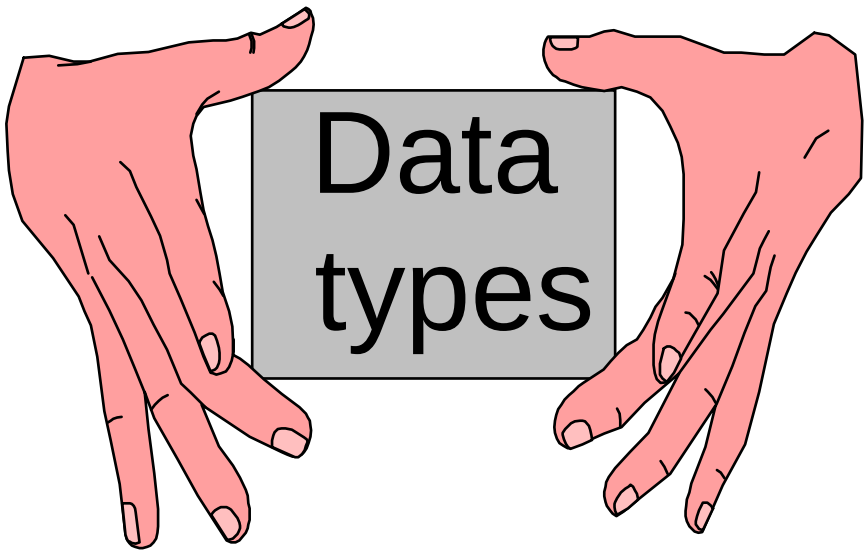
Standard ML



ML Datatypes.1

Concrete Data

- Consists of constructions that can be inspected, taken apart, or joined to form larger constructions
- Examples of concrete data:
  - Lists
    - Test whether or not empty
    - Divide (a non-empty list) into its head and tail
    - Join in new elements
  - Trees, Logical proposition, etc.
- `datatype`
  - Defines a new type along with its constructors
  - A constructor
    - In an expression: creates a value of a datatype
    - In a pattern: describe how to take a value apart

ML Datatypes.2

Types of Drinks

- Cataloguing all types of drinks
  - Coffee has a brand, number of sugar lumps and w/o milk
    - *e.g. Elite, 2 sugars, with milk*
    - **- type Coffee = string*int*bool;**
  - Wine has vineyard name.
    - *e.g Shato-Lablan*
    - **- type Wine = string*int;**
  - Beer has brand name.
    - *e.g. Paulaner.*
    - **- type Beer = string;**
  - Water is simply water. There is nothing more to say.
    - **- type Water = unit;**

    The datatype declaration for the type drink:
  - ```
    datatype drink =
        Water
      | Coffee of string*int*bool
      | Wine of string
      | Beer of string;
    ```

ML Datatypes.3

Values of a datatype

- May belong to compound data structures

```
- val drinks = [Water, Coffee (“Elite”,2,true),
                Beer “Paulaner"];
val drinks = [Water,Coffee ("Elite",2,true),Beer "Paulaner"] : drink list
```

- May be arguments and results of functions
- Remember Patterns?

- Patterns can consist
  - Constants - int, real, string, etc ...
  - Constructs - tuples, datatype constructs
  - Variables - all the rest
  - Underscore - a wildcard

Later ...

ML Datatypes.4
```

Constructors in Patterns

- Create titles for persons:

```
fun title Water = “Nature’s best Drink!"
  | title (Beer brand) = brand^“ Beer”
  | title (Wine brand) = brand^” Wine”
  | title (Coffee (brand, sugars, true)) = brand^” Coffee with “^
Int.toString(sugars)^” lumps, with milk”
  | title (Coffee (brand, sugars, false)) = brand^” Coffee with “^
Int.toString(sugars)^” lumps, no milk”;
> val title = fn : drink -> string
```

