Lecture VII

Data abstraction and modularity

SML Modules

Reference:

♦ Chapter 7 of *ML for the working programmer (2ND EDITION)* by L. C. Paulson. CUP, 1996.

SML Modules

Signatures and structures

♦ An abstract data type is a type equipped with a set of operations, which are the only operations applicable to that type.

Its representation can be changed without affecting the rest of the program.

♦ Structures let us package up declarations of related types, values, and functions.

♦ Signatures let us specify what components a structure must contain.

Example: Polymorphic functional stacks.

signature STACK =

sig
  exception E
  type 'a retype (* <-- INTERNAL REPRESENTATION *)
  val new: 'a retype
  val push: 'a -> 'a retype -> 'a retype
  val pop: 'a retype -> 'a retype
  val top: 'a retype -> 'a
end;

structure MyStack: STACK =
structure
  exception E ;
  type 'a retype = 'a list ;
  val new = [] ;
  fun push x s = x::s ;
  fun split( h::t ) = ( h , t )
      | split _ = raise E ;
  fun pop s = #2( split s ) ;
  fun top s = #1( split s ) ;
end ;
In SML, we can limit outside access to the components of a structure by constraining its signature in transparent or opaque manners.

Further, we can hide the representation of a type by means of an abtype declaration.

The combination of these methods yields abstract structures.
val MyHiddenEmptyStack = MyHiddenStack.new;
val MyHiddenStack0 = MyHiddenStack.push 0 MyHiddenEmptyStack;
val MyHiddenStack01 = MyHiddenStack.push 1 MyHiddenStack0;
val MyHiddenStack0’ = MyHiddenStack.pop MyHiddenStack01;
MyHiddenStack.top MyHiddenStack0’;

val MyHiddenEmptyStack = - : 'a MyHiddenStack.reptype
val MyHiddenStack0 = - : int MyHiddenStack.reptype
val MyHiddenStack01 = - : int MyHiddenStack.reptype
val MyHiddenStack0’ = - : int MyHiddenStack.reptype
val it = 0 : int

**Example:** Generic imperative stacks.

```sml
signature STACK =
  sig
    type itemtype
    val push: itemtype -> unit
    val pop: unit -> unit
    val top: unit -> itemtype
  end;
```
structure intStack
    = Stack(struct type atype = int end);

structure intStack : STACK
intStack.push(0);
intStack.top();
intStack.pop();
intStack.push(4);

val it = (): unit
val it = 0 : intStack.itemtype
val it = (): unit
val it = (): unit