

Patterns

Example: Pattern matching on lists. Write a function `sumlist` that accepts a list of integer values and returns the sum of the integers on the list.

```
- fun sumlist ([ ]) = 0
  | sumlist(x :: xs) = x + sumlist(xs);
```

Patterns

Example: write a function that reverses a given list.

```
- fun reverse ([ ]) = [ ]
  | reverse (x :: xs) = reverse(xs) @ [x];
```

Patterns

Example: match on nested structures. Assume we have a list of persons

```
[(32,185,"married","pilot"),(28,160,"not-married","cook"),...]
```

we want to write a function that returns the age of the first person on the list.

```
- fun get1stAge ((age,weight,mstat,profession)::otherpersons) = age;
```

here we pattern match on the list as well as on the tuples that make up the list

```
- fun get1stAge (L) = #1 hd(L); } same function no patterns matching
```

Note: here we assume that the list of persons is never empty!

Anonymous Variables

Consider the following program:

```
- fun f (0) = "zero"  
  | f (x) = "non-zero";
```

The variable x is never used on the right side of the equation; bad programming practice.

We can rewrite this program using an anonymous variable:

```
- fun f (0) = "zero"  
  | f (_) = "non-zero";
```

Here we pattern match on the structure but we don't exactly care what the precise values are.

Patterns

Pattern matches can also occur in other places in functional programs.

Consider,

```
- val (age,weight,mstat,profession) = (38,185,"married","pilot");
```

pattern!

```
val age = 38 : int  
val weight = 185 : int  
val mstat = "married" : string  
val profession = "pilot" : string
```

This is very different from

```
- val joe = (38,185,"married","pilot");  
val joe = (38,185,"married","pilot") : int * int * string * string
```

Local Definitions: 'Let' Stmt

The aim is to limit the scope of a definition.

Syntax:

```
<let-expr> ::= let <definitions> in <expr>  
<definitions> ::= any valid variable or function definitions  
<expr> ::= any valid expression
```

Note: the value of <expr> before the return value of <let-expr>.

Pattern Matching with Let Stmt

Example: Given a list of elements, write a function that returns two lists,, each with halve the elements of the original list.

```
- fun halve ([ ]) = ([ ], [ ])
  | halve ([a]) = ([a], [ ])
  | halve (a::b::rest) =
    let
      val (x,y) = halve(rest)
    in
      (a::x,b::y)
    end;
```

} x and y are local variables.

Exercise

Write the function less(e,L) that returns a list of integers from the list L each of which is less than the value e.