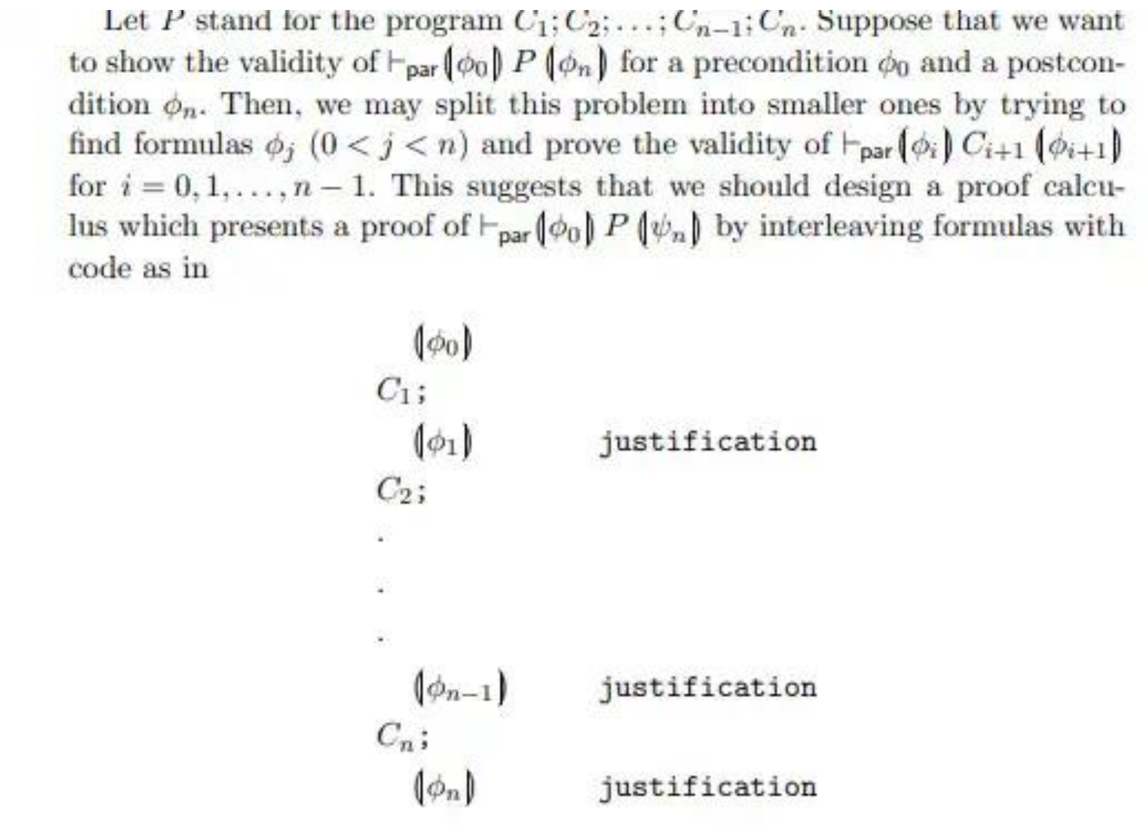
Proof tableaux

It should be clear that proofs in this form are unwieldy to work with. They will tend to be very wide and a lot of information is copied from one line to the next. Proving properties of programs which are longer than Fac1 would be very difficult in this style. The rule for sequential composition suggests a more convenient way of presenting proofs in program logic, called proof tableaux. We can think of any program of our core programming language as a sequence

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

where none of the commands Ci is a composition of smaller programs, i.e. all of the Ci above are either assignments, if-statements or while-statements. Of course, we allow the if-statements and while-statements to have embedded compositions.



Let P stand for the program $C_1; C_2; \dots; C_{n-1}; C_n$. Suppose that we want to show the validity of $\vdash_{\mathsf{par}} (\!|\phi_0|\!)\ P\ (\!|\phi_n|\!)$ for a precondition ϕ_0 and a postcondition ϕ_n. Then, we may split this problem into smaller ones by trying to find formulas ϕ_j $(0 < j < n)$ and prove the validity of $\vdash_{\mathsf{par}} (\!|\phi_i|\!)\ C_{i+1}\ (\!|\phi_{i+1}|\!)$ for $i = 0, 1, \dots, n-1$. This suggests that we should design a proof calculus which presents a proof of $\vdash_{\mathsf{par}} (\!|\phi_0|\!)\ P\ (\!|\psi_n|\!)$ by interleaving formulas with code as in

```
    $(\!|\phi_0|\!)$
$C_1$;
    $(\!|\phi_1|\!)$          justification
$C_2$;
.
.
.
    $(\!|\phi_{n-1}|\!)$      justification
$C_n$;
    $(\!|\phi_n|\!)$          justification
```

Against each formula, we write a justification, whose nature will be clarified shortly. Proof tableaux thus consist of the program code interleaved with formulas, which we call midconditions, that should hold at the point they are written.

Each of the transitions

$$\begin{array}{l} \quad (\!|\phi_i|\!) \\ C_{i+1} \\ \quad (\!|\phi_{i+1}|\!) \end{array}$$

In principle, it seems as though one could start from φ0 and, using C1, obtain φ1 and continue working downwards. However, because the assignment rule works backwards, it turns out that it is more convenient to start with φn and work upwards, using Cn to obtain φn−1 etc.