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 $\forall n > 0 \Rightarrow B(n) = n - 1$. So we define the predicate $P(n) : n > 0 \Rightarrow B(n) = n - 1$. The proof is based on strong induction. That is, we assume that $P(k)$ is true for all $k < n$, and we want to prove $P(n)$. To prove $n > 0 \Rightarrow B(n) = n - 1$ we assume $n > 0$ and we attempt to prove $B(n) = n - 1$. Two cases are possible:

- $n = 1$. In this case, the bar already only consists of small squares. We did use 0 breaks, as claimed.
- $n > 1$. No matter how 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 $n_1, n_2 > 0$. Since $n_1 < n \wedge n_2 < n$ and they are both greater than zero, we can apply on them the IH. 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 $B(n_1) + B(n_2) + 1 = (n_1 - 1) + (n_2 - 1) + 1 = n_1 + n_2 - 1 = n - 1$.

Assigment 2

The proof by strong induction is on the number n of matches in each pile. We assume that the claim holds for all $k < n$, and show that it also holds for n.

Suppose that both piles contain n matches, and that the first player removes j matches from one pile, where $0 < j \leq n$. One pile then contains n matches and the other one $n - j$ matches.

We make a case distinction over j:

- $j == n$: the second player removes j matches from the other pile. After that, both piles are empty, and this means that the second player has removed the last match winning the game.
- $j < n$: the second player removes j matches from the other pile. After that, the two piles contain $n - j$ matches. By induction hypothesis, we assumed that $\forall k < n$ the second

player wins if both the piles contain k matches. Therefore we know that the second player will win the game with $n - j$ matches, and so that it wins the match with n matches.

Assignment 3

The property we want to prove is $\forall a \in \mathrm{Aexp} : P(a) : \forall \sigma, \sigma' \in State, \forall x \in FV(a) \cdot \sigma(x) = \sigma'(x) \Rightarrow \mathcal{A}[\![a]\!]\sigma = \mathcal{A}[\![a]\!]\sigma'$. We prove it by structural induction on a.

- **Base case 1**: a is a numeral n. By definition of $\mathcal{A}$, we have that $\mathcal{A}[\![n]\!]\sigma = \mathbb{N}[\![n]\!]$ and $\mathcal{A}[\![n]\!]\sigma' = \mathbb{N}[\![n]\!]$. Then we have that $\mathcal{A}[\![n]\!]\sigma = \mathcal{A}[\![n]\!]\sigma'$ by the transitive property of equality.
- **Base case 2**: a is a variable y. By definition of FV, $x \in FV(y)$ is only possible if $x = y$. By definition of $\mathcal{A}$, we have that $\mathcal{A}[\![x]\!]\sigma = \sigma(x)$ and $\mathcal{A}[\![x]\!]\sigma' = \sigma'(x)$. Then, by the definition of σ and σ', we have that $\sigma(x) = \sigma'(x)$. Therefore, by transitive property of equality we proved that $\mathcal{A}[\![n]\!]\sigma = \mathcal{A}[\![n]\!]\sigma'$.
- **Inductive case**: a is an arithmetic expression a_1 <OP> a_2 where <OP> $\in \{'+', '-', '*'\}$. By definition of $\mathcal{A}$, we have that $\mathcal{A}[\![a_1$ <OP> $a_2]\!]\sigma = \mathcal{A}[\![a_1]\!]\sigma$ $\overline{\texttt{<OP>}}$ $\mathcal{A}[\![a_2]\!]\sigma$ and $\mathcal{A}[\![a_1$ <OP> $a_2]\!]\sigma' = \mathcal{A}[\![a_1]\!]\sigma'$ $\overline{\texttt{<OP>}}$ $\mathcal{A}[\![a_2]\!]\sigma'$. By inductive hypothesis, we have that $\mathcal{A}[\![a_1]\!]\sigma = \mathcal{A}[\![a_1]\!]\sigma'$ and $\mathcal{A}[\![a_2]\!]\sigma = \mathcal{A}[\![a_2]\!]\sigma'$. So we have that $\mathcal{A}[\![a_1]\!]\sigma$ $\overline{\texttt{<OP>}}$ $\mathcal{A}[\![a_2]\!]\sigma = \mathcal{A}[\![a_1]\!]\sigma'$ $\overline{\texttt{<OP>}}$ $\mathcal{A}[\![a_2]\!]\sigma'$. Then by transitive property of equality we proved that $\mathcal{A}[\![a_1$ <OP> $a_2]\!]\sigma = \mathcal{A}[\![a_1]\!]\sigma'$ $\overline{\texttt{<OP>}}$ $\mathcal{A}[\![a_2]\!]\sigma'$.

Assignment 4

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

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

Let e, y and σ be arbitrary. 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

Consider a generic $S \in V$. By definition of V, the set S contains a finite number of elements. $P(n)$ is the property that states that all descending chains starting from sets with n elements are finite. We want to prove by strong induction over the number of elements of the set that $\forall n \geq 0 : P(n)$.

Let $|S|$ be n. By induction hypothesis we know that $n' < n \Rightarrow P(n)$. We make a case distinction:

- $n = 0$: by definition of $\subsetneq$ the descending chain is composed only by $S = \emptyset$ and so it is finite.
- $n > 0$: S is composed of at least one element. This means that in the descending chain S will be followed by a (possible empty) S' such that $S' \subsetneq S$. By induction hipothesis, we know that the descending chain starting from S' is finite. Therefore the descending chain starting from S is finite as well.

The induction principle for the relation $\subsetneq$ is as follows: $\big(\forall S.(\forall T \subsetneq S.\, P(T)\big) \Rightarrow P(S)) \Rightarrow \forall S.P(S)$, i.e., prove the property P for a set S under the assumption that P holds for any $T \subsetneq S$.

Let us use the induction principle to prove that any set $S \in V$ has $2^{|S|}$ subsets. Let S be a set in V. We have to make a case distinction:

- If $S = \emptyset$ then $2^{|S|} = 2^0 = 1$, that is the empty set itself.
- Otherwise, S contains at least one element and we can consider a $a \in S$. By induction hypothesis, we know that $S \setminus \{a\}$ has $2^{|S|-1}$ subsets $s_1, \ldots, s_{2^{|S|-1}}$. The subsets of S are all subsets of $S \setminus \{a\}$ ($s_1, \ldots, s_{2^{|S|-1}}$, that are $2^{|S|-1}$) and all these subsets unified with $\{a\}$ ($s_1 \cup \{a\}, \ldots, s_{2^{|S|-1}} \cup \{a\}$, that are $2^{|S|-1}$). Then we have $2 \cdot 2^{|S|-1} = 2^{|S|}$ subsets.