Program Verification: Lecture 3

José Meseguer

Computer Science Department
University of Illinois at Urbana-Champaign

Algebras

An (unsorted, many-sorted, or order-sorted) signature Σ is just syntax: provides the symbols for a language; but what is that language talking about? what is its semantics?

It is obviously talking about algebras, which are the mathematical models in which we interpret the syntax of Σ, giving it concrete meaning.

Unsorted algebras are the simplest example: children become familiar with them from the early awakenings of reason. They consist of a set of data elements, and various chosen constants among those elements, and operations on such data.

Algebras (II)

For example, for $\Sigma = (\{Nat\}, F, G)$ the `NAT-MIXFIX` signature, with $F = \{0, s, \_+\_, \_*\_\}$, we can define different algebras by:

- Choosing a set A of data elements of the algebra as the interpretation of the sort symbol Nat (just a symbol!).
- Choosing for each typed function symbol $f : Nat \overset{n}{\ldots} Nat \to Nat$ in G its interpretation as a function $f_{\mathbb{A}} : A^n \to A$.

For example:

1. $\mathbb{N}$, the algebra of natural numbers in whatever notation we wish (Peano, binary, base 10, etc.) with `0` interpreted as the zero element, `s` interpreted as successor, and `_+_` and `_*_` interpreted as natural number addition and multiplication. E.g., $6 +_{\mathbb{N}} 6 = 12$.

2. $\mathbb{N}_k$, the algebra of residue classes modulo k, for k a nonzero natural number. This is a finite algebra whose set of elements can be represented as the set $\{0, \ldots, k-1\}$. We interpret `0` as 0, and for the other operations we perform them in $\mathbb{N}$ and then take the residue modulo k. For example, in $\mathbb{N}_7$ we have $6 +_{\mathbb{N}_7} 6 = 5$.

3. $\mathbb{Z}$, the algebra of the integers, with `0` interpreted as the zero element, `s` interpreted as successor, and `_+_` and `_*_` interpreted as integer addition and multiplication.

4. $\mathbb{Q}$, the algebra of the rational numbers, with `0` interpreted as the zero element, `s` interpreted as adding 1, and `_+_` and `_*_` interpreted as rational addition and multiplication.

5. $\mathbb{R}$, the algebra of the real numbers, with `0` interpreted as the zero element, `s` interpreted as adding 1, and `_+_` and `_*_` interpreted as real number addition and multiplication.

6. $\mathbb{C}$, the algebra of the complex numbers, with `0` interpreted as the zero element, `s` interpreted as adding 1, and `_+_` and `_*_` interpreted as complex number addition and multiplication.

Similarly, for Σ the unsorted signature:

```
sort Boolean .
ops true false : -> Boolean .
op not : Boolean -> Boolean .
ops and or : Boolean Boolean -> Boolean .
```

we can define many algebras, including the following:

1. $\mathbb{B}$ the standard Boolean algebra, with just two elements, say $\{0, 1\}$, with `true` interpreted as 1 and `false` as 0 and with the standard interpretation of `not`, `and`, and `or` as Boolean operations (specified by truth tables).

2. (Powersets) for A any set, we can define on its powerset $\mathcal{P}(A)$ a Σ-algebra for this signature, by interpreting: `true` as A, `false`

as $\emptyset$, **`not`** interpreted as complement (that is, $not(B) = A \setminus B$), and with **`and`**, and **`or`** interpreted, respectively, as set intersection $\cap$ and set union $\cup$.

3. Note the fact that we can also define an algebra for the function symbols $F = \{0, s, \_ + \_, \_ * \_\}$ on $\{0, 1\}$, by interpreting 0 as 0, s as negation, $\_ + \_$ as disjunction, and $\_ * \_$ as conjunction. Likewise, on $\mathcal{P}(A)$ we could interpret 0 as $\emptyset$, s as complement, $\_ + \_$ as $\cup$, and $\_ * \_$ as $\cap$.

   The point of this last example is that function symbols do not mean anything until they are interpretes as actual functions.

