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 8: Structural Induction

Assignment 1

Let $B(n)$ denote the minimum number of breaks needed to split a bar with n squares.

We want to prove that $B(n) = n - 1$. We call this predicate $P(n)$. The proof is based on strong induction. Then we assume that $P(k)$ is true for all $k \leq n$. We want to prove $P(n+1)$. However the first break of the bar is made, we will end up with two smaller pieces. We suppose that the two pieces are of sizes n_1 and n_2. So we have that $n_1 + n_2 = n$ and $n_1, n_2 > 0$. The minimum number of breaks for the initial bar will be $B(n) = B(n_1) + B(n_2) + 1$. By inductive hypothesis, we have that $B(n_1) = n_1 - 1$ and $B(n_2) = n_2 - 1$. Then we have that $B(n_1) + B(n_2) + 1 = (n_1 - 1) + (n_2 - 1) + 1 = n_1 + n_2 - 1 = n - 1$.

Assignment 2

Let $P(n)$ denote the statement "n can be written as a prime or as the product of two or more primes". We want to prove that $\forall n.[n \geq 2 \Rightarrow P(n)]$. Note that for all $k < 2$ the implication $n \geq 2 \Rightarrow P(n)$ trivially holds.

The proof is based on strong induction. Then we assume $P(k)$ is true for all $k \leq n$. We want to prove that $P(n+1)$ is true. We need to consider two distinct cases:

- $n+1$ is prime. $P(n+1)$ is true since $n+1$ can be written as a prime, itself.
- $n+1$ is not prime, that is, there exist two positive integers $2 \leq a, b < n+1$ such that $n+1 = a * b$. Since $a, b < n+1$, by induction hypothesis a and b can be written as a prime or as the product of two or more primes. Then also $n+1$ can be written as a product of primes, that are the concatenation of the primes of a and of the primes of b.

Assignment 3

- **Base step:** t is a leaf. Then we have

$$N(t) = 0$$
$$H(t) = 0$$

So $N(t) < 2^{H(t)}$.

- **Inductive step:** t is a node, that is, $t = \texttt{Node t1 t2}$. We suppose that the property holds for the two children.

  By definition of N, $N(\texttt{Node t1 t2}) = N(t1) + N(t2) + 1$. By inductive hypothesis, we have that $N(t1) < 2^{H(t1)}$ and $N(t2) < 2^{H(t2)}$. Then we know that $N(t1) + N(t2) + 1 \leq 2^{H(t1)} - 1 + 2^{H(t2)} - 1 + 1$. By basic arithmetic properties, we have that $2^{H(t1)} + 2^{H(t2)} - 1 \leq 2^{max(H(t1),H(t2))} + 2^{max(H(t1),H(t2))} - 1 = 2^{1+max(H(t1),H(t2))} - 1 < 2^{1+max(H(t1),H(t2))}$. By definition of H we have that $H(\texttt{Node t1 t2}) = 1 + \texttt{max}(H(\texttt{t1}), H(\texttt{t2}))$, then we have that $2^{1+max(H(t1),H(t2))} = 2^{H(\texttt{Node t1 t2})}$. Then we proved that $N(\texttt{Node t1 t2}) < 2^{H(\texttt{Node t1 t2})}$

Assignment 4

We define $b[y \mapsto e]$ as follows:

$$b[y \mapsto e] = \begin{cases} e_1[y \mapsto e] \; op \; e_2[y \mapsto e] & \text{if } b \text{ is the arithmetic comparison } e_1 \; op \; e_2, \\ \texttt{not } b'[y \mapsto e] & \text{if } b \text{ is the Boolean expression } \texttt{not } b', \text{ and} \\ b_1[y \mapsto e] \oplus b_1[y \mapsto e] & \text{if } b \text{ is the Boolean expression } b_1 \oplus b_2 \\ & \quad \text{with } \oplus \in \{\texttt{and}, \texttt{or}\}. \end{cases}$$

We prove by structural induction over b that $\mathcal{B}[\![b[y \mapsto e]]\!]\sigma = \mathcal{B}[\![b]\!]\big(\sigma[y \mapsto \mathcal{A}[\![e]\!]\sigma]\big)$.

