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

Assignment 1

Here is a solution written in C:

```
int sq(x,y)
{
  int z=0, v=0;
  while(v<y) { z++; v=pow(z,x); }
  return z;
}

int pow(z,x)
{
  int r=1, i=x;
  while(i>0) { r = r*z; i--; }
  return r;
}
```

We split our argument of correctness in various sub-arguments.

Functional correctness of power routine

Firstly, we prove that the routine `pow` is correct, *assuming that it terminates*. The argument for termination is orthogonal (independent) and follows below. In formal methods, a proof that *assumes* termination is called a *partial correctness* proof.

We aim to show that the routine `pow` computes the power z^x, where z is its first argument and x its second, assuming both x and z are positive.

The problem that we face is that, even when we assume termination, the calculating loop body may in principle be executed an arbitrary number of times n. One way to deal with this (which can also be used to reason about infinite loops, if termination is not a requirement) is to make an

inductive argument on the number of loop iterations n so far. In particular, we find a condition on the state that is meant to hold after any finite number of iterations of the loop body, and then prove that it does, by induction on n. In formal methods, this condition is called the *loop invariant*.

In the base case, we considering 0 iterations, while the inductive case justifies that if the invariant holds after n iterations, then it holds after $n + 1$ iterations. This boils down to two proof obligations:

- Base case: the code before the loop ensures that the invariant holds when the loop is reached
- Induction step: the code of the body *preserves* the invariant, assuming that the control condition of the loop is initially true.

We choose our loop invariant to be

$$r = z^{x-i} \ \wedge \ x \geq i \geq 0$$

and make the inductive proof as follows: **Base Case: (n=0)** Since the variable r is initialised to 1, and the variable i to the value of x, we need to show that:

$$1 = z^{x-x} \ \wedge \ x \geq x$$

which follows by arithmetic.

Inductive case: Assume the truth of the loop condition (otherwise there will be no $n + 1$-th iteration):

$$i > 0 \quad \text{(LC)}$$

and the loop invariant at the beginning of the execution of the n-th iteration (our inductive hypothesis):

$$r = z^{x-i} \ \wedge \ x \geq i \geq 0 \quad \text{(IH)}$$

Note that we use i, x here etc. to denote the values of the corresponding program variables at the *start* of the loop iteration in question.

Induction Step: We need to prove that the invariant holds at the end of the $n+1$-th iteration; i.e., for the new values of the variables after executing the loop body. By examining the loop body, we can see that the new value of i will be the old $(i - 1)$, while the new value of r is the old $r * z$. So, in terms of the old values, to justify that the loop invariant now holds after the loop body execution, we need to justify that:

$$r * z = z^{x-(i-1)} \ \wedge \ x \geq i - 1 \geq 0 \quad \text{(PO)}$$

The first conjunct of (PO), is simplified (since $z > 0$) into $r = z^{x-i}$, which is proved by (IH).

We split the second conjunct of (PO) into two separate proof obligations: $x \geq i - 1$, proved by (IH), and $i - 1 \geq 0$, proved by (LC).

Correctness Argument: We have now proved that when the loop terminates, the invariant is guaranteed to be true, i.e., $r = z^{x-i} \ \wedge \ x \geq i \geq 0$ holds. We also know that the negation of the loop condition is true at that point, i.e., $i \leq 0$. Together with the loop invariant, this implies $i = 0$, which implies $r = z^x$ as required.

Termination of the power routine

Note that we also need to show that the execution of `pow` terminates for any valid pair of arguments. To do that, we observe that the expression i is always decreased by the loop body. We also observe that the loop condition does not allow the variant to become negative, and that the variant is initially non-negative. Thus, the loop terminates.

The technique that we used to prove termination is to find an expression that is decreased by the loop body, but cannot decrease forever (for example, because its value is always a natural number). This expression is i in our case. This expression is called the *loop variant*.

We need to check that i is initially non-negative when the loop is reached, and that if i is non-negative at the start of a loop iteration, then it will be non-negative and strictly smaller at the and of the iteration.

Such arguments will become formal in later parts of the course.

Correctness of main routine

The loop invariant is now

$$z \geq 0 \ \wedge \ v = z^x \ \wedge \ (z > 0 \Rightarrow (z-1)^x < y) \wedge \ y \geq 0$$

The loop variant is $\max(y - v, 0)$.

