Lecture 21 November 9, 2010

In lecture 19 we proved

$$\forall x:\mathbb{N}.\ \exists r:\mathbb{N}.\ r^2 \le x < (r+1)^2$$

and produced a recursive function sqrt(x) to compute it. The function is given in the printed notes along with the proof. The function is an example of primitive recursion whose general form over $\mathbb{N}$ is:

$$f(0,y) = g_1(y)$$
$$f(n+1,y) = g_2(n, f(n,y), y)$$

The sqrti function in the printed notes is

```
let rec sqrti =
    if x=0 then 0
    else let r = sqrt(x-1)
         in if (r+1)^2 ≤ n then r+1
            else r
```

The primitive recursive form is

$$root(0) = 0$$
$$root(n+1) = g_2(n, root(n))$$
$$\text{where } g_2(x,y) = \text{if } (y+1)^2 \le x \text{ then } (y+1) \text{ else } y$$

What is useful about this form is that for every primitive recursive function f we can prove by standard induction

$$\forall x:\mathbb{N}.\ \forall y:\mathbb{N}^k.\ \exists v:\mathbb{N}.\ (f(x,y) = v).$$

Lecture 21 continued 2

Let's consider again the while rule proof that there is an integer square root. We use the Hoare while rule for partial correctness, that is, we don't prove termination of the loop at this point.

$r : \mathbb{N}$, $n : \mathbb{N}$	are declarations
$r := 0$	assign r an initial value
$\{ r^2 \leq n \}$	assertion true by Arith
while $(r+1)^2 \leq n$ **do**	
$\{ r^2 \leq n \}$	loop invariant
$r := r+1$	
$\{ r^2 \leq n \}$	by assignment rule
od	
$\{ r^2 \leq n \}$	an invariant
$\{ n < (r+1)^2 \}$	loop exit condition
$\{ r^2 \leq n < (r+1)^2 \}$	combine assertions

We know that the loop terminates because the distance between n and r decreases on each iteration.

What is the recursive function equivalent to this?

$$root(r, n) = \underline{\text{if}}\ (r+1)^2 \leq n\ \underline{\text{then}}\ root(r+1, n)\ \underline{\text{else}}\ r$$

$root(0, n)$ returns the integer square root.

Theorem: $\forall n : \mathbb{N}.\ \forall r : \mathbb{N}.\ (r < n \Rightarrow Root(root(r, n), n))$

Lecture 21 cont. 3

The while rule in general for partial correctness is

{ Invariant }
while bexp do
{ Invariant }
body
{ Invariant }
od
{ Invariant & ¬ bexp }

We can prove the iterative version using the Least Number Principle (LNP)

$$\exists x{:}\mathbb{N}.\ P(x) \Rightarrow \exists y{:}\mathbb{N}.\ (P(y) \wedge \forall z{:}\mathbb{N}\ (z < y \Rightarrow \neg P(z)))$$

$P(x) = R(x,n) = n < (x+1)^2$

$\exists x{:}\mathbb{N}.\ P(x)$ take $x = n$

$\exists y{:}\mathbb{N}.\ n < (y+1)^2 \quad \forall z{:}\mathbb{N}.\ (z < y \Rightarrow \neg R(z,n))$

$\neg R(z,n) = \neg(n < (z+1)^2)$

$\forall z{:}\mathbb{N}.(z < y \supset \neg(n < (z+1)^2))$

$\forall z{:}\mathbb{N}.(z < y \supset (z+1)^2 \leq n)$

thus for $z = y-1$ we have $((y-1)+1)^2 \leq n$

so $y^2 \leq n$.

We can "prove" the LNP using the Hoare while rule for partial correctness as follows.

Program/Proof

if $P(0,n)$ then return 0

else

$\{ \neg P(0,n) \}$

$r := 0 \quad \{ \forall_3 : \mathbb{N}. (3 < r+1 \Rightarrow \neg P(3,n)) \}$

while $\neg P(r+1, n)$ do

$\{ \neg P(r,n) \}$

$\{ \forall_3 : \mathbb{N}. (3 < r+1 \Rightarrow \neg P(3,n)) \}$

$r := r+1$

$\{ \neg P(r,n) \}$

$\{ \forall_3 : \mathbb{N}. (3 < r+1 \Rightarrow \neg P(3,n)) \}$ by Lemma

od

$\{ P(r+1, n) \}$ thus $\exists y : \mathbb{N}. P(y,n)$ take $y = r+1$

$\{ \forall_3 : \mathbb{N}. (3 < r+1 \Rightarrow \neg P(3,n)) \}$

$\{ \forall_3 : \mathbb{N}. (3 < y \Rightarrow \neg P(3,n)) \}$

Note, we assume that $P(r,n)$ is decidable, we can compute whether $P(r,n)$ or $\neg P(r,n)$.

In some cases we can implement recursion using iteration on a finite set of state variables like r. This kind of evaluation is called tail recursive (it avoids using a stack implementation for those who know about compilers).

Compare the normal recursive evaluation of factorial defined by $fact(0) = 1$, $fact(n+1) = fact(n) \cdot (n+1)$

and this version

```
a := 1
m := 1
{ a = m! }
while m ≤ n do
  { m ≤ n }
  a := a * m
  m := m+1
  { a = (m-1)! }
od
{ n = m+1 }
{ a = (m-1)! }
{ a = n! }
```

The while loop is recursive

$wh(n, m, a) =$ if $m \leq n$ then $wh(n, m+1, a*m)$ else a

$fact(n) = wh(n, 1, 1)$.

Some primitive recursive functions.

Let $S(x) = x+1$. call this $a_0(x,y)$

$add(0,y) = y$

$add(x+1,y) = S(add(x,y))$ call this $a_1(x,y)$

$mult(0,y) = 0$

$mult(x+1,y) = add(mult(x,y),y)$ call this $a_2(x,y)$

$exp(0,y) = 1$ (this $y^0 = 1$)

$exp(x+1,y) = mult(exp(x,y),y)$ ($y^{x+1} = y^x * y$)
call this $a_3(x,y)$

$hyp(0,y) = y$

$hyp(x+1,y) = exp(hyp(x,y),y)$ $y^{y^{y^{\cdots^{y}}}}\}x$
call this $a_4(x,y)$

$a_{m+1}(0,y) = y$

$a_{m+1}(x+1,y) = a_m(a_{m+1}(x,y),y)$

This gives a version of Ackermann's function
$ack(m,x,y)$, a non primitive recursive function.

Over lists primitive recursion is:

$f(nil,y) = g_1(y)$

$f(h.t,y) = g_2(h,t,f(t,y),y)$

The append function, $l_1 @ l_2$ is

$nil @ y = y$

$(h.t) @ y = h.(t @ y)$