modified by the program. Logical variables achieve just that: in the specification $(\!|x = x_0 \wedge x \geq 0|\!)$ `Fac2` $(\!|y = x_0!|\!)$ the x_0 is a logical variable and we read it as being universally quantified in the precondition. Therefore, this specification reads: for all integers x_0, if x equals x_0, $x \geq 0$ and we run the program such that it terminates, then the resulting state will satisfy y equals $x_0!$. This works since x_0 cannot be modified by `Fac2` as x_0 does not occur in `Fac2`.

Definition 4.10 For a Hoare triple $(\!|\phi|\!)\ P\ (\!|\psi|\!)$, its set of logical variables are those variables that are free in ϕ or ψ; and don't occur in P.

$$\frac{(\!|\phi|\!)\ C_1\ (\!|\eta|\!) \qquad (\!|\eta|\!)\ C_2\ (\!|\psi|\!)}{(\!|\phi|\!)\ C_1; C_2\ (\!|\psi|\!)}\ \text{Composition}$$

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

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

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

$$\frac{\vdash_{\mathrm{AR}} \phi' \to \phi \qquad (\!|\phi|\!)\ C\ (\!|\psi|\!) \qquad \vdash_{\mathrm{AR}} \psi \to \psi'}{(\!|\phi'|\!)\ C\ (\!|\psi'|\!)}\ \text{Implied}$$

Figure 4.1. Proof rules for partial correctness of Hoare triples.

Proof rules are applied bottom-up.

loop inv.

$\vdash_{AR}$ e.g. $\forall x\,(x \neq x+1)$

For example, let ψ be $x = 6$ and E be 5. Then (ψ) `x = 5` $(\psi[x/E])$ does *not* hold: given a state in which x equals 6, the execution of `x = 5` results in a state in which x equals 5. But $\psi[x/E]$ is the formula $5 = 6$ which holds in no state.

The right way to understand the **Assignment** rule is to think about what you would have to prove about the initial state in order to prove that ψ holds in the resulting state. Since ψ will – in general – be saying something about the value of x, whatever it says about that value must have been true of E, since in the resulting state the value of x is E. Thus, ψ with E in place of x – which says whatever ψ says about x but applied to E – must be true in the initial state.

1. Suppose P is the program `x = 2`. The following are instances of axiom **Assignment**:
   a $(2 = 2)\, P(x = 2)$
   b $(2 = 4)\, P(x = 4)$

   a If you want to prove $x = 2$ after the assignment `x = 2`, then we must be able to prove that $2 = 2$ before it. Of course, 2 is equal to 2, so proving it shouldn't present a problem.
   b If you wanted to prove that $x = 4$ after the assignment, the only way in which it would work is if $2 = 4$; however, unfortunately it is not. More generally, $(\perp)\, x = E\, (\psi)$ holds for any E and ψ – why?

2. Suppose P is $\mathrm{x} = \mathrm{x} + 1$. By choosing various postconditions, we obtain the following instances of the assignment axiom:

a $(x + 1 = 2)\, P(x = 2)$

$\psi : x = 2$

Assignment : $x = x + 1$ E

$\psi(E/x) : x + 1 = 2$

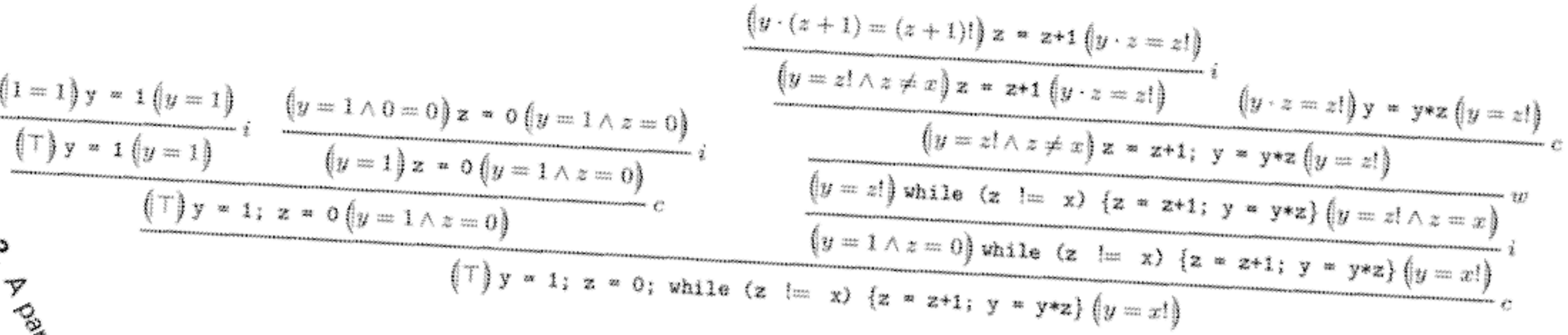

Figure 4.2. A partial-correctness proof for Fac1 in tree form.

a sequence

Program P

$C_1;$
$C_2;$
.
.
.
C_n

$(\!|\phi_0|\!)$ → Pre condition for P

PROOF OUTLINE

$(\!|\phi_0|\!)$

$C_1;$

$(\!|\phi_1|\!)$ justification

$C_2;$

.
.
.

$(\!|\phi_{n-1}|\!)$ justification → mid-condition

$C_n;$

$(\!|\phi_n|\!)$ justification → post-condition for P

Definition 4.12 The process of obtaining ϕ_i from C_{i+1} and ϕ_{i+1} is called computing the weakest precondition of C_{i+1}, given the postcondition ϕ_{i+1}. That is to say, we are looking for the logically weakest formula whose truth at the beginning of the execution of C_{i+1} is enough to guarantee ϕ_{i+1}[4].

[4] ϕ is weaker than ψ means that ϕ is implied by ψ in predicate logic enlarged with the basic facts about arithmetic: the sequent $\vdash_{AR} \psi \rightarrow \phi$ is valid. We want the weakest formula, because we want to impose as few constraints as possible on the preceding code. In some cases, especially those involving while-statements, it might not be possible to extract the logically weakest formula. We just need one which is sufficiently weak to allow us to complete the proof at hand.

The weakest precondition ϕ' is then checked to see whether it follows from the given precondition ϕ. Thus, we appeal to the Implied rule of Figure 4.1.

$\phi_i \rightarrow \phi'$ is true

$\{\phi'\}\ C_i\ \{\phi_{i+1}\}$

Implied