# Part III: Algebraic Types

- Lists are a special instance of an *algebraic type*.
- Algebraic types are given by a number of *constructors*, which can take parameters.
- Members of an algebraic type are accessed via *pattern matching*.
- Many functional languages provide special syntax for algebraic types and pattern matching.

**Example:** Lists in Haskell (minus infix syntax, currying):

```
data List a = Nil | Cons (a, List a)

append (xs, ys) =
  case xs of
    Nil ⇒ ys
  | Cons (x, xs1) ⇒ Cons (x, append (xs1, ys))
```

# Abstract Data Types

- Algebraic types often arise naturally as implementations of abstract data types.
- An abstract data type is defined by a set of functions with their types and a set of axioms which apply to combinations of the functions.

**Example:** Sets of integers.

Name of abstract data type: IntSet.
Functions:

empty : IntSet
insert: (Int, IntSet) → IntSet
member: (Int, IntSet) → Boolean

Axioms:

member (x, empty) = false
member (x, insert (x, s)) = true
member (x, insert (y, s)) = member (x, s) **if** x != y

Note that the behavior of the type is completely defined, even though no data representation or function implementations are given.

Reference:

J. Guttag: Abstract Data Types and the Development of Data Structures. *Communications of the ACM* vol. 20, nr. 3, pages 396–404.

# IntSet's as a Data Type

An implementation of IntSets as a data type can be derived from the specification as follows:

```
data IntSet = Empty | Insert (Int, IntSet)

member (x, Empty)          = false
member (x, Insert (y, s))  = if (x == y) then true
                             else member (x, s)
```

Can this be generalized?

# General Implementation Scheme

We can often split the set of functions defined in an abstract data type into two sets:

- A set of *generators* $G$.
- A set of *accessors* $A$.

The sets should be chosen such that each axiom is of the form

$$A(G_1, ..., G_n) \quad = \quad ...$$

If all axioms can be represented in this way, an implementation in terms of an algebraic data type suggests itself:

- Every generator function $G$ becomes a constructor of the algebraic type.
- Every accessor function $A$ becomes a function with all axioms starting with $A$ as defining equations.

# Encoding Algebraic Types

How can algebraic data types be represented in Funnel?

**Idea:** Look at Church encodings:

- A type with alternatives needs to implement the corresponding **case** expression.
- The **case** expression takes one function per alternative as parameter.
- This function represents the branch corresponding to the alternative.
- In funnel, we group the set of branch functions in a record.

# Encoding Lists

A case expression for lists can be represented as a record:

```
{
    def Nil          = ...  // branch for Nil lists
    def Cons (x, xs) = ...  // branch for Cons lists
}
```

Let's assume lists are objects with a match method, which takes a record representing a case expression and invokes the right branch of this record.

Then append would be coded as follows:

```
def append (xs, ys) =
    xs.match {
        def Nil = ys
        def Cons (x, xs1) = List.Cons (x, append (xs1, ys))
    }
```

What does this remind you of?

# Constructing Lists

It remains to define how lists are constructed.

As before, we will have two constructors, Nil and Cons, wrapped in a List module.

Each constructor needs to define just the match method; everything else can then be defined in terms match.

This leads to the following structure:

```
val List = {
    def Nil          = { def match v =
    def Cons (x, xs) = { def match v =
}
```

How is this completed?

All other operations on lists can be written in terms of match.

For example:

```
def isEmpty (xs) = xs.match {
    def Nil = true
    def Cons (x, xs1) = false
}
def head (xs) = xs.match {
    def Nil = error
    def Cons (x, xs1) = x
}
def tail (xs) = xs.match {
    def Nil = error
    def Cons (x, xs1) = xs1
}
```

**Exercise:** Write implementations of map, foldl and zip which use the new representation of lists.

**Exercise:** Write an implementation of IntSet in Funnel.