Comp 205:
Comparative Programming Languages

Functional Programming Languages:

More Lists
- Recursive definitions
- List comprehensions

Lecture notes, exercises, etc., can be found at:
www.csc.liv.ac.uk/~grant/Teaching/COMP205/

Recursion

The type of lists is “recursively defined”:
A list is either:
- empty, or
- is an element followed by a list

Many function definitions follow this pattern:

```
length [] = 0
unzip (x : xs) = 1 + length xs
```

Recursion

Compute the sum of all the numbers in a list:

```
sumAll :: [Integer] -> Integer
```

The sum of all the numbers in the empty list is 0:

```
sumAll [] = 0
```

The sum of all the numbers in a list \( x : xs \) is \( x + \) the sum of all the numbers in \( xs \):

```
sumAll (x : xs) = x + sumAll xs
```

Recursion

Concatenate all the strings in a list of strings:

```
concat :: [[Char]] -> [Char]
```

Concatenating all the strings in the empty list gives the empty string:

```
concat [] = []
```

Concatenating all the strings in a list \( s : ss \) is \( s \) appended to the concatenation of all the strings in \( ss \):

```
concat (s : ss) = s ++ concat ss
```

Patterns

Patterns can be combined:

```
zip :: [Int] -> [String] -> [(Int, String)]
zip [] ss = []
zip is [] = []
zip (i:is) (s:ss) = (i,s) : zip is ss
```

More Patterns

Patterns can be complex:

```
unzip [] = []
unzip ((x,y) : ps) = (x:xs, y:ys)
where
    (xs,ys) = unzip ps
```
List Comprehensions

Haskell (and other languages like Miranda) provides a special notation for constructing new lists from old:

suppose \( \text{x} = \{1, 2, 3, 4, 5, 6\} \), then

\[
\{ 2 \times x \mid x \leftarrow \text{x} \}
\]

is the list \(\{2, 4, 6, 8, 10, 12\}\).

List Comprehensions

The selector can contain a pattern rather than simply a variable. For example, a function to add each pair in a List of pairs of numbers:

\[
\text{addAll} :: \{ \text{((Int, Int))} \} \rightarrow \text{[Int]}
\]

\[
\text{addAll \text{ps}} = \{ x+y \mid (x, y) \leftarrow \text{ps} \}
\]

List Comprehensions

More than one selector can be used:

\[
\{ (x, y) \mid x \leftarrow \{1, 2, 3\}, y \leftarrow \{5, 6\} \}
\]

denotes the list

\[
\{(1, 5), (1, 6), (2, 5), (2, 6), (3, 5), (3, 6)\}
\]

List Comprehensions

List comprehensions are expressions, so they can be used in definitions:

\[
\text{coords} :: \{ \text{(Int, Int)} \}
\]

\[
\text{coords} = \{ (x, y) \mid x \leftarrow \{0..9\}, y \leftarrow \{0..9\}\}
\]

\[
\text{mkCoords} :: \text{[Int]} \rightarrow \text{[Int]} \rightarrow \{ \text{(Int, Int)} \}
\]

\[
\text{mkCoords \text{xs \text{ys}}} = \{ (x, y) \mid x \leftarrow \text{xs}, y \leftarrow \text{ys} \}
\]
Summary

Key topics:

- Recursion
- List Comprehensions

Next: Higher-Order Functions