ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

D. Basin and P. Müller

Formal Methods and Functional Programming

Solutions of Exercise Sheet 7: Motivation and Induction

Updated: 27/05/10

Assignment 1

```
/* compute the floor of the x-th root of y */
z := 0;
v := 0;
while v < y do
    /* compute (z+1)^x and store it in the auxiliary variable v */
    v := 1;
    i := 0;
    while i < x do
        v := v * (z+1);
        i := i+1
    end;
    /* increase result when still below input value */
    if v <= y  then
        z := z+1
    end
end
```

We first give arguments that the program terminates whenever the variables x and y store in the initial state the values $m > 0$ and $n \geq 0$, respectively. For simplicity, we will not distinguish between a variable and the value that it stores when the state is clear from the context.

Let us observe that the inner while loop terminates. We can identify $(x - i)$ as a *loop variant* - an expression whose value is always a natural number, which is guaranteed to decrease each time the loop body is executed[1]. This follows from the assumption that initially $\mathtt{x} > 0$ and $i = 0$ and

[1] Loop variants will be introduced formally later on in the course.

that x is never assigned to a new value in the program. Note that in each iteration i increases by 1.

Now, we argue that the outer while loop terminates. This time we claim *max*$(0, (y - v))$ works as a loop variant. Note that as soon as this variant becomes zero (i.e., once $v \geq y$), the loop is guaranteed to terminate. We observe that y never gets assigned a new value. So, since the inner while loop terminates in each iteration of the outer while loop, it suffices to show that in each iteration, the value stored in v after the inner while loop is greater than its value in the previous iteration at that program point. In fact, this is the case, since z gets increased at the end of each iteration if $\mathtt{v} \leq \mathtt{y}$. Hence, if $\mathtt{v} < \mathtt{y}$ at the end of the outer while loop, we have that in the next iteration at the end of the inner while loop, v stores a greater value.

Let us now argue that the program computes $\lfloor \sqrt[x]{y} \rfloor$ under the assumption that the program terminates. This requires us to reason in a structured way about unbounded numbers of loop-body executions. For this we need to (informally) introduce the concept of a *loop invariant*. A loop invariant is a predicate that holds (at least) whenever the loop guard is evaluated - i.e., at the start of every execution of the loop body, and at the point where the loop condition fails. Such an invariant is therefore guaranteed to hold directly after the loop has finished executing (along with the negation of the loop condition). Picking a loop invariant can be difficult[2]. Once chosen, it is necessary to prove that the loop invariant holds when the loop is first reached, and is preserved by any execution of the loop body (i.e., we assume it holds at the start of an (arbitrary) execution of loop body, and prove it will hold again after body has been executed). Having proved these facts, it is safe to assume that the invariant always holds after the loop has finished (this reasoning is reminiscent of a proof by induction!).

We first observe that the inner while loop satisfies the invariant $i \leq x \wedge \mathtt{v} = (\mathtt{z}+1)^{\mathtt{i}}$. It is easy to check that this property holds when the loop is first reached, and is preserved each time the loop body is executed (check this!). Therefore, we can deduce that after any execution of the inner while loop, It follows that after the inner while loop, this invariant holds, along with the negation of the loop condition, i.e., we also know that $i \geq x$. Combining this with our invariant gives us that $x = i$ and so we know $\mathtt{v} = (\mathtt{z}+1)^{\mathtt{x}}$ after every execution of the inner loop *(Obs1)*.

Now we need to try to identify a suitable loop invariant for the outer loop. First, we notice that $z^x \leq y$ is an invariant of the outer loop (recall: $z + 1$ is always the next value to be tested). Furthermore, since our loop condition is expressed in terms of v, it would help to have some idea of what v stores at each loop iteration. By inspecting the if-statement at the end of the loop, we observe that:

1. if $v \leq y$ then the value of z gets incremented - i.e., z becomes the value whose power was just tested (i.e., $z^x = v$ after this increment).
2. if $v > y$ then the value of z does not get incremented, so $z + 1$ remains the value whose power was just tested (i.e., $(z + 1)^x = v$ after this increment).

[2]Loop invariants will also be introduced formally in a later part of the course.

Combining these properties, we choose the loop invariant: $z^x \leq y \wedge (v \leq y \Rightarrow v = z^x) \wedge (v > y \Rightarrow v = (z+1)^x$.

