A module system independent of base languages

(하위 언어에 독립적인 모듈시스템)

임현승　　박성우

2nd ROSAEC Center Workshop
9 – 11 July 2009

The ML module system

- Structures (or modules)
  - Collections of related declarations such as definitions of datatypes and associated operations
- Functors (or parameterized modules)
  - Functions from structures to structures
- Signatures and functor signatures
  - Specify interfaces to structures and functors.
- Nested modules
  - Allows modules as components.
- Higher-order functors
  - Takes functors as arguments.
- Abstract types
  - Hide the implementation details of types.

- Facilitates modular programming: flexible program construction, code reuse, data abstraction, and information hiding

Examples in Objective Caml

```
module type ORD =
 sig
  type t;                              (* abstract *)
  val compare: t -> t -> bool
 end

module type SETFUN =
 functor (Elt: ORD) ->
  sig
   type element = Elt.t                (* concrete *)
   type set                            (* abstract *)
   val  empty : set
   val add : element -> set -> set
   val member : element -> set -> bool
  end
```

```
module Ord_Int : ORD =
 struct type t = int ... end
module Ord_String : ORD =
 struct type t = string ... end
module Ord_ADT : ORD =
 struct type t = user_defined_ADT ... end

module SetList : SETFUN =
 functor (Elt : ORD) ->
  struct
   type element = Elt.t
   type set = element list
   ...
  end
module SetArray : SETFUN = ...
module SetWhatever : SETFUN = ...
```

Examples in Objective Caml

module MakeSet (SetFun : SETFUN) (Elt : ORD) = SetFun (Elt)

module intListSet = MakeSet (SetList) (Ord_Int)

Module stringListSet = MakeSet (SetList) (Ord_String)

…

- By implementing only three modules of type ORD and three functors of type SETFUN,

we have 3 x 3 = 9 set modules!!

Now suppose you are implementing a language of your own

- Want to incorporate wonderful modular programming constructs in the ML module system into your language.
- But how??
- My experience says that it is hard to understand the underlying theory of the ML module system… OTL
- Because of the interdependence of module and base languages:
  - the interaction between modules and abstract types
- Most previous work on the ML module system assume that the base language consists of terms and types.
  - What if you want to include some additional interesting features such as logical properties, dataflow graphs, …

A module system independent of base languages

- Base language = abstract declarations + abstract specifications
- Only a few assumptions on the base language
- Trade abstract types for the independence between the module system and the base language
  - Still, we provide a restricted form of abstract types, which we believe useful enough in practice.
- Ideally, our module system aims to support any base language.
- For details, …