

Algebraic Specification and OBJ

Julio Mariño

Universidad Politécnica de Madrid

October 31, 2006

- Joseph A. Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsugi and Jean-Pierre Jouannaud: *Introducing OBJ*
- Martin Wirsing: *Algebraic Specification*. In Handbook of Theoretical Computer Science, vol. B: Formal Models and Semantics. Elsevier.

specification languages

- According to the approach to data modelling:

abstract model-based A set of primitive mathematical domains is assumed. Operations use (maybe implicitly) a couple of predicates to establish the connection between input and output data: pre- and post-conditions.

Some examples: Z, VDM, Larch, B, etc.

algebraic These allow for self-contained specifications independent of the representation of data. The system is decomposed into a set of abstract data types and each operation specified in terms of its relation with the others, possibly by means of an equational theory. Some examples: Clear, OBJ, ACT-ONE, etc.

- Domain-specific languages
- Calculi for concurrency

abstract data types

- **main idea:** clear separation between the interface (contract) and the implementation.
Data representation must be hidden to the users (clients) of the ADT.
- Some classical software problems have been related to not using data hiding appropriately:
 - Y2K
 - euro
- Most modern methodologies for software development are driven by data abstraction – Booch, UML, etc.

abstract data types

example: stacks

One implementation in Haskell:

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

```
data Stack a = Stack [a]
```

```
emptyStack :: Stack a  
emptyStack = Stack []
```

```
push :: a -> Stack a -> Stack a  
push x (Stack xs) = Stack (x:xs)
```

```
pop :: Stack a -> Stack a  
pop (Stack [])      = error "pop_emptyStack"  
pop (Stack (x:xs)) = Stack xs
```

```
top :: Stack a -> a  
top (Stack [])      = error "top_emptyStack"  
top (Stack (x:_)) = x
```

```
isEmpty :: Stack a -> Bool  
isEmpty (Stack xs) = null xs
```

abstract data types

example: stacks

An alternative implementation:

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

```
data Stack a = Empty | Push a (Stack a)
```

```
emptyStack :: Stack a  
emptyStack = Empty
```

```
push :: a -> Stack a -> Stack a  
push x ss = Push x ss
```

```
pop :: Stack a -> Stack a  
pop Empty      = error "pop_emptyStack"  
pop (Push x ss) = ss
```

```
top :: Stack a -> a  
top Empty      = error "top_emptyStack"  
top (Push x ss) = x
```

```
isEmpty :: Stack a -> Bool  
isEmpty Empty      = True  
isEmpty (Push x ss) = False
```

universal algebra

- Universal algebra is the area of mathematics that studies the issues common to all algebraic structures (groups, fields, rings, etc).
- An *algebra* is some set A (the carrier) along with some functions from tuples of A into A . The set of symbols used to refer to those operations is called the *signature*.
- **Example:** groups
 - ① $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
 - ② $e \cdot x = x = x \cdot e$
 - ③ $x \cdot (-x) = e = (-x) \cdot x$

algebraic specification = universal algebra + ADTs

- Some early researchers (Liskov, Guttag) realized that data abstraction was not that different from universal algebra:

Signature	\iff	Interface
equational laws	\iff	specification of observable behaviour

- Some algebraic specification languages: Clear, OBJ (a family), ACT-ONE, CASL, etc.

examples

Natural numbers:

```
obj NAT is
  sort Nat .
  op 0 : -> Nat .
  op s_ : Nat -> Nat [prec 1] .
  op _+_ : Nat Nat -> Nat .
  vars L M N : Nat .
  eq M + 0 = M .
  eq M + s N = s(M + N) .
endo
```

examples

Stacks (of natural numbers):

```
obj STACK-OF-NAT is
  sorts Stack NeStack .
  subsort NeStack < Stack .
  protecting NAT .
  op empty : -> Stack .
  op push : Nat Stack -> NeStack .
  op top_ : NeStack -> Nat .
  op pop_ : NeStack -> Stack .
  var X : Nat .
  var S : Stack .
  eq top (push(X,S)) = X .
  eq pop (push(X,S)) = S .
endo
```

semantics

what does an algebraic specification mean?

- A *model* for an algebraic specification is an algebra (carrier plus operations) that satisfies all equations.
- More than one model may exist for a given specification, so... What is the *canonical* meaning?
- Possible answer: *initial algebras*
- How can it be built? (computed?)

executing an algebraic specification

- We will be interested in proving facts about our specifications, in general, equations of the form

$$t_1 = t_2$$

where t_1 and t_2 are members of $Term(\Sigma)$, being Σ the signature of our specification.

- One way of *executing* the specification would be to compute the initial algebra for the specification and then to check whether t_1 and t_2 belong in the same equivalence class.
- This would be far too inefficient in most cases, so real systems use *Knuth-Bendix* completion to recast this problem in a rewriting setting.