6.- Structural Induction and Datatypes

How do you define the following *infinite* sets?

1. $\{3, 7, 11, 15, 19, 23, \ldots\}$
2. The set of all strings formed by 0's and 1's.
3. The set of all strings on some alphabet Σ.
4. The set of all arithmetic expressions on integers with sum and product.

Inductively defined sets

To define a set S "inductively" we need to give three things:

- Basis: Specify one or more elements that are in S.
- Induction Rule: Give one or more rules telling how to construct a new element from an existing element in S.
- Closure: Specify that no other elements are in S.

- The closure is generally assumed implicitly.
- The functions that constructs the basis and the induction rules are called constructors.

Datatypes

- An inductive datatype is a type whose values are created using a fixed set of constructors.
- Each constructor has the form: `Ct(params)`
- Examples:
  - `datatype List = Nil | Cons(int, List)`
  - `datatype Tree1 = Nil | Node(Tree1, int, Tree1)`
  - `datatype Tree2 = Leaf | Node(Tree2, int, Tree2)`
  - `datatype Tree3 = Leaf(int) | Node(Tree3,Tree3)`

- Datatypes can be generic/polymorphic.
- An inductive datatype is declared as follows:

  `datatype D<`$T_1, \ldots, T_n$`> = Ctors`

  where
  - `Ctors` is a non-empty `|`-separated list of constructors and
  - $T_1, \ldots, T_n$ are type-parameters/variables.
- Examples
  - `datatype List<T> = Nil | Cons(T, List<T>)`
  - `datatype Tree1<T> = Nil | Node(Tree1<T>, T, Tree1<T>)`
  - ```
    datatype Tree2<T,S> = Leaf(S)
                        | Node(Tree2<T,S>, T, Tree2<T,S>)
    ```
    ```

- A constructor can optionally declare a destructor for any of its parameters.
  - `Cons(head: T, tail: List<T>)`
  - `Node(left: Tree1<T>, root: T, right: Tree1<T>)`
- For each constructor `Ct`, the datatype implicitly declares a boolean member `Ct?`, which returns true for those values that have been constructed using `Ct`.

```
var t0 := Tree1.Nil;
var t1 := Node(t0, 5, t0);
```

then `t1.Node?` evaluates to true and `t0.Node?` does to false, whereas `t1.Nil?` evaluates to false and `t0.Nil?` is true.

`match` statement/expression

- Likewise the if-statement/expression, `match` for inductive datatypes can be used (with different syntax) as an statement (in methods, i.p. lemmas):

```
match Expr {
  case Nil => Stmts0
  case Node(l, d, r) => Stmts1
}
```

or an expression (in functions, i.p. predicates):

```
match Expr
    case Nil => Expr0
    case Node(l, d, r) => Expr1
```

- Patterns can be nested.
- Patterns should be linear (i.e. each variable occurs exactly once in a pattern)

```
match xs
    case Nil => ...
    case Cons(h1,Nil) => ...
    // case Cons(h1,Cons(h1,t)) => ... //Non-linear pattern
    case Cons(h1,Cons(h2,t)) => ...
    case Cons(h1,Cons(h2,Cons(h3,Nil))) => ...
    case Cons(h1,Cons(h2,Cons(h3,Cons(h4,t)))) => ...
```

Defining functions over inductively defined sets

Let S be an inductively defined set.
To inductively define a function f on the set S

- Basis: For all constant constructors c of S, define $f(c)$.
- Induction: For each constructor c of arity k, define $f(c(x_1, \ldots, x_k))$ in terms of $f(x_1), \ldots, f(x_k)$ (if neccesary).

Exercises:

- Define a function that returns
    - the length of a list.
    - the number of pairs of equal consecutive elements in a list.
    - the depth of a tree.
- Define a predicate saying whether a tree is full (i.e all its nodes have zero or two children).

Structural Induction Proofs

Let S be an inductively defined set.
Let $P(x)$ be a property of x.
Claim. To show that, for all $x \in S$, $P(x)$, it suffices to show that:

- Basis: For all constant constructors c of S, $P(c)$.
- Induction: For each constructor c of arity k, show that if $P(x_1), \ldots, P(x_k)$, then $P(c(x_1, \ldots, x_k))$.

($P(x_1), \ldots, P(x_k)$ are the induction hypothesis)

Exercise: Prove that the number of pairs of equal consecutive elements in a list is less or equal its length.