- Base Case: $b = e_1 \; op \; e_2$. We have that

$$\begin{aligned} \mathcal{B}[\![(e_1 \; op \; e_2)[y \mapsto e]]\!]\sigma &= \mathcal{B}[\![e_1[y \mapsto e] \; op \; e_2[y \mapsto e]]\!]\sigma \\ &= \mathcal{A}[\![e_1[y \mapsto e]]\!]\sigma \; \overline{op} \; \mathcal{A}[\![e_2[y \mapsto e]]\!]\sigma \\ &\overset{\text{(a)}}{=} \mathcal{A}[\![e_1]\!]\big(\sigma[y \mapsto \mathcal{A}[\![e]\!]\sigma]\big) \; \overline{op} \; \mathcal{A}[\![e_2]\!]\big(\sigma[y \mapsto \mathcal{A}[\![e]\!]\sigma]\big) \\ &= \mathcal{B}[\![e_1 \; op \; e_2]\!]\big(\sigma[y \mapsto \mathcal{A}[\![e]\!]\sigma]\big) \, . \end{aligned}$$

- Step Case: $b = \texttt{not } b'$. We have that

$$\begin{aligned} \mathcal{B}[\![(\texttt{not } b')[y \mapsto e]]\!]\sigma &= \mathcal{B}[\![\texttt{not } b'[y \mapsto e]]\!]\sigma \\ &= \neg \, \mathcal{B}[\![b'[y \mapsto e]]\!]\sigma \\ &\overset{\text{IH}}{=} \neg \, \mathcal{B}[\![b']\!]\big(\sigma[y \mapsto \mathcal{A}[\![e]\!]\sigma]\big) \\ &= \mathcal{B}[\![\texttt{not } b']\!]\big(\sigma[y \mapsto \mathcal{A}[\![e]\!]\sigma]\big) \, . \end{aligned}$$

- Step Case: $b = b_1 \oplus b_2$ with $\oplus \in \{\texttt{and}, \texttt{or}\}$. We have that

$$\begin{aligned} \mathcal{B}[\![(b_1 \oplus b_2)[y \mapsto e]]\!]\sigma &= \mathcal{B}[\![(b_1[y \mapsto e] \; \oplus \; b_2[y \mapsto e]]\!]\sigma \\ &= \mathcal{B}[\![b_1[y \mapsto e]]\!]\sigma \; \overline{\oplus} \; \mathcal{B}[\![b_2[y \mapsto e]]\!]\sigma \\ &\overset{\text{IH}}{=} \mathcal{B}[\![b_1]\!]\big(\sigma[y \mapsto \mathcal{A}[\![e]\!]\sigma]\big) \; \overline{\oplus} \; \mathcal{B}[\![b_2]\!]\big(\sigma[y \mapsto \mathcal{A}[\![e]\!]\sigma]\big) \\ &= \mathcal{B}[\![b_1 \oplus b_2]\!]\big(\sigma[y \mapsto \mathcal{A}[\![e]\!]\sigma]\big) \, . \end{aligned}$$

  Here, $\overline{\oplus}$ denotes the corresponding Boolean operation.

