

Introduction to SML

Disjoints Sets

Michael R. Hansen

mrh@imm.dtu.dk

Informatics and Mathematical Modelling
Technical University of Denmark

Overview

- Disjoint Sets
 - The datatype – simple version
 - case expressions

Disjoint Sets: An Example

A *shape* is either a circle, a square, or a triangle

- the union of three *disjoint* sets

A *datatype* declaration for shapes:

```
datatype shape = Circle of real
               | Square of real
               | Triangle of real*real*real;
```

Answer from the SML system:

```
> datatype shape
>   con Circle = fn : real -> shape
>   con Square = fn : real -> shape
>   con Triangle = fn : real * real * real -> shape
```

Constructors of a datatype

The *tags* **Circle**, **Square** and **Triangle** are *constructors* of values of type **shape**

```
- Circle 2.0;  
> val it = Circle 2.0 : shape  
  
- Triangle(1.0, 2.0, 3.0);  
> val it = Triangle(1.0, 2.0, 3.0) : shape  
  
- Square 4.0;  
> val it = Square 4.0 : shape
```

Equality on **shapes** is defined provided ...

```
- Triangle(1.0, 2.0, 3.0) = Square 2.0;  
> val it = false : bool
```

Constructors in Patterns

```
fun area(Circle r)          = Math.pi * r * r
  | area(Square a)          = a * a
  | area(Triangle(a,b,c)) =
      let val d = (a + b + c)/2.0
      in Math.sqrt(d*(d-a)*(d-b)*(d-c))
      end;
> val area = fn : shape -> real
```

- a constructor only matches itself

```
    area (Circle 1.2)
  ~> (Math.pi * r * r, [r ↦ 1.2])
  ~> ...
```

The **case**-expression

Form:

```
case exp of
     $pat_1 \Rightarrow e_1$ 
  |  $pat_2 \Rightarrow e_2$ 
    ...
  |  $pat_k \Rightarrow e_k$ 
```

Example:

```
fun area s =
  case s of
    (Circle r)           => Math.pi * r * r
  | (Square a)           => a*a
  | (Triangle(a,b,c)) =>
      let val d = (a + b + c)/2.0
      in Math.sqrt(d*(d-a)*(d-b)*(d-c))
      end;
```

Enumeration types – the **order** type

`datatype order = LESS | EQUAL | GREATER;`

Predefined 'compare' functions, e.g.

$$\text{Int.compare}(x, y) = \begin{cases} \text{LESS} & \text{if } x < y \\ \text{EQUAL} & \text{if } x = y \\ \text{GREATER} & \text{if } x > y \end{cases}$$

Example:

```
fun countLEG []          = (0, 0, 0)
  | countLEG(x::rest) =
      let val (y1, y2, y3) = countLEG rest in
      case Int.compare(x, 0) of
          LESS      => (y1+1, y2  , y3  )
        | EQUAL     => (y1  , y2+1, y3  )
        | GREATER   => (y1  , y2  , y3+1)
      end;
```

The option type

```
datatype 'a option = NONE | SOME of 'a;
```

Example

```
fun smallest []      = NONE
  | smallest(x::xs) =
    case smallest xs of
      NONE => SOME x
    | SOME y => if x < y then SOME x else SOME y;

> val smallest = fn : int list -> int option

- smallest [2, ~3, 6];
> val it = SOME ~3 : int option
```

smallest — continued

The predefined function **valOf**:
exception Option;

```
fun valOf(SOME x) = x  
  | valOf NONE    = raise Option;
```

```
> val 'a valOf = fn : 'a option -> 'a
```

```
- 3 + valOf(smallest [1,2,9]);
```

```
> val it = 4 : int
```