→ valeurs possibles
→ opérations possibles

Algebraic Abstract Data Types: Introduction and Syntax

Didier Buchs

Université de Genève

23 septembre 2021

Algebraic Abstract Data Types

- Informal introduction
- AADT Signature
- Terms with variables
- Equations and axioms
- Examples
- Graceful presentations
- Examples

Syntax

Formal and Mathematical basis

- Algebraic view
  - heterogeneous algebra (Birkhoff) = sets + operations
  - Logical view of their properties (Horn clauses)
- Computer science
  - Type = set of data + operations
  - Some code for describing the behavior of these types
- Support of an Abstraction point of view
  - Information hiding (realization hiding)
  - Functional approach (data hiding)

Informal example : Manipulation of strings

→ ne suffit pas à soi même → on utilise pas que des string pour coder le tout

- Mandatory operations :
  fonctions
  - An empty string (new) V
  - Concatenation of two strings (append) V
  - Concatenation of one character to the string (add to) V
  - Computation of the length (size) V → int
  - Test of emptiness (isEmpty?) V → booleau
  - Equality of two strings (=) V
  - Selection of the first element (first) V
- Necessary types for defining the string abstract data type :
  - character : the character AADT
  - natural : the type of the natural numbers
  - boolean : the type of the boolean values

Signature

Definition of set of values and operations = signatures

- signatures
  - sorts names (or types)
  - operations names with profile (arity) nameofoperation : domain − > co-domain

```
Adt StringSpec;
  Interface
    sorts string, character, natural, boolean;
  Operations
     new: () -> string;
     append _ _: string, string -> string;
     add _ to _: character, string -> string;
     size _ : string -> natural;
     isEmpty? _ : string -> boolean;
     _ = _: string, string -> boolean;
     first _ : string -> character;
```

Remarks on the syntax : generalized prefix,infix and postfix notations

Prefix :

position des paramètres

prédicat

```
append _ _ : string, string -> string;
```

nom fonction

types de donné accepté par la fonction

résultat

constructible terms

```
append x y
append( x y )
(append x y)
```

Remarks on the syntax(2)

Infix :

```
_ = _: string, string -> boolean;
```

constructible terms

```
x = y
```

```
(x = y)
```

Remarks on the syntax(3)

Mixfix :

```
add _ to _: character, string -> string;
```

constructible terms

```
add append( x y) to c
add c to append( x y)
add first(x) to y
```

Remarks on the signature

Terminology :

- string is the sort of interest → principal
- character, natural et boolean are auxiliary sorts
  ↳ utilisés

Observation operations :

↳ on observe les opérations qui retournent des types auxiliaires

```
_ = _: string, string -> boolean;
size _ : string -> natural;
isEmpty? _ : string -> boolean;
first _ : string -> character;
```

Definition (Observer)

An observer is an operation with the profile :
interest sort and ev. auxiliary sorts $\rightarrow$ auxiliary sort

Remarks on the signature(2)

Modifier operations :

```
new: () -> string;
add _ to _: character, string-> string;
append _ _: string, string -> string;
```

} générateurs

Definition (Modifier)

A modifier is an operation with the profile :
interest sort and ev. auxiliary sorts → interest sort

A subclass of modifier is the operations generating all values of the domain.

$\boxed{\text{Nat: } 0 \quad s(\text{Nat})}$

Definition (Generator)

A generator is an operation with the profile :
interest sort and ev. auxiliary sorts → interest sort

Definition of basic set concepts

We recall here some usual definitions.

- **S** be universe of all sort names (type names).
- Universe are used to provide disjoint domains for sets
- two different universe are disjoint

Definition (Disjoint Union)

a disjoint union is a union where elements of the union are always considered different i.e. $\forall A, B$ sets,
$A' = A \times \{0\}, A' \cong A$
$B' = B \times \{1\}, B' \cong B$
$A \amalg B \Leftrightarrow A' \cup B'$

Example :

Definition of S-sorted set

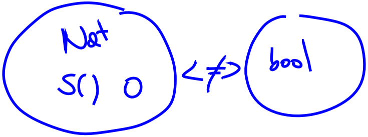
We give here some basic definitions for typing objects.