Base Case: The proof obligation is:

$$0 \geq 0 \ \wedge \ 0 = 0^x \ \wedge \ (0 > 0 \Rightarrow (0-1)^x < y) \wedge \ y \geq 0$$

which is true (note that we assumed x to be strictly positive; not zero).

Induction Hypothesis: Assume

$$z \geq 0 \ \wedge \ v = z^x \ \wedge \ (z > 0 \Rightarrow (z-1)^x < y) \wedge \ y \geq 0 \quad \text{(IH)}$$

and

$$v < y \quad \text{(LC)}$$

Induction Step: The proof obligation (using also the correctness of the auxiliary routine) becomes (in terms of the old values of the variables):

$$z+1 \geq 0 \ \wedge \ (z+1)^x = (z+1)^x \ \wedge \ (z+1 > 0 \Rightarrow z^x < y) \wedge \ y \geq 0$$

Only the third conjunct is non-trivial; we need to show that $z^x < y$. But we know $v = z^x$ (proved by IH) and so from the loop condition (LC), we get $z^x < y$.

Correctness Argument: After the loop body terminates, we have:

$$z \geq 0 \ \wedge \ v = z^x \ \wedge \ (z > 0 \Rightarrow (z-1)^x < y) \wedge \ y \geq 0 \wedge \ y \leq v$$

If $z = 0$, then we have $v = 0$ and so $y = 0$, and therefore the returned value 0 is correct.

If $z > 0$, then we have $(z-1)^x < y \leq z^x$, which implies:

$$z - 1 < \sqrt[x]{y} \leq z$$

as required.

Termination: The loop variant is always decreased by the loop body. Furthermore, we know it starts off non-negative, and remains so each time the loop iterates. Therefore the loop terminates.

Assignment 2

For our convenience, we name the two defining axioms for `plus`

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

The proof is as follows:

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

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

Base case: $x =$ `Zero`

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

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

Proof: For arbitrary y, z

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

Assignment 3

Using the following labels for the definition cases:

$$\begin{array}{ll} \texttt{plus Zero n} \texttt{ = n} & (P1) \\ \texttt{plus (Succ m) n} \texttt{ = Succ (plus n m)} & (P2) \end{array}$$

$$\begin{array}{lll} \texttt{leq Zero n} \texttt{ = true} & (L1) \\ \texttt{leq (Succ m) Zero} \texttt{ = false} & (L2) \\ \texttt{leq (Succ m) (Succ n)} \texttt{ = leq m n} & (L3) \end{array}$$

The proof is:

Proposition: $\forall n, m \in$ `Nat`$\cdot$ `(leq (plus` n m`)` n `= true)` $\Rightarrow$ m `= Zero`

Proof: By structural induction on n

Base case: $n =$ `Zero`

To show: $\forall m \in$ `Nat`$\cdot$ `(leq (plus Zero` m`) Zero = true)` $\Rightarrow$ m `= Zero`

We use quantifier elimination. For arbitrary m, we assume

`(leq (plus Zero` m`) Zero = true)` (AS1)

and need to prove that m `= Zero`.

We argue by contradiction: suppose instead that

$m \neq$ `Zero` (AS2)

Then, by the definition of `Nat`, m `= Succ` m' for some $m' \in$`Nat`. However, we then find that:

$$\begin{array}{lcl} \texttt{leq (plus Zero } m\texttt{) Zero} & \overset{P1}{=} & \texttt{leq (Succ } m'\texttt{) Zero} \\ & \overset{L2}{=} & \texttt{false} \end{array}$$

which contradicts our assumption (AS1). This contradiction shows that the assumption (AS2) was false, and we conclude $m =$ `Zero` as required.