Assignment 5

```
data Aexp = Num Integer
          | Var String
          | Add Aexp Aexp
          | Sub Aexp Aexp
          | Mul Aexp Aexp

data Op = Eq | Neq | Le | Leq | Ge | Geq

data Bexp = Rel Op Aexp Aexp
          | Not Bexp
          | Or Bexp Bexp
          | And Bexp Bexp

data State = VarAssign (String -> Integer)

evalAexp :: Aexp -> State -> Integer
evalAexp (Num n)     _               = n
evalAexp (Var x)     (VarAssign val) = val x
evalAexp (Add e1 e2) sigma = (evalAexp e1 sigma) + (evalAexp e2 sigma)
evalAexp (Sub e1 e2) sigma = (evalAexp e1 sigma) - (evalAexp e2 sigma)
evalAexp (Mul e1 e2) sigma = (evalAexp e1 sigma) * (evalAexp e2 sigma)

evalBexp :: Bexp -> State -> Bool
evalBexp (Rel op e1 e2) sigma =
    (evalOp op) (evalAexp e1 sigma) (evalAexp e2 sigma)
  where evalOp Eq  = (==)
        evalOp Neq = (/=)
        evalOp Le  = (<)
        evalOp Leq = (<=)
        evalOp Ge  = (>)
        evalOp Geq = (>=)
evalBexp (Not b)     sigma = not (evalBexp b sigma)
evalBexp (Or b1 b2)  sigma = (evalBexp b1 sigma) || (evalBexp b2 sigma)
evalBexp (And b1 b2) sigma = (evalBexp b1 sigma) && (evalBexp b2 sigma)
```

Assignment 6

Extension of the semantics

The only extensions to **Bexp** are `Aexp`, `implies` and `iff`.
The definitions are as follows:

$$
\begin{aligned}
\mathcal{B}'[\![\texttt{Aexp}]\!]\sigma &= \mathcal{B}[\![\texttt{Aexp\#0}]\!]\sigma \\
\mathcal{B}'[\![b_1' \texttt{ implies } b_2']\!]\sigma &= \begin{cases} tt & \text{if } \mathcal{B}'[\![b_1']\!]\sigma = ff \text{ or } \mathcal{B}'[\![b_2']\!]\sigma = tt \\ ff & \text{otherwise} \end{cases} \\
\mathcal{B}'[\![b_1' \texttt{ iff } b_2']\!]\sigma &= \begin{cases} tt & \text{if } \mathcal{B}'[\![b_1']\!]\sigma = \mathcal{B}'[\![b_2']\!]\sigma \\ ff & \text{otherwise} \end{cases}
\end{aligned}
$$

Proof of logical equivalence
We use induction on the structure of boolean expressions of **Bexp'**.

1. (base case) $b' = \texttt{Aexp}$.
   Let's guess that a corresponding $b \in$ **Bexp** expression is `Aexp#0`. We have to show that

   $$\mathcal{B}'[\![\texttt{Aexp}]\!]\sigma = \mathcal{B}[\![\texttt{Aexp\#0}]\!]\sigma$$

   This is trivially true, since by definition of $\mathcal{B}'$ we have that $\mathcal{B}'[\![\texttt{Aexp}]\!]\sigma = \mathcal{B}[\![\texttt{Aexp\#0}]\!]\sigma = \mathcal{B}[\![\texttt{Aexp\#0}]\!]\sigma$

2. (composite element) $b' = b_1' \texttt{ implies } b_2'$.
   Our guess here is $b = \texttt{not } b_1 \texttt{ or } b_2$, where $\mathcal{B}'[\![b_1']\!] = \mathcal{B}[\![b_1]\!]$ and $\mathcal{B}'[\![b_2']\!] = \mathcal{B}[\![b_2]\!]$. From the induction hypothesis we know that there exist such b_1 and $b_2 \in$ **Bexp** expressions.

