Modularity and Backpack

Maciek Makowski (@mmakowski)

January 1, 2015

1 ML Module System

As far as module systems go, the ML module system is often considered to offer the best combination of expressive power with good theoretical properties[Dreyer05]. It is instructive to have a brief look at this venerable system to see what Haskell is missing.

1.1 Structures

Similarly to Haskell, some of the objects that can be defined at the top level of an ML program are functions, data types, type aliases and constants. These can be grouped into modules called structures:

```
structure IntInteger =
  struct
    type integer = int
    val zero = 0
    fun add a b = a + b
    fun mul a b = a * b
  end
```

This is a definition of structure `IntInteger` which contains a type alias `integer` for `int`, a constant `zero` and two functions, `add` and `mul`.

1.2 Signatures

Structures represent concrete modules. Abstract interfaces of modules can be represented as singatures. For example, our `IntInteger` structure matches the following signature:

```
signature INTEGER =
  sig
    type integer
    val zero: integer
    fun add: integer -> integer -> integer
    fun mul: integer -> integer -> integer
```

```
end
```

The structure can then be declared to implement a given signature:

```
structure IntInteger: INTEGER = ...
```

1.3 Functors

Modules parameterised by other modules can be represented as functors. For example, given a module implementing natural numbers we can define rational numbers as follows:

```
functor RationalFun(I: INTEGER) =
  struct
    type rational = I.integer * I.integer
    fun nom r   = #1 r
    fun denom r = #2 r
    fun add r1 r2 =
      (I.add (I.mul (nom r1) (denom r2)
             (I.mul (nom r2) (denom r1)),
       I.mul (denom r1) (denom r2)))
  end
```

TODO: instantiation

2 Haskell Backpack

2.1 Packages and Modules

A direct translation of the `IntInteger` structure from the ML example to Backpack looks as follows:

```
package int-integer where
  IntInteger = [
    type Integer = Int
    zero = 0
    add a b = a + b
    mul a b = a * b
  ]
```

`int-integer` is the name of the *package* – a new concept introduced by backpack. A package consists of one ore more *modules*, which are defined using existing Haskell module syntax.

2.2 Signature Packages

```
package integer-sig where
  Integer = [
    type Integer
  ]
```

2.3 Linking

TODO
: Recursive linking possible – better than ML.

References

[Backpack] Scott Kilpatrick, Derek Dreyer, Simon Peyton Jones, Simon Marlow, *Backpack: Retrofitting Haskell with Interfaces*, `http://plv.mpi-sws.org/backpack/backpack-paper.pdf`

[Dreyer05] Derek Dreyer, *Understanding and Evolving the ML Module System*, `https://www.mpi-sws.org/~dreyer/thesis/main.pdf`