We now need to show that this invariant is initially established, and is preserved by every execution of the loop body. When the loop is first reached, we have $z = 0$, $v = 0$, $y \geq 0$, $x \geq 0$ from which the invariant follows straightforwardly.

Now we need to show that whenever the loop body is executed, the invariant is preserved. Suppose that the invariant holds at the start of a loop execution. From the first conjunct of the invariant, we know that $z^x \leq y$. By our previous work *(Obs1)*, we know that once the inner loop has been executed, we will have $v = (z+1)^x$. According to the if-condition, we will then increment z exactly when $v \leq y$. Therefore in either case, we will have $z^x \leq y$ at the end of the outer loop iteration. Furthermore, from the if-condition the other two conjuncts of the invariant follow directly.

Having made these checks, it is now safe for us to assume that after the outer loop terminates, the loop invariant holds, and the loop condition is false. That is, we know $z^x \leq y \wedge (v \leq y \Rightarrow v = z^x) \wedge (v > y \Rightarrow v = (z+1)^x$ and also $v \geq y$. It remains to show that z is indeed the floor of the xth root of y. Using the knowledge that $v \geq y$, we consider two cases:

$v == y$: Then, by the second conjunct of the invariant, we have $v = z^x$. Therefore $z^x = y$ and so z is exactly the xth root of y.

$v > y$: Then, by he third conjunct of the invariant, we have $v = (z+1)^x$. Combining this with $v > y$ and with the first conjunct of the invariant, we obtain $z^x \leq y < (z+1)^x$. So z is the floor of the xth root of y, as required.

Assignment 2

To prove the statement we first define $Q(n)$ as "We can pay exactly n Rappen with 5 and 3 Rappen coins only" and $P(n)$ as $n \geq 8 \Rightarrow Q(n)$.

Proposition: $\forall n \in \mathbb{N}.\ P(n)$.

Proof: Strong induction on n.

Show: For arbitrary n, $P(n)$ holds. That is, show that $n \geq 8 \Rightarrow Q(n)$.

IH: $\forall m < n.\ P(m)$, i.e. $\forall m.\ 8 \leq m < n \Rightarrow Q(m)$

We show $n \geq 8 \Rightarrow Q(n)$ by assuming $n \geq 8$ and proving $Q(n)$, considering the following four cases for n:

- $n = 8$: $Q(n)$ holds by adding 3 Rappen and 5 Rappen
- $n = 9$: $Q(n)$ holds by adding three times 3 Rappen
- $n = 10$: $Q(n)$ holds by adding two times 5 Rappen

- $n \geq 11$: Since $n-3 \geq 8$ and due to our induction hypothesis we can assume that $P(n-3)$ holds. Hence, $Q(n)$ holds by adding another 3 Rappen coin.

Using weak induction we could not use the resulting induction hypothesis, since we assume $Q(n-3)$, not $Q(n-1)$.

Note that we do not need a base case for the induction (strong induction proofs do not have a base cases), although we choose to perform a case distinction on n during the proof. Note that our induction hypothesis is applicable even for the cases where $n = 8, 9, 10$. However, it doesn't help us with these cases (for $n = 8$ it doesn't even tell us anything). The case distinction we choose to perform separates the cases which we choose to prove "directly" from those for which we choose to use the inductive hypothesis, but this is entirely a tactic driven by this particular proof and has nothing to do with the induction schema for strong induction.

By contrast, note that weak induction always requires a base case for $n = 0$ (and *only* for this case) because the form of the inductive case doesn't make any sense for 0 - it would amount to a hypothesis about $n = -1$, which is not a natural number. Again, this has nothing to do with the case(s) we might choose to treat separately for a particular proof.

Assignment 3

NOTE: Unfortunately there was a typo on the exercise sheet, in which the recursive definition of `plus` (rule $(R2)$ below) had its arguments swapped around. Well done to those who managed to work-around (or ignore!) this.

Recall the (correct) definition of the addition operation on Peano numerals:

```
data Nat  =  Zero
          |  Succ Nat
```

```
plus Zero n      = n                (R1)
plus (Succ m) n  = Succ (plus m n)  (R2)
```

Proposition: $\forall x, y, z \in$ `Nat. plus (plus` x y`)` z `= plus` x `(plus` y z`)`

Proof: Let $P(x) := \forall y, z \in$ `Nat. plus (plus` x y`)` z `= plus` x `(plus` y z`)`. We prove $\forall x \in$ `Nat.`$P(x)$ by structural induction on x.

