Advanced Programming Topics

Abstract Data Types - Specification and Implementation
Lecture 7

James Marshall

N.B. Abstract Data Types** are not to be confused with **Algebraic Data Types

Why use Abstract Data Types?

Abstract Data Types allow modular software to be designed. They:

- provide a clear *interface* to use the data type (typically called the *signature*, where the functions provided have a defined *syntax*)
- *hide* details of the type's implementation from the user, allowing us to subsequently modify it without breaking code that makes use of it

Axiomatic Specification of ADTs

How should we specify ADTs? We can give an informal definition of how a data structure behaves, but often this will be incomplete, and insufficient for someone without prior knowledge to implement it. The behaviour to be expected by users may also be ambiguous as a result.

With some work, however, we can formally specify the properties of an ADT. If we are careful this can provide a complete and unambiguous description of how the ADT can be used. We can also then use this specification to prove that our implementation of an ADT meets that specification.

Example 1 - Stacks

We wish to specify a *stack* ADT. The *signature* is:

- `emptyStack`
- `isEmpty`
- `top`
- `push`
- `pop`

The elements of the signature are divided into three types:

Constructors	
Observers	
Mutators	

We provide the *syntax* for the elements of the signature as follows:

- `emptyStack :: Stack a`
- `isEmpty :: Stack a -> Bool`
- `top :: Stack a -> a`
- `push :: a-> Stack a -> Stack a`
- `pop :: Stack a -> Stack a`

The *axiomatic specification* of the stack ADT is as follows:

Axiom	*Meaning*
`isEmpty emptyStack == True  --(specIsEmpty.1)`	
`isEmpty (push x s) == False --(specIsEmpty.2)`	
`top (emptyStack) == error "empty stack has no top"  --(specTop.1)`	
`top (push x s) == x  --(specTop.2)`	
`pop (emptyStack) == error "cannot pop an empty stack"  --(specPop.1)`	
`pop (push x s) == s  --(specPop.1)`	

N.B. this specification has simpler error handling than the version in Mike Stannett's notes

Example 2 - Sets

Signature:

- `emptySet`
- `isEmpty`
- `isMember`
- `addMember`
- `removeMember`
- `union`
- …

Syntax:

- `emptySet :: Set a`
- `isEmpty :: Set a -> Bool`
- `isMember :: a -> Set a -> Bool`
- `addMember :: a -> Set a -> Set a`
- `removeMember :: a -> Set a -> Set a`
- `union :: Set a -> Set a -> Set a`

Some Axioms:

```
•isEmpty emptySet == True                    --(isEmpty.1)
•isEmpty (addMember x s) == False            --(isEmpty.2)
•isMember x emptySet == False                --(isMember.1)
•isMember x (addMember x s) == True          --(isMember.2)
•isMember x (removeMember x s) == False      --(isMember.3)
•(isMember x s) == True || (isMember r s) == True =>
   isMember (union r s) == True              --(isMember.4)
•(isMember x s) == False && (isMember r s) == False =>
   isMember (union r s) == False              --(isMember.5)
•removeMember x (addMember x s) == s         --(removeMember.1)
•union emptySet emptySet == emptySet         --(union.1)
•union emptySet s == s                       --(union.2)
•union s emptySet == s                       --(union.3)
```

Q: do you think this axiomatic specification is complete
for the signature and syntax given?

Implementing ADTs in Haskell (Thompson chapters 15 and 16 for more details)

ADTs are naturally implemented in Haskell using `modules`. In a separate file we define a module, including the functions that are visible by users `import`ing that module (*i.e.* the *signature* of the ADT). For the stack example,

```
module Stack (emptyStack, isEmpty, push, pop, top) where

data MyStack a = Top a (MyStack a) | Empty

emptyStack :: MyStack a
isEmpty :: MyStack a -> Bool
push :: a -> MyStack a -> MyStack a
pop :: MyStack a -> MyStack a
top :: MyStack a -> a
```

N.B. if a module importing others wants to hide parts of them from itself and its users, it can; *e.g.*
      `import Stack hiding (pop,isEmpty)`
is possible, even if it's not that useful. This can be useful if function names clash, for example.