COMP 3610 Tutorial 2

10 August, 2023

Exercise 1

1. Use induction over natural numbers to prove that $3^n - 1$ is a multiple of 2, for all natural numbers $n \geq 1$.

2. Prove, by structural induction over lists:

$$\text{length } (xs@ys) = (\text{length } xs) + (\text{length } (ys))$$

   Here, @ stands for list concatenation. Before starting the induction, you should derive a formal definition for this operator, as well as for the function length.

Exercise 2

The following is a grammar for non-empty sequences of nested parentheses:

$$par ::= () \mid [] \mid (par) \mid [par] \mid par\ par$$

1. Show the structural induction principle for *par*.

2. Use the induction principle to show that for all instances of *par*, for each "("-symbol, there is a matching ")"-symbol.

Exercise 3

Consider the following data type and function definitions in Haskell:

```haskell
data IntList = INil | ICons Int IntList
data IntTree = Leaf Int | Node Int IntTree IntTree

listSum :: IntList -> Int
listSum INil = 0
listSum (ICons n r) = n + (listSum r)

treeSum :: IntTree -> Int
treeSum (Leaf n) = n
```



```
treeSum (Node n l r) = n + (treeSum l) + (treeSum r)

flatten :: IntTree -> IntList
flatten (Leaf n) = ICons n INil
flatten (Node n l r) = ICons n ((flatten l) @ (flatten r))
```

Here, @ is a version of the concatenation function you derived for exercise one, adapted to the IntList data type.

1. Prove that, for all IntLists l and r,

   listSum (l @ r) = (listSum l) + (listSum r)

2. Prove that, for all IntTrees t,

   treeSum t = listSum (flatten t)

Exercise 4

Using IMP extended with functions as in Lecture 5,

1. Write a program P that returns 2 under call-by-value semantics and 3 under call-by-name semantics. Show the intermediate program states for both executions.

2. Write a program P that runs forever under call-by-value semantics while terminating under call-by-name semantics. Show the intermediate program states for both executions (in the first version, until you encounter a state you have seen before).

