Lists in OCaml

• The basic data structure in OCaml
  – Lists can be of *arbitrary length*
    • Implemented as a linked data structure
  – Lists must be *homogeneous*
    • All elements have the same type

• Operations
  – Construct lists
  – Destruct them via pattern matching
Constructing Lists

Syntax

- \([\ ]\) is the empty list (pronounced “nil”)
- \(e_1::e_2\) prepends element \(e_1\) to list \(e_2\)
  - Operator \(::\) is pronounced "cons"
  - \(e_1\) is the head, \(e_2\) is the tail
- \([e_1; e_2; ...; e_n]\) is syntactic sugar for \(e_1::e_2::...::e_n::[]\)

Examples

\[
\begin{align*}
3::[] & \quad (* \text{ The list [3] } *) \\
2::(3::[]) & \quad (* \text{ The list [2; 3] } *) \\
[1; 2; 3] & \quad (* \text{ The list 1::(2::(3::[])) } *)
\end{align*}
\]
Constructing Lists

Evaluation

• [] is a value
• To evaluate \([e_1;...;e_n]\), evaluate \(e_1\) to a value \(v_1\), ...., evaluate \(e_n\) to a value \(v_n\), and return \([v_1;...;v_n]\)
Examples

# let y = [1; 1+1; 1+1+1] ;;
val y : int list = [1; 2; 3]

# let x = 4::y ;;
val x : int list = [4; 1; 2; 3]

# let z = 5::y ;;
val z : int list = [5; 1; 2; 3]

# let m = "hello"::"bob"::[ ];;
val m : string list = ["hello"; "bob"]
Typing List Construction

Nil:
[]: 'a list
i.e., empty list has type \( t \ list \) for any type \( t \)

Cons:
If \( e_1 : t \) and \( e_2 : t \ list \) then \( e_1 :: e_2 : t \ list \)

With parens for clarity:
If \( e_1 : t \) and \( e_2 : (t \ list) \) then \( (e_1 :: e_2) : (t \ list) \)

Polymorphic type: like a generic type in Java
# Examples

This expression has type `string` but an expression was expected of type `int`.

```
# let x = [1;'world'] ;;
```

This expression has type `string` but an expression was expected of type `int`.

```
# let m = [[1];[2;3]];;
val y : int list list = [[1]; [2; 3]]
```

```
# let y = 0::[1;2;3] ;;
val y : int list = [0; 1; 2; 3]
```

```
# let w = [1;2]::y ;;
This expression has type `int list` but is here used with type `int list list`
  • The left argument of `::` is an element, the right is a list
  • Can you construct a list `y` such that `[1;2]::y` makes sense?
```
Lists in Ocaml are Linked

- \([1;2;3]\) is represented above
  - A nonempty list is a pair (element, rest of list)
  - The element is the head of the list
  - The pointer is the tail or rest of the list
    - ...which is itself a list!

- Thus in math (i.e., inductively) a list is either
  - The empty list \([\ ]\)
  - Or a pair consisting of an element and a list
    - This recursive structure will come in handy shortly

CMSC 330 - Spring 2020