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

IH: $\forall m \in$ `Nat`$\cdot$ `(leq (plus` n' m`)` n' `= true)` $\Rightarrow$ m `= Zero`

To show:

$$\forall m \in \texttt{Nat} \cdot \texttt{(leq (plus (Succ } n'\texttt{) } m\texttt{) (Succ } n'\texttt{) = true)} \Rightarrow m \texttt{ = Zero}$$

For arbitrary m, we assume `(leq (plus (Succ` n'`)` m`) (Succ` n'`) = true)` and aim to deduce that m `= Zero`:

$$\begin{array}{ll} & \texttt{(leq (plus (Succ } n'\texttt{) } m\texttt{) (Succ } n'\texttt{) = true)} \\ \overset{P2}{\Rightarrow} & \texttt{(leq (Succ (plus } n' \texttt{ } m\texttt{)) (Succ } n'\texttt{) = true)} \\ \overset{L3}{\Rightarrow} & \texttt{leq (plus } n' \texttt{ } m\texttt{) } n' \texttt{ = true} \\ \overset{IH}{\Rightarrow} & m \texttt{ = Zero} \end{array}$$

Assignment 4

We would like to apply structural induction on the type of lists of integers. Using $P(l)$ as an induction hypothesis does not seem to be enough however. For example, assume $P(l)$ and try to prove $P(x : l)$

$$\texttt{sum } (x : l) \;=\; \texttt{sumaux } (x : l)\ 0 \;= ?$$

and we are already stuck! The induction hypothesis $P(l)$ does not say anything about `sumaux`, so we cannot use it.

Our failure to use $P(l)$ was to be expected: `sum` is not recursively defined. It is `sumaux` that is recursively defined, which means that the induction should apply to the definition of `sumaux`.

It is a case where we need to *generalize* the formula that we want to prove, to take into account all possible inputs to `sumaux` (formally this means that we prove a *stronger* property that what is expected). The formula to be proved is:

$$\forall l \in \texttt{[Int]} \cdot \forall i \in \mathbb{Z} \cdot \left(\texttt{sumaux } l\ i \;=\; i + \sum_{j=0}^{(\texttt{length } l)-1} l\,!\,!\,j \right)$$

We can now apply induction on l.

Base case: $l =$ `[]`. We need to prove:

$$\forall i \in \mathbb{Z} \cdot \texttt{ sumaux [] } i \;=\; i + \sum_{j=0}^{-1} \texttt{[]!!}j$$

where the summation over an empty domain that appears here is equal to 0. The formula to be proven is equal to

$$\forall i \in \mathbb{Z} \cdot \texttt{ sumaux [] } i \;=\; i$$

which is true by the definition of `sumaux`.

Induction hypothesis. Assume:

$$\forall i \in \mathbb{Z} \cdot \texttt{ sumaux } l\ i \;=\; i + \sum_{j=0}^{(\texttt{length } l)-1} l\texttt{!!}j \quad \text{(IH)}$$

for some arbitrary integer list l. Also assume that x is an arbitrary integer.

Induction step. We need to prove

$$\forall i \in \mathbb{Z} \cdot \texttt{ sumaux } (x : l)\ i \;=\; i + \sum_{j=0}^{(\texttt{length } (x:l))-1} (x : l)\texttt{!!}j$$

To prove this quantification, we use quantifier elimination. This means that we assume i' is an arbitrary integer, and prove instead:

$$\texttt{sumaux } (x : l)\ i' \;=\; i' + \sum_{j=0}^{(\texttt{length } (x:l))-1} (x : l)\texttt{!!}j$$

By the definition of `sumaux` this is equivalent to

$$\texttt{sumaux } l\ (x + i') \;=\; i' + \sum_{j=0}^{(\texttt{length } (x:l))-1} (x : l)\texttt{!!}j$$

Now, we apply (IH) specializing the quantification for $i = x+i'$. This gives:

$$i' + x + \left(\sum_{j=0}^{(\texttt{length } l)-1} l\texttt{!!}j \right) \;=\; i' + \sum_{j=0}^{(\texttt{length } (x:l))-1} (x : l)\texttt{!!}j$$

or, equivalently:

$$x + \left(\sum_{j=0}^{(\texttt{length}\ l)-1} l\,!\,!\,j \right) = \sum_{j=0}^{(\texttt{length}\ (x:l))-1} (x:l)\,!\,!\,j$$