Base case: $x =$ `Zero`

Show: $\forall y, z \in$ `Nat. plus (plus Zero` y`)` z `= plus Zero (plus` y z`)`

Proof: For arbitrary `y,z`

```
plus (plus Zero y) z  =(R1)  plus y z
                      =(R1)  plus Zero (plus y z)
```

Induction step: $x =$ `Succ x'`

IH: $\forall y, z \in$ `Nat. plus (plus x'` y`)` z `= plus x' (plus` y z`)`

Show: $\forall y, z \in$ `Nat. plus (plus (Succ x') y) z = plus (Succ x') (plus y z)`

Proof: For arbitrary `y,z`

$$\begin{aligned}
\texttt{plus (plus (Succ x') y) z} &\overset{R2}{=} \texttt{plus (Succ (plus x' y)) z} \\
&\overset{R2}{=} \texttt{Succ (plus (plus x' y) z)} \\
&\overset{IH}{=} \texttt{Succ (plus x' (plus y z))} \\
&\overset{R2}{=} \texttt{plus (Succ x') (plus y z)}
\end{aligned}$$

Assignment 4 - Headache of the week

Proposition: $\forall m, n \in$ `Nat. plus` m n `= plus` n m

Proof: Let $P(m) :=$ $\forall n \in$ `Nat. plus` m n `= plus` n m. We prove $\forall m \in$ `Nat.`$P(m)$ by structural induction on m.

Base case: $m =$ `Zero`

Show: $\forall n \in$ `Nat. plus Zero` n `= plus` n `Zero`

Proof: Structural induction on n

Base case: $n =$ `Zero`

Show: `plus Zero Zero = plus Zero Zero`

Trivially true.

Induction step: $n =$ `Succ n'`

IH: `plus Zero n' = plus n' Zero`

Show: `plus Zero (Succ n') = plus (Succ n') Zero`

$$\begin{aligned}
\texttt{plus Zero (Succ n')} &\overset{R1}{=} \texttt{(Succ n')} \\
&\overset{R1}{=} \texttt{Succ (plus Zero n')} \\
&\overset{IH}{=} \texttt{Succ (plus n' Zero)} \\
&\overset{R2}{=} \texttt{plus (Succ n') Zero}
\end{aligned}$$

Induction step: $m =$ `Succ m'`

IH1: $\forall n \in$ `Nat. plus m'` n `= plus` n `m'`

Show: $\forall n \in$ `Nat. plus (Succ m')` n `= plus` n `(Succ m')`

Proof: Structural induction on n

Base case: $n =$ `Zero`

Show: `plus (Succ m') Zero = plus Zero (Succ m')`

$$
\begin{array}{ll}
\texttt{plus (Succ m') Zero} & \overset{R2}{=} \texttt{ Succ (plus m' Zero)} \\
& \overset{IH1}{=} \texttt{ Succ (plus Zero m')} \\
& \overset{R1}{=} \texttt{ Succ m'} \\
& \overset{R1}{=} \texttt{ plus Zero (Succ m')}
\end{array}
$$

Induction step: $n =$ `Succ n'`

IH2: `plus (Succ m') n' = plus n' (Succ m')`

Show: `plus (Succ m') (Succ n') = plus (Succ n') (Succ m')`

$$
\begin{array}{ll}
\texttt{plus (Succ m') (Succ n')} & \overset{R2}{=} \texttt{ Succ (plus m' (Succ n'))} \\
& \overset{IH1}{=} \texttt{ Succ (plus (Succ n') m')} \\
& \overset{R2}{=} \texttt{ Succ (Succ (plus n' m'))} \\
& \overset{IH1}{=} \texttt{ Succ (Succ (plus m' n'))} \\
& \overset{R2}{=} \texttt{ Succ (plus (Succ m') n')} \\
& \overset{IH2}{=} \texttt{ Succ (plus n' (Succ m'))} \\
& \overset{R2}{=} \texttt{ plus (Succ n') (Succ m')}
\end{array}
$$