   We have to show that

   $$\mathcal{B}'[\![b_1' \texttt{ implies } b_2']\!]\sigma = \mathcal{B}[\![\texttt{not } b_1 \texttt{or } b_2]\!]\sigma$$

   Using the definition of **Bexp'** and the induction hypothesis, we get

   $$\mathcal{B}'[\![b_1' \texttt{ implies } b_2']\!]\sigma = \begin{cases} tt & \text{if } \mathcal{B}[\![b_1]\!]\sigma = ff \text{ or } \mathcal{B}[\![b_2]\!]\sigma = tt \\ ff & \text{otherwise} \end{cases}$$

   Using the definition of **Bexp** we get

   $$
   \begin{aligned}
   \mathcal{B}[\![\texttt{not } b_1 \texttt{ or } b_2]\!]\sigma &= \begin{cases} tt & \text{if } \mathcal{B}[\![\texttt{not } b_1]\!]\sigma = tt \text{ or } \mathcal{B}[\![b_2]\!]\sigma = tt \\ ff & \text{otherwise} \end{cases} \\
   &= \begin{cases} tt & \text{if } \mathcal{B}[\![b_1]\!]\sigma = ff \text{ or } \mathcal{B}[\![b_2]\!]\sigma = tt \\ ff & \text{otherwise} \end{cases}
   \end{aligned}
   $$

   □

3. (composite element) $b' = b_1' \texttt{ iff } b_2'$.
   Our guess this time is

   $$b = (b_1 \texttt{ and } b_2) \texttt{ or } (\texttt{not } b_1 \texttt{ and not } b_2)$$

   where $\mathcal{B}'[\![b_1']\!] = \mathcal{B}[\![b_1]\!]$ and $\mathcal{B}'[\![b_2']\!] = \mathcal{B}[\![b_2]\!]$. Again, from the induction hypothesis we know that there exist such b_1 and $b_2 \in$ **Bexp** expressions.

We have to show that

$$\mathcal{B}'[\![b_1' \texttt{ iff } b_2']\!]\sigma = \mathcal{B}[\![(b_1 \texttt{ and } b_2) \texttt{ or } (\texttt{not } b_1 \texttt{ and not } b_2)]\!]\sigma$$

Using the definition of **Bexp'** and the induction hypothesis, we get

$$\mathcal{B}'[\![b_1' \texttt{ iff } b_2']\!]\sigma \;=\; \begin{cases} tt & \text{if } \mathcal{B}[\![b_1]\!]\sigma = \mathcal{B}[\![b_2]\!]\sigma \\ ff & \text{otherwise} \end{cases}$$

Using the definition of **Bexp** we get

$$\begin{aligned} \mathcal{B}[\![(b_1 \texttt{ and } b_2) \texttt{ or } (\texttt{not } b_1 \texttt{ and not } b_2)]\!]\sigma &= \begin{cases} tt & \text{if } \mathcal{B}[\![b_1 \texttt{ and } b_2]\!]\sigma = tt \text{ or} \\ & \quad \mathcal{B}[\![\texttt{not } b_1 \texttt{ and not } b_2]\!]\sigma = tt \\ ff & \text{otherwise} \end{cases} \\ &= \begin{cases} tt & \text{if } \mathcal{B}[\![b_1]\!]\sigma = tt \text{ and } \mathcal{B}[\![b_2]\!]\sigma = tt \quad \text{or} \\ & \quad \mathcal{B}[\![\texttt{not } b_1]\!]\sigma = tt \text{ and } \mathcal{B}[\![\texttt{not } b_2]\!]\sigma = tt \\ ff & \text{otherwise} \end{cases} \\ &= \begin{cases} tt & \text{if } \mathcal{B}[\![b_1]\!]\sigma = tt \text{ and } \mathcal{B}[\![b_2]\!]\sigma = tt \quad \text{or} \\ & \quad \mathcal{B}[\![b_1]\!]\sigma = ff \text{ and } \mathcal{B}[\![b_2]\!]\sigma = ff \\ ff & \text{otherwise} \end{cases} \\ &= \begin{cases} tt & \text{if } \mathcal{B}[\![b_1]\!]\sigma = \mathcal{B}[\![b_2]\!]\sigma \\ ff & \text{otherwise} \end{cases} \end{aligned}$$

□

4. Other composite elements are just applications of the induction hypothesis.