To prove the last formula, we can start from the right side, as follows:

$$\begin{aligned}
\sum_{j=0}^{(\texttt{length}\ (x:l))-1} (x:l)\,!\,!\,j &= (x:l)\,!\,!\,0 + \left(\sum_{j=1}^{(\texttt{length}\ (x:l))-1} (x:l)\,!\,!\,j \right) \\
&= x + \left(\sum_{j=0}^{(\texttt{length}\ (x:l))-2} (x:l)\,!\,!\,(j+1) \right) \\
&= x + \left(\sum_{j=0}^{(\texttt{length}\ l)-1} l\,!\,!\,j \right)
\end{aligned}$$

Original Problem. The original problem is now proved as follows. Assume l is an arbitrary list. Our proof obligation is:

$$\texttt{sum}\ l = \sum_{i=0}^{(\texttt{length}\ l)-1} l\,!\,!\,i$$

By the definition of `sum`, this is equivalent to:

$$\texttt{sumaux}\ l\ 0 = \sum_{i=0}^{(\texttt{length}\ l)-1} l\,!\,!\,i$$

By the lemma that we have proved (specialization for $i = 0$):

$$0 + \left(\sum_{j=0}^{(\texttt{length}\ l)-1} l\,!\,!\,j \right) = \sum_{i=0}^{(\texttt{length}\ l)-1} l\,!\,!\,i$$

which is true.

Assignment 5

Useful Lemmas

Apart from Lemma 1 and Lemma 2, given in the exercise sheet, our proof uses the following lemmas:

Lemma 3: List concatenation is associative:

$$(a\texttt{++}b)\texttt{++}c \;=\; a\texttt{++}(b\texttt{++}c)$$

Lemma 4: Length of concatenation:

$$\texttt{length}(a\texttt{++}b) \;=\; \texttt{length}\ a + \texttt{length}\ b$$

Lemma 5: Length of comprehension:

$$\texttt{length}(\texttt{[x | x<-}l\texttt{,}\ P(\texttt{x})\texttt{]}) + \texttt{length}(\texttt{[x | x<-}l\texttt{,}\ \neg P(\texttt{x})\texttt{]}) \;=\; \texttt{length}\ l$$

Lemma 6: Indexing concatenation:

$$\begin{array}{l} k \in \mathbb{N} \;\wedge\; k < \texttt{length}\ a \;\;\Rightarrow\;\; (a\texttt{++}[p]\texttt{++}b)\texttt{!!}k \;=\; a\texttt{!!}k \\ (a\texttt{++}[p]\texttt{++}b)\texttt{!!}(\texttt{length}\ a) \;=\; p \\ k \in \mathbb{N} \;\wedge\; k > \texttt{length}\ a \;\;\Rightarrow\;\; (a\texttt{++}[p]\texttt{++}b)\texttt{!!}k \;=\; b\texttt{!!}(k - \texttt{length}\ a - 1) \end{array}$$

Main Proof

First, we need to define what it means for a list l to be sorted. Our predicate *sorted* is defined as follows:

$$\mathit{sorted}(l) \;\stackrel{\text{def}}{=}\; \forall i, j \in \mathbb{N} \cdot i < j < \texttt{length}\ l \Rightarrow\; l\texttt{!!}i \leq l\texttt{!!}j$$

Apart from sortedness, we need the property that `qsort` l has the same length as l.

We are going to prove the property by *strong induction* on the length of l. That is, we assume that

$$\begin{array}{l} \forall m \in \texttt{[Int]} \cdot \texttt{length}\ m < \texttt{length}\ l \\ \quad\Rightarrow\; \mathit{sorted}(\texttt{qsort}\ m) \;\wedge\; \texttt{length}\ m = \texttt{length}(\texttt{qsort}\ m) \quad (\mathsf{IH}) \end{array}$$

and our proof obligation is now

$$\mathit{sorted}(\texttt{qsort}\ l) \;\wedge\; \texttt{length}\ l = \texttt{length}(\texttt{qsort}\ l) \quad (\mathsf{PO1})$$

We now split into two cases:

- $l = \texttt{[]}$

  In this case, by the definition of `qsort`, and that of `length`, (PO1) becomes

  $$\mathit{sorted}(\texttt{[]}) \ \wedge \ 0 = 0$$

  By the definition of *sorted*, (PO1) is equivalent to a vacuous universal quantification, and therefore true.