General Definition of Unsorted Algebras

For Σ an unsorted signature $\Sigma = (\{s\}, F, G)$, with single sort s, an unsorted Σ-algebra is a pair $\mathbb{A} = (A, \_\_{\mathbb{A}})$, where A, called the interpretation of s, is a set specifying the data elements in the algebra, and $\_\_{\mathbb{A}}$ is a symbol interpretation function that maps:

- each constant symbol $a :\longrightarrow s$ in G to an element $a_{\mathbb{A}} \in A$

- each n-ary function symbol $f : s \stackrel{n}{\ldots} s \longrightarrow s$ in G to a function $f_{\mathbb{A}} : A^n \longrightarrow A$.

Note that we distinguish between the algebra $\mathbb{A}$ and the set A. This is because each interpretation in A of the (typed) function symbols G defines a different algebra: we can have: $(A, \_\_{\mathbb{A}}) \neq (A, \_\_{\mathbb{A}'})$. Let us see an example.

Example: Dual Boolean Algebras

For two Boolean algebras: the standard one $\mathbb{B} = (\{0,1\}, \_\_{}_{\mathbb{B}})$, and the powerset algebra $\mathbb{P}(A) = (\mathcal{P}(A), \_\_{}_{\mathbb{P}(A)})$ for A a set, we can define their corresponding dual boolean algebras, $\mathbb{B}^{\circ} = (\{0,1\}, \_\_{}_{\mathbb{B}^{\circ}})$ and $\mathbb{P}^{\circ}(A) = (\mathcal{P}(X), \_\_{}_{\mathbb{P}^{\circ}(A)})$ as follows:

- $\mathbb{B}^{\circ} = (\{0,1\}, \_\_{}_{\mathbb{B}^{\circ}})$, where $\_\_{}_{\mathbb{B}^{\circ}}$ interprets `true` as 0, `false` as 1, `not` as negation, `and` as disjunction, and `or` as conjunction.

- $\mathbb{P}^{\circ}(A) = (\mathcal{P}(A), \_\_{}_{\mathbb{P}^{\circ}(A)})$, where $\_\_{}_{\mathbb{P}^{\circ}(A)}$, interprets `true` as $\emptyset$, `false` as A, `not` as complement, `and` as $\cup$, and `or` as $\cap$.

Any Boolean algebra $\mathbb{A}$ has an order defined by:
$x \leq y \Leftrightarrow x \texttt{ or } y = y$. The dual $\mathbb{A}^{\circ}$ of $\mathbb{A}$ reverses this order: $x \leq y$ in $\mathbb{A}$ iff $y \leq x$ in $\mathbb{A}^{\circ}$. The point of this example is that many different algebras can have the same data set A.

The algebra of Arithmetic Expressions

Consider the signature $\Sigma = (\{Nat\}, F, G)$, with the usual typing on $F = \{0, s, +, *\}$. Σ defines the (prefix) grammar:

$$Nat \quad :: \quad 0 \mid s(Nat) \mid +(Nat, Nat) \mid *(Nat, Nat)$$

generating the set ("language") T_Σ of all arithmetic expressions. (Prefix notation avoids ambiguity in parsing).

Set theoretically, T_Σ can be inductively defined as the smallest set such that:

- $0 \in T_\Sigma$
- $t \in T_\Sigma \Rightarrow s(t) \in T_\Sigma$
- $t_1, t_2 \in T_\Sigma \Rightarrow +(t_1, t_2) \in T_\Sigma \ \wedge \ *(t_1, t_2) \in T_\Sigma$.

We can now use T_Σ as the data set of a Σ-algebra of arithmetic expressions $\mathbb{T}_\Sigma = (T_\Sigma, \_\_{\mathbb{T}_\Sigma})$ defined as follows:

The algebra of Arithmetic Expressions (II)

The symbol interpretation function $\_\_{\mathbb{T}_\Sigma}$ of $\mathbb{T}_\Sigma = (T_\Sigma, \_\_{\mathbb{T}_\Sigma})$ maps:

- 0 to $0_{\mathbb{T}_\Sigma} = 0 \in T_\Sigma$
- s to the unary function $s_{\mathbb{T}_\Sigma} : T_\Sigma \ni t \mapsto s(t) \in T_\Sigma$.
- $+$, resp. $*$, to the binary functions:
  $+_{\mathbb{T}_\Sigma} : T_\Sigma^2 \ni (t_1, t_2) \mapsto +(t_1, t_2) \in T_\Sigma$, resp.,
  $*_{\mathbb{T}_\Sigma} : T_\Sigma^2 \ni (t_1, t_2) \mapsto *(t_1, t_2) \in T_\Sigma$.