Definition (S-Sorted Set)

Let $S \subseteq \mathbf{S}$ be a finite set. A *S-sorted set* A is a disjoint union of a family of sets indexed by S ($A = \coprod_{s \in S} A_s$), noted as $A = (A_s)_{s \in S}$.

Remark : In our theory this is a disjoint partition, for non-disjoint partition there is theory of ordered sorts.

Example :



Definition of signature

Based on S-sets we have :

nous permet de décrire nos opérations

Definition (Signature)

A *signature* is a couple $\Sigma = \langle S, F \rangle$, where $S \subseteq \mathbf{S}$ is a finite set of sorts and $F = (F_{w,s})_{w \in S^*, s \in S}$ is a $(S^* \times S)$-sorted set of function names of $\mathbf{F}$. Each $f \in F_{\epsilon,s}$ is called a *constant*.

→ Sort
→ Fonctions

Example (Give the signature for stack of naturals) :

plusieurs paramètres d'entré possibles

que un type de sortie

_ _ Nat x Nat → Nat
is prime _ : Nat → bool
pair _ : Nat → bool

pop _ : stack → stack
top _ : stack → Element
push _ _ : Element, stack → stack

Definition of terms

Definition (Terms of a Signature)

Let $\Sigma = \langle S, F \rangle$ be a signature and X be a S-sorted set of variables. The set of terms of Σ over X is a S-sorted set $T_{\Sigma,X}$, where each set $(T_{\Sigma,X})_s$ is inductively defined as follows :

x∈ Nat
fonction qui prend rien en entrée

- each variable $x \in X_s$ is a term of sort s, i.e., $x \in (T_{\Sigma,X})_s$
- each constant $f \in F_{\epsilon,s}$ is a term of sort s, i.e., $f \in (T_{\Sigma,X})_s$
- for all operations that are not a constant $f \in F_{w,s}$, with $w = s_1 \ldots s_n$, and for all n-tuple of terms $(t_1 \ldots t_n)$ such that all $t_i \in (T_{\Sigma,X})_{s_i}$ $(1 \leq i \leq n)$, $f(t_1 \ldots t_n) \in (T_{\Sigma,X})_s$

What means this term ?

zero: → Nat

char String

```
add c to x = append(x y)
append (IsEmpty(new),add x to c)
```

bool String

Definition of axioms

Definition (Axioms on variables)

Let $\Sigma = \langle S, F \rangle$ be a signature and X be a S-sorted set of variables. The *axioms on variables* X are equational terms $t = t'$ such that $t, t' \in (T_{\Sigma,X})_{s}$,.

Example : $x+0 = x$ element neutre de l'addition

Remark : Variables are universally quantified

$X + S(y) = S(x+y)$

List:

empty → List

cons : _ _ : Nat, List → list

eq : _ _ : List, list → bool

→ eq $(\underbrace{empty}_{List}, \underbrace{empty}_{List})$ = true

$\boxed{\begin{array}{l} x_1, x_2 \in Nat \\ L_1 L_2 \in List \end{array}}$

→ eq (cons(x_1, L_1), cons(x_2, L_2)) = $\underbrace{eq(x_1, x_2)}_{eq\ Nat}$ and eq(L_1, L_2)

eq (cons(x_1, L_1), empty) = false

String Axioms

Axioms ⟶ liste vide

```
isEmpty?(new) = true;
isEmpty?(add c to x) = false;
#( new) = 0;
#(add c to x) = # (x) + 1;
append(new, x) = x;
append(add c to x, y) = add c to (append( x,y));
(new = new) = true;
(add c to x = new) = false;
(new = add c to x) = false;
(add c to x = add d to y) = (c = d) and (x = y);
+ axioms of first
Where
x,y:string; c,d:character;
End StringSpec;
```

vérifie si empty

⟶ not new

is empty _ : string ⟶ bool

_ : string ⟶ Int

⟶ calcul taille

comparaison

String Axioms(2)

Be carefull ! ! : The symbol $=$ is either a signature operator and a meta-operator of the basic logic of the specification language.