- $l = p : xs$

  In this case, we define:

  $$\begin{aligned} l_1 &= \texttt{[x| x<-}xs\texttt{, x}\leq p\texttt{]} \\ l_2 &= \texttt{[x| x<-}xs\texttt{, x}> p\texttt{]} \end{aligned}$$

  and observe that the definition of `qsort`, we have

  $$\texttt{qsort}(l) \ = \ \texttt{qsort}\ l_1\ \texttt{++}\ \texttt{[}p\texttt{]}\ \texttt{++}\ \texttt{qsort}\ l_2$$

  By Lemma 1, and the definition of `length`, we have for both $i = 1, 2$:

  $$\texttt{length}\ l_i \leq \texttt{length}\ xs = (\texttt{length}\ l) - 1 < \texttt{length}\ l$$

  Now (IH) gives us for both $i = 1, 2$:

  $$\mathit{sorted}(\texttt{qsort}\ l_i) \ \wedge \ \texttt{length}(\texttt{qsort}\ l_i) = \texttt{length}\ l_i$$

  We break (PO1) into two proof obligations:

  $$\begin{aligned} &\mathit{sorted}(\texttt{qsort}\ l) \quad \text{(PO1.1)} \\ &\texttt{length}(\texttt{qsort}\ l) = \texttt{length}\ l \quad \text{(PO1.2)} \end{aligned}$$

  and we perform the following computation:

  $$\begin{aligned} \texttt{length}(\texttt{qsort}\ l) &\overset{L3,L4}{=} \texttt{length}(\texttt{qsort}\ l_1) + \texttt{length}\texttt{[}p\texttt{]} + \texttt{length}(\texttt{qsort}\ l_2) \\ &\overset{IH}{=} \texttt{length}\ l_1 + \texttt{length}\texttt{[}p\texttt{]} + \texttt{length}\ l_2 \\ &= \texttt{length}\texttt{[}p\texttt{]} + \texttt{length}\ l_1 + \texttt{length}\ l_2 \\ &\overset{L5}{=} \texttt{length}\texttt{[}p\texttt{]} + \texttt{length}\ xs \\ &\overset{L4}{=} \texttt{length}(p : xs) \\ &= \texttt{length}\ l \end{aligned}$$

  which proves (PO1.2).

To prove (PO1.1), we use quantifier elimination. That is, we assume that i, j are arbitrary integers, such that

$$0 \leq i < j < \texttt{length}\ l \quad \text{(AS)}$$

and prove:

$$l\,!\,!i \leq l\,!\,!j \quad \text{(PO1.2a)}$$

We split into cases.

- Case 1: $j < \texttt{length}\ l_1$. By Lemma 6 and (AS), we have

  $$l\,!\,!i = l_1\,!\,!i \ \wedge\ l\,!\,!j = l_1\,!\,!j$$

  Together with (IH) (instantiate $m = l_1$) and (AS), this proves (PO1.2a).
- Case 2: $j = \texttt{length}\ l_1$. By Lemma 6 and (AS), we have

  $$l\,!\,!i = l_1\,!\,!i \ \wedge\ l\,!\,!j = p$$

  Lemma 2 proves (PO1.2a).
- Case 3: $i < \texttt{length}\ l_1 < j$. By Lemma 6, we have

  $$l\,!\,!i = l_1\,!\,!i \ \wedge\ l\,!\,!j = l_2\,!\,!(j - \texttt{length}\ l_1 - 1)$$

  Lemma 2 now gives us $l\,!\,!i \leq p < l\,!\,!j$ which proves (PO1.2a).
- Case 4: $i = \texttt{length}\ l_1 < j$. By Lemma 6, we have

  $$l\,!\,!i = p \ \wedge\ l\,!\,!j = l_2\,!\,!(j - \texttt{length}\ l_1 - 1)$$

  Lemma 2 proves (PO1.2a).
- Case 5: $\texttt{length}\ l_1 < i < j$. By Lemma 6, we have

  $$l\,!\,!i == l_2\,!\,!(i - \texttt{length}\ l_1 - 1) \ \wedge\ l\,!\,!j = l_2\,!\,!(j - \texttt{length}\ l_1 - 1)$$

  (IH) (instantiate $m = l_2$) proves (PO1.2a).

By (AS), all cases are covered. This completes the proof.