1 Datatypes and Structural Induction

As discussed in class, Haskell datatypes have corresponding structural induction principles.

1.1 Nat

The datatype definition for natural numbers was given as:

\[
dataype Nat = Zero | Succ Nat
\]

The induction principle for the type \(Nat\) is

\[
(P(Zero) \land \forall k :: Nat. P(k) \Rightarrow P(Succ k)) \Rightarrow \forall n :: Nat. P(n)
\]

Thus to show a property \(P\) holds for every finite natural number, show two things:

**Case Zero**: Show \(P(Zero)\).

**Case Succ**: For arbitrary \(k :: Nat\) assume \(P(k)\) and show \(P(Succ k)\).

Note, if we want to reason about possibly infinite elements of the type we add the case to show \(P(\bot)\). This is true for each of the induction principles below.

1.2 Lists

Recall the definition of lists containing elements of type \(a\).

\[
dataype List a = Nil | Cons a (List a)
\]

The induction principle for finite lists is

\[
(P(\text{Nil}) \land \forall x :: a. \forall xs :: List a. P(xs) \Rightarrow P(\text{Cons} x xs)) \Rightarrow \forall ys :: List a. P(ys)
\]

Thus to show a property \(P\) holds for every list, show two things:

**Case Nil**: Show \(P(\text{Nil})\).

**Case Cons**: For arbitrary \(x :: a\) and \(xs :: List a\) assume \(P(xs)\) and show \(P(\text{Cons} x xs)\).
1.3 Binary Trees

Recall the definition of the datatype for binary trees containing elements of type \( a \).

\[
\text{datatype } BTree a = \text{Leaf} | \text{Node } a (BTree a) (BTree a)
\]

The induction principle for finite binary trees is

\[
(P(\text{Leaf}) \\
\land \forall x :: a. \forall t_1, t_r :: BTree a. (P(t_1) \land P(t_r)) \Rightarrow P(\text{Node}(v, t_1, t_r)))
\Rightarrow \forall t :: BTree a. P(t)
\]

Thus to show a property \( P \) holds for every \( BTree \), show two things:

- **Case Leaf**: Show \( P(\text{Leaf}) \).
- **Case Node**: For arbitrary \( x :: a \) and \( t_1, t_r :: BTree a \) assume \( P(t_1) \) and assume \( P(t_r) \) and show \( P(\text{Node}(v, t_1, t_r)) \).

## 2 Assignment

Write the structural induction principles for the finite instances of the following types.

1. Trees having nodes with both one and with two children.

\[
\text{OneTwoTree } a = \text{Empty} | \text{Single } a (\text{OneTwoTree } a) | \text{Branch } a (\text{OneTwoTree } a) (\text{OneTwoTree } a)
\]

2. Trees whose nodes have exactly three children.

\[
\text{ThreeTree } a = \text{Leaf} | \text{Node } a (\text{ThreeTree } a) (\text{ThreeTree } a) (\text{ThreeTree } a)
\]
3. A data type for propositional formulas.

\[ \text{Formula} = \text{Bottom} \mid \text{Not Formula} \mid \text{Implies Formula Formula} \]