5.5 Tail Recursion and Iteration

Tail Recursion and While-loops

5.5.1 Fibonacci numbers. The function $\mathrm{fib}(n)$ yielding the n-th element of the *sequence of Fibonacci* satisfies the following recurrences:

$$\begin{aligned} \mathrm{fib}(0) &= 0 \\ \mathrm{fib}(1) &= 1 \\ \mathrm{fib}(n+2) &= \mathrm{fib}(n+1) + \mathrm{fib}(n). \end{aligned}$$

This is an example of course of values recursive definition, where the arguments of recursive applications decrease in the relation $<$: we have $n+1 < n+2$ for the first recursive application and $n < n+2$ for the second.

We can use the recurrences directly for computation. For instance:

$$\begin{aligned} \mathrm{fib}(4) &= \mathrm{fib}(3) + \mathrm{fib}(2) = \big(\mathrm{fib}(2) + \mathrm{fib}(1)\big) + \mathrm{fib}(2) = \\ &= \Big(\big(\mathrm{fib}(1) + \mathrm{fib}(0)\big) + \mathrm{fib}(1)\Big) + \mathrm{fib}(2) = \Big(\big(1 + \mathrm{fib}(0)\big) + \mathrm{fib}(1)\Big) + \mathrm{fib}(2) = \\ &= \big((1+0) + \mathrm{fib}(1)\big) + \mathrm{fib}(2) = \big(1 + \mathrm{fib}(1)\big) + \mathrm{fib}(2) = (1+1) + \mathrm{fib}(2) = \\ &= 2 + \mathrm{fib}(2) = 2 + \big(\mathrm{fib}(1) + \mathrm{fib}(0)\big) = 2 + \big(1 + \mathrm{fib}(0)\big) = \\ &= 2 + (1+0) = 2 + 1 = 3. \end{aligned}$$

The only problem is that the computation sequence is too long. In order to compute the number $\mathrm{fib}(n)$ one needs to use the defining recurrences approximately $\mathrm{fib}(n)$ times. The Fibonacci function grows as fast as the exponential function and to compute the function in this way is simply too wasteful.

We give here more satisfactory implementation of the Fibonacci function by an imperative program. The following PASCAL-like program computes $\mathrm{fib}(n+1)$ into the variable a:

$a, b := 1, 0;$
while $n \neq 0$ *do*
    $n, a, b := n - 1, a + b, a;$

The reader will note that the while-loop is executed only n times. This example is usually given as the 'standard argument' against declarative programming where the recursive version is clearly inferior to the imperative.

The argument is fallacious as one should define an auxiliary ternary function $g(n, a, b)$ with two *accumulators* a and b by primitive recursion:

$$\begin{aligned} g(0, a, b) &= a \\ g(n+1, a, b) &= g(n, a+b, a) \end{aligned}$$

and then we take the following identity as an alternate program for $\mathrm{fib}(n)$:

$$\begin{aligned} \mathrm{fib}(0) &= 0 \\ \mathrm{fib}(n+1) &= g(n,1,0). \end{aligned} \tag{1}$$

The number of recursions of $g(n,a,b)$ is exactly the same as the number of iterations of the loop of the imperative program. Moreover, a good compiler can remove the so-called *tail recursion* in the definition of f and compile it similarly as the while-loop in the above PASCAL-like program.

It remains to show that the identity (1) is true. For that we need the following property of the auxiliary function g:

$$\vdash_{\mathrm{PA}} \ \forall k\, g\big(n, \mathrm{fib}(k+1), \mathrm{fib}(k)\big) = \mathrm{fib}(n+1+k) \tag{2}$$

which is proved by induction on n. In the base case take any k and we have

$$g\big(0, \mathrm{fib}(k+1), \mathrm{fib}(k)\big) = \mathrm{fib}(k+1) = \mathrm{fib}(0+1+k).$$

In the induction step take any k and we have

$$\begin{aligned} g\big(n+1, \mathrm{fib}(k+1), \mathrm{fib}(k)\big) &= g\big(n, \mathrm{fib}(k+1)+\mathrm{fib}(k), \mathrm{fib}(k+1)\big) = \\ &= g\big(n, \mathrm{fib}(k+2), \mathrm{fib}(k+1)\big) = g\big(n, \mathrm{fib}(k+1+1), \mathrm{fib}(k+1)\big) \overset{\mathrm{IH}}{=} \\ &= \mathrm{fib}(n+1+k+1) = \mathrm{fib}(n+1+1+k). \end{aligned}$$

Note that the induction hypothesis is applied with $k+1$ in place of k.

We are now ready to prove (1):

$$\begin{aligned} \mathrm{fib}(n+1) = \mathrm{fib}(n+1+0) &\overset{(2)}{=} g\big(n, \mathrm{fib}(0+1), \mathrm{fib}(0)\big) = \\ &= g\big(n, \mathrm{fib}(1), \mathrm{fib}(0)\big) = g(n,1,0). \end{aligned}$$

Tail Recursion and For-loops

5.5.2 For-loop by backward recursion. Consider the function f_n defined by the following course of values recursion:

$$\begin{aligned} f_0 &= 0 \\ f_1 &= 1 \\ f_{n+2} &= f_{n+1} + f_n + n. \end{aligned}$$

The following PASCAL-like program computes f_{n+2} into the variable a:

```
a, b := 1, 0;
for i := 0 to n do
    a, b := a + b + i, a;.
```

We can simulate the for-loop by the following recursive definition of an auxiliary function g:

$$g(i,n,a,b) = a \leftarrow i > n$$
$$g(i,n,a,b) = g(i+1,n,a+b+i,a) \leftarrow i \le n.$$

We have $i < i+1 \le n+1$ for $i \le n$ and therefore the definition is legal because the measure $n+1 \dot{-} i$ decreases for the arguments of the (only) recursive application g:

$$i \le n \to n+1 \dot{-} (i+1) < n+1 \dot{-} i. \tag{1}$$

We say that g is defined by backward recursion on the difference $n+1 \dot{-} i$.

We claim that

$$\vdash_{\mathrm{PA}} \; f_{n+2} = g(0,n,1,0). \tag{2}$$

We first prove the following property of the auxiliary function g:

$$\vdash_{\mathrm{PA}} \; i \le n+1 \to g(i,n,f_{i+1},f_i) = f_{n+2} \tag{3}$$

by induction with the measure $n+1 \dot{-} i$. Assume $i \le n+1$ and consider two cases. If $i = n+1$ then (3) follows directly from definition:

$$g(n+1,n,f_{n+1+1},f_{n+1}) = f_{n+1+1} = f_{n+2}.$$

If $i \le n$ then $n+1 \dot{-} (i+1) < n+1 \dot{-} i$ by (1) and we get (3) as follows

$$g(i,n,f_{i+1},f_i) = g(i+1,n,f_{i+1}+f_i+i,f_{i+1}) = g(i+1,n,f_{i+2},f_{i+1}) \overset{\mathrm{IH}}{=} f_{n+2}.$$

We are now in position to prove (2):

$$g(0,n,1,0) = g(0,n,f_1,f_0) = g(0,n,f_{0+1},f_0) \overset{(3)}{=} f_{n+2}.$$