The most intuitive way to understand these operations is to represent t by its abstract syntax tree. Then, these are tree-building operations: $s_{\mathbb{T}_\Sigma}$ makes t a subtree of a tree with root s, and $+_{\mathbb{T}_\Sigma}$ (resp. $*_{\mathbb{T}_\Sigma}$) make t_1, t_2 subtrees of a tree with root $+$ (resp. $*$).

Term Algebra: the General Definition

Arithmetic expressions are just one example. The same construction works for any unsorted signature $\Sigma = (\{s\}, F, G)$. It defines the term Σ-algebra $\mathbb{T}_\Sigma = (T_\Sigma, \_\mathbb{T}_\Sigma)$ as follows.

(1). The set T_Σ of Σ-terms is the smallest set such that: (i) $a \in T_\Sigma$ for each constant a in F, and (ii) if $t_1, \ldots, t_n \in T_\Sigma$, then $f(t_1, \ldots, t_n) \in T_\Sigma$ for each $f : s \overset{n}{\ldots} s \longrightarrow s$ in Σ, $n \geq 1$.

(2). The symbol interpretation function $\_\mathbb{T}_\Sigma$ maps:

- each constant a in G to $a_{\mathbb{T}_\Sigma} = a \in T_\Sigma$.
- each $f : s \overset{n}{\ldots} s \longrightarrow s$ in G, $n \geq 1$, to the function: $f_{\mathbb{T}_\Sigma} : T_\Sigma^n \ni (t_1, \ldots, t_n) \mapsto f(t_1, \ldots, t_n) \in T_\Sigma$.

Again, the $f_{\mathbb{T}_\Sigma}$ for each (typed) $f \in G$, are just tree-building operations on the abstract syntax trees belonging to T_Σ.

The Algebra Defined by a Functional Module

Million-Dollar Question: What is the meaning (i.e., semantics or interpretation) of a Maude functional module `fmod` (Σ, E) `endfm`?

Million-Dollar Answer: For reasonable (Σ, E) it is an algebra, denoted $\mathbb{C}_{\Sigma/E}$, and called its canonical term algebra.

The algebra $\mathbb{C}_{\Sigma/E}$ is the most intuitive thing imaginable: it is the model the programmer has in mind and intends for his/her program `fmod` (Σ, E) `endfm`.

For example, the canonical term algebra $\mathbb{C}_{\Sigma/E}$ when $\Sigma = \{0, s, +, *\}$ and `fmod` (Σ, E) `endfm` is the `NAT-MIXFIX` module is exactly the algebra $\mathbb{N}$ of the natural numbers (in Peano notation) described in slide 3 of this lecture.

I define $\mathbb{C}_{\Sigma/E}$ first for the `NAT-MIXFIX` module, and then do so in general. First: what are the data elements of $\mathbb{C}_{\Sigma/E}$?

Constructor Terms = the Data Elements of $\mathbb{C}_{\Sigma/E}$

Recall that in the signature $\Sigma = (\{Nat\}, \{0, s, +, *\}, G)$ of `NAT-MIXFIX`, the opertors in $\Omega = (\{Nat\}, \{0, s\}, G_0)$, $(G_0 \subset G)$, were declared with the `[ctor]` declaration as data constructors.

This exactly means that the intended data elements of `NAT-MIXFIX` are precisely the Peano natural numbers `0`, `s(0)`, `s(s(0))`, ..., that is, the set T_Ω, called the constructor terms.

Q: Why are the operators 0 and s called constructors?

A: Because in the term algebra $\mathbb{T}_\Omega = (T_\Omega, \_{}_{\mathbb{T}_\Omega})$ the constant 0 and the tree-building function $s_{\mathbb{T}_\Omega}$ are used to build or construct all the Peano natural numbers as trees.

Therefore, the canonical term algebra $\mathbb{C}_{\Sigma/E}$ has the form: $\mathbb{C}_{\Sigma/E} = (T_\Omega, \_{}_{\mathbb{C}_{\Sigma/E}})$. The pending question is: How is its symbol interpretation function $\_{}_{\mathbb{C}_{\Sigma/E}}$ defined?

