Introduction to SML
Tables

Anne E. Haxthausen

ah@imm.dtu.dk

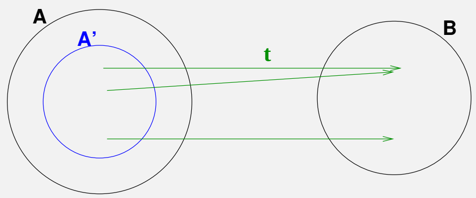
Computer Science and Engineering
Informatics and Mathematical Modelling
Technical University of Denmark

Learning Objectives

- The table concept
- An SML implementation of tables
- Modelling/programming with tables

The table concept

A *table* from a set A to a set B is a finite subset A' of A together with a function $t : A' \to B$ defined on A':



a1	b1
a2	b2
⋮	
an	bn

A' = the *domain* of t

An SML table type constructor: `table`

Signature: `type ('a,'b) table`

$\boxed{(A, B)\ \texttt{table}}$ is the type of all tables from type A to type B

Example: `(string, int) table` is a table type

One of the values in this type represents the table:

"Mon"	7
"Tue"	0
"Wed"	5

An implementation of `table` type constructor

a1	b1
a2	b2
⋮	
an	bn

A table `t` can be represented by a list of table entries:

```
[(a1, b1), ..., (an, bn)]
```

satisfying the invariant that `ai`$\neq$`aj` for `i`$\neq$`j`

Declaration of a table type constructor `table`:

```
type ('a,'b) table = ('a * 'b) list
```


An SML table module (in appendix E)

```
signature Table =
sig
   type ('a,'b) table
   exception Table
   val empty    :  (''a,'b) table
   ...
end

structure Table :> Table =
struct
   type ('a,'b) table = ('a * 'b) list
   exception Table
   val empty = []
   ...
end
```


Table operations

constructors: `empty, singleton, fromList`

modifiers: `update/insert, delete/remove, map, filter`

observers:

- non Boolean: `toList, getval/lookup, find`
- Boolean: `isKey, exists, all`

others: `fold, split`

Below we show types and sample applications of the operations.
For their implementation: see appendix E.

Table operations

```
val toList :  (''a,'b) table -> (''a * 'b) list
val fromList :  (''a * 'b) list -> (''a,'b) table
val empty :  (''a,'b) table
val singleton :  ''a * 'b -> (''a,'b) table
```

```
- load ``Table'';
- open Table;
- datatype week = M | Tu | W | Th | F | Sa | Su;
- val t0 = empty;
- toList(t0);
> val it = [] : (''c * 'd) list

- val t1 = singleton(M, 8);
- toList(t1);
> val it = [(M, 8)] : (week * int) list

- val t2 = fromList([(M, 8), (Tu, 7), (W, 0)]);
- toList(t2);
> val it = [(W, 0), (Tu, 7), (M, 8)] : (week * int) list
```

Table operations

`val update :''a * 'b * (''a,'b) table -> (''a,'b) table`

```
- toList(t2);
> val it = [(W, 0), (Tu, 7), (M, 8)] : (week * int) list

- val t3 = update(Th, 8, t2);
- toList(t3);
> val it = [(W, 0), (Tu, 7), (M, 8), (Th, 8)] : (week * int) list


- val t4 = update(Th, 0, t3);
- toList(t4);
> val it = [(W, 0), (Tu, 7), (M, 8), (Th, 0)] : (week * int) list
```


Table operations

`val delete : ''a * (''a,'b) table -> (''a,'b) table`

```
- toList(t4);
> val it = [(W, 0), (Tu, 7), (M, 8), (Th, 0)] : (week * int) list

- val t5 = delete(M, t4);
- toList(t5);
> val it = [(W, 0), (Tu, 7), (Th, 0)] : (week * int) list
```


Some table observers

`val isKey : ''a * (''a,'b) table -> bool`

`val getval : ''a * (''a,'b) table -> 'b`

`val lookup : ''a * (''a,'b) table -> 'b option`

```
- toList(t5);
> val it = [(W, 0), (Tu, 7), (Th, 0)] : (week * int) list

- isKey(Th, t5);
> val it = true : bool

- getval(Th, t5);
> val it = 0 : int

- lookup(Th, t5);
> val it = SOME 0 : int option

- isKey(M, t5);
> val it = false : bool

- getval(M, t5);
! Uncaught exception:
! Table

- lookup(M, t5);
> val it = NONE : int option
```


Table operations

`val filter : (''a * 'b -> bool) -> (''a,'b) table -> (''a,'b) table`

```
- fun isWeekend(day, hours) = day = Sa orelse day = Su;
> val 'e isWeekend = fn : week * 'e -> bool

- val t = fromList
          [(M,2), (Tu,7), (W,8), (Th,8), (F,7), (Sa,0), (Su,4)];

- val t' = filter(isWeekend)(t);
- toList(t');
> val it = [(Su, 4), (Sa, 0)] : (week * int) list
```

Other table operations

```
val map      : (''a * 'b -> 'c) -> (''a,'b) table -> (''a,'c) table

val exists   : (''a * 'b -> bool) -> (''a,'b) table -> bool

val all      : (''a * 'b -> bool) -> (''a,'b) table -> bool

val fold     : (''a * 'b *'c -> 'c) -> 'c
                                       ->  (''a, 'b) table -> 'c
val split    : (''a,'b) table ->
                  (''a * 'b * (''a,'b) table) option
val find     : (''a * 'b -> bool) -> (''a,'b) table -> (''a * 'b) option
```


Modelling using tables

Examples:

- The cash register example in section 12.1.
- The symbolic differentiation example in section 12.2.
- The part lists example in chapter 13
  (see separate collection of overheads).