Type system features of SML

Igor Rents
RISC-Linz
irents@risc.uni-linz.ac.at

June 28, 2000

Outline of the talk

- Polymorphic functions
- Polymorphic datatypes
- Parametric types
- Structures (modules)
- Signatures
- Existential type quantification
  - On the level of values $\Longrightarrow$ abstract datatypes
  - On the level of structures $\Longrightarrow$ signatures
- Universal type quantification
  - On the level of values $\Longrightarrow$ parametrized types
  - On the level of structures(modules) $\Longrightarrow$ parametrized modules (functors)

- **Polymorphic functions**

SML	Paper
fun Id x=x	value Id=fun(x) x
> val Id=fn: 'a $\rightarrow$ 'a	Id:$\forall a.a \rightarrow a$
fun First (x,y)=x	value Id=fun(x,y) x
> val First=fn: ('a$\star$'b) $\rightarrow$ 'a	First:$\forall$ a.b. $(a \otimes b) \rightarrow a$

Table 1: Polymorphic functions.

Equality testing is not possible for functions and abstract datatypes!

Example:

fun mem x [] = false
  | mem x (y::l) = (x=y) **orelse** (mem x l);
> val mem=fn:''a * ''a list $\rightarrow$ bool

- **Polymorphic datatypes**

  datatype ('a,'b) sum= In1 of 'a | Inv2 of 'b
  In1 "a": (string,'a) sum
  [In1 "tyrant", In2 1040]:((string, int) sum) list

- **Parametric types**

SML	Paper
type 'a Pair= 'a * 'a	type Pair[T]=T$\otimes$T

Table 2: Parametric types.

- **Structure**=a collection of related declarations

- **Signature**=documents type information of a structure ("type" of a structure)

- **Functor**=maps structures into a structure

Example:

```
structure Queue1 =
  struct
  type 'a T = 'a list;
  exception E;

  val empty = [];

  fun add(q,x) = q @ [x];

  fun null(x::q) = false
    | null  = true;

  fun head(x::q) = x
    | head [] = raise E;

  fun tail(x::q) = q
    | tail [] = raise E;
end;
```

Access via compound names:
The type of queues: α Queue1.T

Queue1.E, Queue1.empty, Queue1.add, Queue1.null, Queue1.head, Queue1.tail

```
structure Queue2 =
  struct
  type 'a T = empty
    | add of 'a T * 'a;
  exception E;

  fun null(add _) = false;
  fun null empty = true;

  fun head(add(empty,x)) = x
    | head (add(q,x)) = head q
    | head empty = raise E;

  fun tail(add(empty,x)) = empty
    | tail (add(q,x)) = add(tail q,x)
    | tail empty = raise E;
end;

open Queue2;

local open Queue2 in
    fun qtake(i,q) =
    if null q orelse i¡=0 then []
    else (head q) :: qtake(i-1,tail p)
end;
```

signature QUEUE1=
  sig
  type 'a list
  exception E
  val empty:'a list
  val add:'a list * 'a → 'a list
  val null: 'a list → bool
  val head: 'a list → 'a
  val tail: 'a list → 'a list
end;

signature QUEUE2=
  sig
  type 'a T = empty | add of 'a T ⋆ 'a
  exception E
  val empty:'a T
  val add:'a T * 'a → 'a T
  val null: 'a T → bool
  val head: 'a T → 'a
  val tail: 'a T → 'a T
end;

```
signature QUEUE=
  sig
  type 'a T
  exception E
  val empty:'a T
  val add:'a T * 'a → 'a T
  val null: 'a T → bool
  val head: 'a T → 'a
  val tail: 'a T → 'a T
end;
```

Existential type quantification (abstract datatypes)

- Abstract types

```
exception BadTime;
abstype time = time of int * int
  with
    fun maketime(hrs,mins) =
        if hrs<0 orelse 23<hrs orelse
          mins<0 orelse 59<mins
        then raise BadTime
        else time(hrs,mins)
    fun hours(time(t1,t2)) = t1
    fun minutes(time(t1,t2)) = t2
end;
```

```
>type time
>val maketime = fn:int⋆int→time
>val hours=fn:time→int
>val minutes=fn:time→int
```

val t=maketime(8,30);
>**val** t = -:time
(hours t, minutes t);
> **val** it = (8,30):int⋆int

type time = (∃ time'){
maketime:int⋆int→time';
hours:time'→int
minutes:time'→int
}

- Modules

- **Structure**=a collection of related declarations

- **Signature**=documents type information of a structure