Advanced Programming Topics

Abstract Data Types - Specifications: Completeness, Implementation, Proof
Lecture 8
James Marshall

In lecture 7 it was claimed that "[…] *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.****"*

In this lecture we'll have a go at doing this for some simple examples… in principle it's quite simple, we just use the same kind of *equational reasoning* we already used in inductive proofs (*i.e.* rewriting Haskell equations using Haskell function definitions)

But, before we do that, let's look at whether our stack ADT specification was a *complete specification* or not. Here it is again:

```
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.2)
```

Completeness
If the specification is complete, then we should be able to prove other statements about the behaviour of the ADT using the axioms. We can compare these statements against our intuitive understanding of what the ADT should do (if we have one)

Example (from Mike Stannett's notes)
Intuitively, we know that if we have a non-empty stack, if we pop something then push it straight back on, we're left with the original stack. This isn't one of the axioms specifying the stack ADT, so can we prove it using those axioms? *I.e.* we try to prove that

```
isEmpty s == False => push (top s) (pop s) == s
```

Example 1 - Stack Implementation

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

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

emptyStack :: MyStack a
emptyStack = Empty                  --(empty)

isEmpty :: MyStack a -> Bool
isEmpty (Top x s) = False           --(isEmpty.1)
isEmpty Empty = True                --(isEmpty.2)

push :: a -> MyStack a -> MyStack a
push x s = Top x s                  --(push)

pop :: MyStack a -> MyStack a
pop (Top x s) = s                   --(pop.1)
pop Empty =                         --(pop.2)
  error "Cannot pop an empty stack”

top :: MyStack a -> a
top (Top x s) = x                   --(top.1)
top Empty =                         --(top.2)
  error "No top on an empty stack"
```

Prove that the implementation satisfies the stack ADT's axiom

```
isEmpty (push x s) == False
```

Prove that the implementation satisfies the stack ADT's axiom

```
pop (push x s) == s
```

Example 2 - Sets

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

Exercise: try to prove some of the above axioms are satisfied by the implementation on the COM2001 homepage

But wait, is our specification even complete?

Intuitively, membership of a set shouldn't depend on what order we inserted and removed things form that set. But could we prove (for example) that

```
isMember 1 (insertMember 2 (removeMember 1 emptyStack)) == False
```

using only the axioms above?