CPSC 429/529: Functional Programming
Carsten Schürmann
Date: January 15, 2001

Lecture 4: Lists

1 Lists

Lists are probably the most common composite data structure in Functional Programming Structure. Introduced in the 1970s, LISP was a purely functional programming language whose only mean to represent data were lists. Following the same principles as before, we introduce new datastructures in a type directed fashion. If τ is any arbitrary type, then $[\tau]$ is the type of lists of elements of τ.

$$\text{Types:} \quad \tau \quad ::= \quad \ldots \mid [\tau]$$

Next, we analyze which values are added by this new type. Again, because we are taking Haskell as programming language, we have to be concerned with lazyness, even in the representation of values.

$$\text{Values:} \quad V \quad ::= \quad \ldots \mid [] \mid P_1 \ : \ P_2$$

$[]$ is the empty list, and $P_1 \ : \ P_2$ is a list, with P_1 as head, and P_2 has tail. The interesting properties of lists are that they are lazy. For example, given a list of the form $P = P_1 \ : \ P_2$ neither P_1f nor P_2 may be values, nevertheless, the entire term is. In such a situation we may continue computation with the program P as a whole, however as soon as we look inside the data structure, we risk a no return.

Example 1.1 (Lists as Values) Here are two examples of lists in Haskell. Internally, both are represented the same way, the second however is easier to write thanks to syntactic sugar.

```
x :: [Float]
x = run 4.0 : 7.11 : []

x' :: [Float]
x' = [4.0, 7.11]
```

Haskell offers a few built-in operators, that allows the manipulation of lists. We name only a few.

$$\begin{array}{llll} \text{Programs:} & P & ::= & \ldots \mid \text{head } P \mid \text{tail } P \mid P_1 \mathbin{++} P_2 \\ & & & \mid (\text{case } P \text{ of } [] \to P_1 \mid (h : t) \to P_2) \end{array}$$

Note, that the two variables h, t are free in the second branch P_2 of the case program.

Example 1.2 (Programming with Lists) The following declarations are Haskell code.

```
-- heads and tails
hx = head x
tx = tail x

-- append
z = x ++ y

-- adding elements of a list
add x =
  case x  of
    [] -> 0.0
    h : t -> h + (add t)

-- using functions.
add' = \x ->
  case x  of
    [] -> 0.0
    h : t -> h + (add' t)

-- making it even more abstract
fold' = \l -> \c -> \f ->
  case l  of
    [] -> c
    h : l' -> f (h,  fold' l' c f)

-- where is the mistake
fold = \l -> \c -> \f ->
  case l  of
    [] -> c
    h : l' -> f h (fold l' c f)
```

As usual, after we define programs, we also have to define what the operational semantics is supposed to do. We do this by giving additional rules for the evaluation judgment $P \hookrightarrow V$.

$$\frac{P \hookrightarrow (P_1 \ : \ P_2) \qquad P_1 \hookrightarrow V_1}{\text{head } P \hookrightarrow V_1}\,\textsf{evhead} \qquad \frac{P \hookrightarrow (P_1 \ : \ P_2) \qquad P_2 \hookrightarrow V_2}{\text{tail } P \hookrightarrow V_2}\,\textsf{evtail}$$

$$\frac{}{[] \mathbin{++} P \hookrightarrow P}\,\textsf{evappnil} \qquad \frac{P_2 \mathbin{++} P \hookrightarrow V}{(P_1 \ : \ P_2) \mathbin{++} P \hookrightarrow (P_1 \ : \ V)}\,\textsf{evappcons}$$

$$\frac{P \hookrightarrow [] \qquad P_1 \hookrightarrow V}{\begin{pmatrix} \text{case } P & \\ \quad \text{of } \ [] \to P_1 & \\ \quad \ \mid \ \ (h : t) \to P_2 & \end{pmatrix} \hookrightarrow V}\,\textsf{evlistcasenil}$$

$$\frac{P \hookrightarrow (P_h : P_t) \qquad [P_h/h, P_t/t]P_1 \hookrightarrow V}{\begin{pmatrix} \text{case } P & \\ \quad \text{of } \ [] \to P_1 & \\ \quad \ \mid \ \ (h : t) \to P_2 & \end{pmatrix} \hookrightarrow V}\,\textsf{evlistcasecons}$$

Finally, we must define, what it means for lists, and the new programs to be well-typed. We extend the definition of judgment $\Gamma \vdash P : \tau$ by the following rules.

$$\frac{\Gamma \vdash P_1 : \tau \qquad \Gamma \vdash P_2 : [\tau]}{\Gamma \vdash (P_1 : P_2) : [\tau]}\,\textsf{oflist}$$

$$\frac{\Gamma \vdash P : [\tau]}{\Gamma \vdash \text{head } P : \tau}\,\textsf{ofhead} \qquad \frac{\Gamma \vdash P : [\tau]}{\Gamma \vdash \text{tail } P : [\tau]}\,\textsf{oftail}$$

$$\frac{\Gamma \vdash P_1 : [\tau] \qquad \Gamma \vdash P_2 : [\tau]}{\Gamma \vdash (P_1 \mathbin{++} P_2) : [\tau]}\,\textsf{ofapp}$$

$$\frac{\Gamma \vdash P : [\tau] \qquad \Gamma \vdash P_1 : \tau' \qquad \Gamma, h : \tau, t : [\tau] \vdash P_2 : \tau'}{\Gamma \vdash \begin{pmatrix} \text{case } P & \\ \quad \text{of } \ [] \to P_1 & \\ \quad \ \mid \ \ (h : t) \to P_2 & \end{pmatrix} : \tau'}\,\textsf{oflistcase}$$