- Get brand names of drinks in a list:

```
fun pname []                = []
  | pname ((Beer s)::ps)    =   s::(pname ps)
  | pname ((Wine s) ::ps) = s::(pname ps)
  | pname ((Coffee (s,_,_)) ::ps) = s::(pname ps)
  | pname (p :: ps)         = pname ps;
> val pname = fn : drink list -> string list
```

ML Datatypes.5

Patterns in Value Declarations

- `val P = E`
  - Defines the variables in the pattern P to have the corresponding values of expression E.
- Extracting a peasant name

```
- val p = Beer “Paulaner";
- val (Beer s) = p;
Warning: binding not exhaustive
val s = " Paulaner " : string
- val (Wine s) = Beer "Paulaner";
uncaught exception nonexhaustive binding failure
```

val declaration fails if the matching fails

- What will happen for the following:

```
- val Water = Water;     (* OK *)
- val Water = “Paulaner"; (* FAIL - types mismatch *)
```

Can't ruin constructors

ML Datatypes.6

Enumeration Datatypes

- Enumeration types
  - A datatype consisting of a finite number of constants
    - `datatype bool = true | false;`
    - `fun not true  = false`
      `  | not false = true`
- But no order on the elements like Pascal, C

ML Datatypes.7

Polymorphic Datatypes

- We can use datatypes to “unite” two different types:
  - `datatype number = whole of int`
    `  |  fraction of real;`
- We can abstract on this idea, and create a datatype uniting any two types; ‘a and ‘b
  - `datatype ('a,'b)union =  type1 of 'a`
    `  | type2 of 'b;`
- Three things are declared as a result:
  - the type operator union
  - the two constructors
    - type1 : 'a -> ('a,'b)union
    - type2 : 'b -> ('a,'b)union

ML Datatypes.8

Datatype ('a,'b)union

- `('a,'b) union` is a disjoint union of 'a and 'b
  - Contains a copy of 'a OR a copy of 'b
  - **type1** and **type2** can be viewed as labels that distinguish 'a from 'b
- Allows several types where only a single type is allowed:
  - ((string,int)union)list comprises string and integers
  - `[type2 Water,type1 "Technion"] :` ((string,drink)union) list
  - `[type1 "Audi",type2 80,type1 "Technion"] :` ((string,int)union)list
  - `type1 "Technion" :` (string, 'a) union

ML Datatypes.9

The Disjoint union

- Pattern-matching can test whether `type1` or `type2` is present

```
- fun concat1 []           = ""
    | concat1 ((type1 s)::l) = s ^ concat1 l
    | concat1 ((type2 _)::l) =     concat1 l;
val concat1 = fn:(string, 'a)union list -> string
```

- The disjoint union can express any other non-recursive datatype
  - The type **drink** can be represented by:
    `((unit,string*int*bool)union,(string,string)union)sum`
  - With the following constructors:
    - `Water =type1(type1())`
    - `Coffee(b,s,m)=type1(type2(b,s,m))`
    - `Wine(b)=type2(type1(b))`
    - `Beer(b)=type2(type2(b))`

ML Datatypes.10

Trees

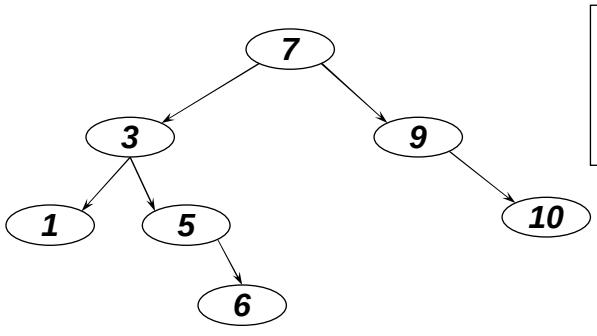
- **datatype** 'a tree =
    Lf
  | Br **of** 'a * 'a tree * 'a tree

```
- val tree2 = Br(2,Br(1,Lf,Lf),Br(3,Lf,Lf));
val tree2 = ... : int tree
- val tree5 = Br(5,Br(6,Lf,Lf),Br(7,Lf,Lf));
val tree5 = ... : int tree
- val tree4 = Br(4,tree2,tree5);
val tree4 = Br(4,Br(2,Br(1,Lf,Lf),Br(3,Lf,Lf)),
          Br(5,Br(6,Lf,Lf),Br(7,Lf,Lf))) : int tree
- fun count Lf           = 0
    | count (Br(v,t1,t2)) = 1 + count t1 + count t2;
val count = fn : 'a tree -> int
```