Signature of auxiliary sorts :

```
true: -> boolean;
false: -> boolean;
```

Gen

```
not _ : boolean -> boolean;
_ and_ ,_ or_ : boolean, boolean -> boolean ;
0: -> natural; 1: -> natural;
succ: natural -> natural;
_ + _ ,_ -_ ,_ *_ ,_ /_ : natural, natural -> natural ;
_ =_ : natural, natural -> boolean;
a: () -> character; b: () -> character;
....
_ =_ : character, character -> boolean;
```

Boolean Axioms

```
Adt Booleans;
Interface
Sorts boolean; → liste types
Operations
Gen  true , false : -> boolean;
Mod  not _ : boolean -> boolean;
op   _ and _ ,_ or _ , _ xor _ ,_ = _: boolean boolean -> boolea
Body
Axioms
not(true) = false; not(false) = true;
(true and b) = b; (false and b) = false;
(true or b) = true; (false or b) = b;
(false xor b) = b; (true xor b) = not(b);
(true = true) = true; (true = false) = false;
(false = true) = false; (false = false) = true;
Where b : boolean;
```

Exercise

Write the axioms of a sort Stack with the signature ;

```
Adt Stack;
Interface
Use Naturals, Booleans;
Sorts stack;
Operations
empty : -> stack;
push _ _: natural stack -> stack;
pop _ : stack -> stack;
top _ : stack -> natural;
_ = _ : stack stack -> boolean;
```

Exercise

Axioms of a sort Stack :

Conditional Axioms

Positive conditional axioms (Horn clause with equality) :

Definition (Axioms on variables)

Let $\Sigma = \langle S, F \rangle$ be a signature and X be a S-sorted set of variables. The *conditional axioms on variables X* are $t_1 = t'_1 \wedge ... \wedge t_n = t'_n \Rightarrow t = t'$ such that $t, t' \in (T_{\Sigma,X})_s, t_1, t'_1 \in (T_{\Sigma,X})_{s_1}, ..., t_n, t'_n \in (T_{\Sigma,X})_{s_n},$.

```
isEmpty(x)= false =>
first(add c to x) = first x;
isEmpty(x)= true =>
first(add c to x) = c;
```

si pas empty ⇝ alors

Is it necessary ?

Graceful presentations → de façon plus sure

Graceful presentations It is a method for writing axioms without :

- the possibility of writing contradictory axioms ↳ 1 = 0 → exemple
- forgetting cases. → pas complet

Principle for each operation of the signature : not(x) = ? ↳ true → false

- Write on the left of the equation a term starting with the name of this operation.
- Iterate on all parameter of the operation the following principle from left to right :
  - Use a variable for this parameter
  - If it is not possible to write a valid axiom for this variable decompose using the generators
  - If a generator is not sufficient for the decomposition in sub case use conditions

ou découpe en 2 cas disjoints

later

General Property : sufficient completeness and hierarchical consistence are guaranteed

Example of axiomatisation

`x + y = ?` → on commence par le cas basique, s'il est 'suffisant' ou le fini, sinon ou

decomposition of the second parameter with both constructors ! clarifie avec des cas différents

→ décompose

```
x + 0 = x;
x + succ(y) = succ(x+y);
```

Exercise : Application to

```
x > y = ?
```

Example of axiomatisation : Sets of naturals

check screen shots

Example of axiomatisation : Tables of naturals

Summary

- Sorts and functions
- Axioms
- Graceful Presentation
- Subtilities when algebras are not free