

Recursion

- Simple recursion to traverse lists
- Recursive data structures and recursive functions

Recursion

At the level of abstraction at which most formal models are developed, the inefficiency of a recursive implementation is not such a major issue. As a result, recursive data types, and recursive functions to traverse them, are comparatively common.

Perhaps the simplest case in which recursion is used is in traversing a sequence data structure, e.g. to perform some operation on all of the elements of the sequence.

We can also define data structures recursively, e.g. tree structures, and again recursive functions are needed to traverse these structures.

Recursion

As a first example, consider the following type definitions from the model of a system to track the movement of containers on aircraft for an air freight business.

A flight may be modelled as a record with an identifier and a sequence of containers in the flight (representing the linear layout of containers in the hold of an aircraft):

```
Flight :: fid : FlightId
        cargo : seq of Container

Container :: content_type : <Animal> | <Food> | <Neutral>
           weight : nat
```

There are a number of functions we might wish to define on this structure. Suppose we are asked to record a restriction that the total weight of the cargo on a flight must not exceed 5000 units.

```
Flight :: fid : FlightId
        cargo : seq of Container

inv mk_Flight(fid,cargo) == TotalWeight(cargo) <= 5000
```

Recursion

Now we need to define the auxiliary function `TotalWeight`

```
TotalWeight: (seq of Container) -> nat
```

```
TotalWeight(s) ==
```

Note that we must ensure the recursion terminates (just as you always make sure that a loop terminates) and so we include a “base case” which does not lead to a recursive call of the function. Typically the base case relates to a basic element of the data type: an empty or nil value.

Recursion

Exercise:

The XOR operator is used widely in encryption: it calculates the exclusive OR of two sequences of bits. Give a recursive definition of this operator.

XOR: (seq of bool) * (seq of bool) -> (seq of bool)

What does your function do if the two sequences aren't the same length?

Recursion

Recursion is also possible in type definitions. This is particularly common where tree and graph structures are to be modelled.

Example

We often have to model representations of more abstract data structures. For example, sets and mappings are quite abstract and may not be available in an implementation language or may be inefficient to use. This example models a set of natural numbers as a binary tree to allow efficient updating and checking. Such a binary tree:

- has two (possibly nil) branches and a number at each node; and
- is arranged so that all the numbers in the left branch of a node are less than (and all the numbers in the right branch are greater than) the number in the node.

e.g.

Recursion

The binary tree structure can be modelled as follows:

```
Setrep = [Node]
Node :: left  : Setrep
      value  : nat
      right  : Setrep

inv mk_Node(left,value,right) ==
  forall lv in set gather(left) & lv < value and
  forall rv in set gather(right) & rv > value
```

To define the invariant, we used an auxiliary function which returns all the numbers stored in a given tree. The definition of this function uses recursion.

Recursion

The gather function:

```
gather: Setrep -> set of nat
```

```
gather(sr) ==
```

Observe that this function gets us from the concrete representation of the set back to its abstract counterpart. Such functions are called retrieve functions and are used to show that a concrete representation is faithful to its abstract specification.

Recursion

Exercise

Define a function `add` which, given a number and a `Setrep`, adds the number at the correct point in the `Setrep`, returning the updated `Setrep`.

Review

- Recursion is used in functions that have to traverse larger data structures such as sequences.
- When defining a recursive function, remember to ensure that there is a *base case* to guarantee termination.
- Recursive data types can also be defined to model structures such as trees and graphs. Traversal of such data structures also implies the use of a recursive function.