ML Datatypes.11

Binary Search Trees

- The search tree will hold pairs of (key,value).
  The key will be an int.
- The tree is sorted so any key on the left sub-tree is smaller than the current key (larger for the right sub-tree)



Remember: We will work on trees of pairs

ML Datatypes.12

Binary Search Trees - Search & Insert

```
exception Bsearch of string;
fun blookup(Br((a,x),t1,t2), b) =
      if      b<a then blookup(t1,b)
      else if a<b then blookup(t2,b)
                  else x
  | blookup(Lf,b) =
      raise Bsearch("lookup: "^Int.toString(b));
> val blookup = fn:(int * 'a) tree * int -> 'a

fun binsert(Lf,b, y) = Br((b,y), Lf, Lf)
  | binsert(Br((a,x),t1,t2),b,y) =
      if b<a then Br((a,x),binsert(t1,b,y),t2)
      else
      if a<b then Br((a,x),t1,binsert(t2,b,y))
      else (*a=b*)
          raise Bsearch("insert: "^Int.toString(b));
> val binsert = fn:(int * 'a)tree * int * 'a -> (int * 'a)tree
```


Question from a test (2004)

שאלה 1

סעיף א'

נתון:

```
datatype ('a,'b) union = type1 of 'a | type2 of 'b;
```

נכתוב פונקציה חד-חד-ערכית foo ב- MLשהטיפוס שלה הוא:

```
(unit, (bool->'a) * (bool->'b)) union  ->
                    (unit, ((bool,bool) union -> ('a,'b) union)) union
```

להלן הגדרת הפונקציה. השלימו את החסר בה:

```
fun foo (type1___) =      _____
  | foo (type2 (f: bool->'a, g: bool->'b) ) =      _____________________________

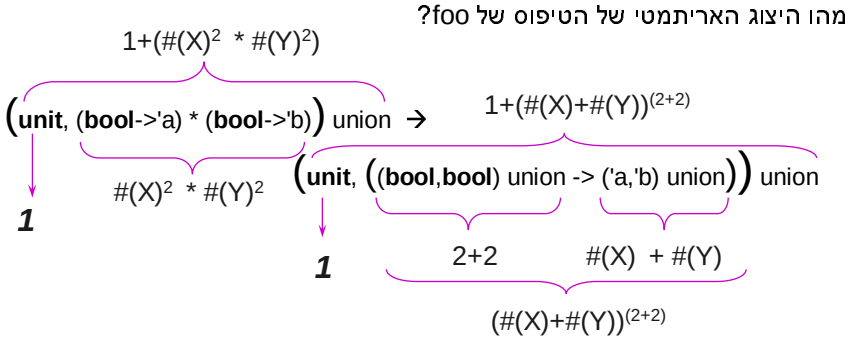
                                  _____________________________________________;

fun foo (type1()) = type1()
  | foo (type2((f: bool->'a,g: bool->'b))) =
                          type2( fn (type1(b)) => type1(p(b))
                                  |  (type2(b)) => type2(q(b)) );
```

Question from a test (2004)

סעיף ב'

מהו היצוג האריתמטי של הטיפוס של foo?



$1+(\#(X)+\#(Y))^{(2+2)}$ ^ $1+(\#(X)^2 * \#(Y)^2)$

ML Datatypes.15