DD2552 Seminar 9: Module types and typeclasses

Karl Palmskog

KTH

Wednesday September 27, 2023

Course material

- PFPL chapter 44
- papers to be presented(?) on Standard ML module system, typeclasses
  - MacQueen, Wadler, et al.

Homework 1 aftermath

- reference solutions coming up later this week
- aim for grading to be done by early next week
- too easy, too challenging, useful?
  - feedback welcome
  - also in course evaluation

Paper presentations

- one week to go, please flag up paper ASAP
- 10-15 minutes per student, followed by at least a few minutes of questions
- if using slides, good to aim for at most 1 slide per minute of talk
- paper need not be presented in full
  - fine to focus on some idea or even some example
  - showing code snippets often helpful

Modules

- data and functions are programming in the small
  - small pieces of functionality
  - small building blocks of data
- modules (and typeclasses) are about programming in the large
  - "separable and reusable components"
  - large utility libraries intended for reuse
  - large-scale software system construction
- modules can be **open** or **sealed**

Example: SML queue module type

```
signature QUEUE = sig
 type 'a Queue
 val empty : 'a Queue
 val isEmpty : 'a Queue -> bool
 val snoc : 'a Queue * 'a -> 'a Queue
 val head : 'a Queue -> 'a
 val tail : 'a Queue -> 'a Queue
end
```

Example: SML ordered module type

```
signature ORDERED = sig
 type T
 val eq : T * T -> bool
 val lt : T * T -> bool
 val leq : T * T -> bool
end

structure IntOrd : ORDERED = struct
 type T = int
 val eq = (=)
 val lt = (<)
 val leq = (<=)
end
```

Example: SML sortable signature

```
signature SORTABLE = sig
  structure Elem : ORDERED
  type Sortable
  val empty : Sortable
  val add : Elem.T * Sortable -> Sortable
  val sort : Sortable -> Elem.T list
end
```

Example: Haskell sortable typeclass

```
class Sortable s where
 empty :: s a
 add :: Ord a => a -> s a -> s a
 sort :: Ord a => s a -> [a]

instance Sortable T.RBTree where
 empty = T.empty
 add = T.insert
 sort = T.toList
```

Example: SML heap module type

```
signature HEAP = sig
 structure Elem : ORDERED
 type Heap
 val empty : Heap
 val isEmpty : Heap -> bool
 val insert : Elem.T * Heap -> Heap
 val merge : Heap * Heap -> Heap
 val findMin : Heap -> Elem.T
 val deleteMin : Heap -> Heap
end
```

Example: Haskell heap typeclass

```
class Heap h where
 empty     :: Ord a => h a
 isEmpty   :: Ord a => h a -> Bool
 insert    :: Ord a => a -> h a -> h a
 merge     :: Ord a => h a -> h a -> h a
 findMin   :: Ord a => h a -> a
 deleteMin :: Ord a => h a -> h a
```

Example: SML functor

```
functor SizedHeap (H : HEAP) : HEAP = struct
 structure Elem = H.Elem
 datatype Heap = NE of int * H.Heap
 val empty = NE (0, H.empty)
 fun isEmpty NE (n, h) = (n = 0)
 fun insert (x, NE (n, h)) =
  NE (n + 1, H.insert (x, h))
 fun merge (NE (n1, h1), NE (n2, h2)) =
  NE (n1 + n2, H.merge (h1, h2))
 fun findMin NE (n, h) = H.findMin h
 fun deleteMin NE (n, h) =
  NE (n - 1, H.deleteMin h)
end
```

Example : SML functor

```
functor QueueWithCons (Q : QUEUE) : QUEUE = struct
 type 'a Queue = 'a list * 'a Q.Queue
 val empty = ([], Q.empty)
 fun isEmpty ([], q) = Q.isEmpty q | isEmpty _ = false
 fun cons (x, (xs, q)) = (x::xs, q)
 fun snoc ((xs, q), x) = (h, Q.snoc (q, x))
 fun head ([], q) = Q.head q
   | head (x::xs, q) = x
 fun tail ([], q) = Q.tail q
   | tail (x::xs, q) = (xs, q)
end
```

A. Rossberg on SML modules

"ML is two languages in one: there is the core, with types and expressions, and there are modules, with signatures, structures and functors."

"Modules form a separate, higher-order functional language on top of the core. There are both practical and technical reasons for this stratification; yet, it creates substantial duplication in syntax and semantics, and it reduces expressiveness. For example, selecting a module cannot be made a dynamic decision."

https://people.mpi-sws.org/~rossberg/1ml/

First- and second-class modules

- are module definitions just expressions?
- first-class module values can depend on runtime
  - can be convenient when passing options via command line
  - see OCaml
- second-class module values are statically determined
  - reasoning and type checking much easier
  - chosen for Standard ML

Module-based libraries in production

- Standard ML Basis library (pioneer)
- OCaml Stdlib
- Jane Street Core for OCaml (industrial)
- CakeML Basis library (verified)