Properties Needed to Define $\mathbb{C}_{\Sigma/E}$

Defining the symbol interpretation function $\_{}_{\mathbb{C}_{\Sigma/E}}$ of $\mathbb{C}_{\Sigma/E} = (T_\Omega, \_{}_{\mathbb{C}_{\Sigma/E}})$ requires two properties of the equations:

```
eq N + 0 = N .
eq N + s(M) = s(N + M) .
eq N * 0 = 0 .
eq N * s(M) = N + (N * M) .
```

(1). Unique Termination. Given any Σ-term t, the repeated application in any order of the above equations to t as left-to-right simplification rules always terminates with a unique result. In particular, the Maude command "`red` t ." always terminates.

(2). Sufficient Completeness. The simplification of any Σ-term t always terminates in a constructor term. This would fail if any of the above equations had been omitted.

Defining $\mathbb{C}_{\Sigma/E}$ for `NAT-MIXFIX`

I claim that the equations E in `NAT-MIXFIX` satisfy the Unique Termination and Sufficient Completeness properties. We shall see that both properties can be checked automatically with Maude tools.

Q: Assuming these two properties, how is the symbol interpretation function $\_{}_{\mathbb{C}_{\Sigma/E}}$ defined?

A: Using the `red` command! Assuming those two properties exactly means that the process of simplifying a Σ-term t to termination with the equations E always results in a single constructor term, denoted $t!_E$. This defines a function:

$$\_!_E : T_\Sigma \ni t \mapsto t!_E \in T_\Omega$$

which is precisely the function implemented in Maude by the `red` command. How is $\_{}_{\mathbb{C}_{\Sigma/E}}$ defined? See the next slide.

Defining $\mathbb{C}_{\Sigma/E}$ for **NAT-MIXFIX** (II)

The definition of the canonical term algebra $\mathbb{C}_{\Sigma/E} = (T_\Omega, \_{\mathbb{C}_{\Sigma/E}})$ is then easy. $\_{\mathbb{C}_{\Sigma/E}}$ maps:

- 0 to $0_{\mathbb{C}_{\Sigma/E}} = 0!_E = 0 \in T_\Omega$
- s to $s_{\mathbb{C}_{\Sigma/E}} : T_\Omega \ni t \mapsto s(t)!_E = s(t) \in T_\Omega$, and
- $+$ (resp. $*$) to the function:

$$+_{\mathbb{C}_{\Sigma/E}} : T_\Omega^2 \ni (t_1, t_2) \mapsto +(t_1, t_2)!_E \in T_\Omega$$

$$(resp.\ \ *_{\mathbb{C}_{\Sigma/E}} : T_\Omega^2 \ni (t_1, t_2) \mapsto *(t_1, t_2)!_E \in T_\Omega).$$

This just means that, e.g., $s(s(0)) +_{\mathbb{C}_{\Sigma/E}} s(s(0))$ is the result returned by `red s(s(0)) + s(s(0)) .` That is, `s(s(s(s(0))))`.

Defining $\mathbb{C}_{\Sigma/E}$ in General

Let `fmod` (Σ, E) `endfm` be a functional module with unsorted signature Σ and constructor subsignature Ω, were the E satisfy: (1) Unique Termination and (2) Sufficient Completeness, so that there is a simplification function $\_!_E : T_\Sigma \ni t \mapsto t!_E \in T_\Omega$. Assume that $\forall t \in T_\Omega,\ t!_E = t$. Then, the semantics of `fmod` (Σ, E) `endfm` is the canonical term algebra $\mathbb{C}_{\Sigma/E} = (T_\Omega, \_{\mathbb{C}_{\Sigma/E}})$, where $\_{\mathbb{C}_{\Sigma/E}}$ maps:

- any constant a in Σ to $a_{\mathbb{C}_{\Sigma/E}} = a!_E \in T_\Omega$.
- any $f : s \overset{n}{\ldots} s \longrightarrow s$ in Σ, $n \geq 1$, to the function:

$$f_{\mathbb{C}_{\Sigma/E}} : T_\Omega^n \ni (t_1, \ldots, t_n) \mapsto f(t_1, \ldots, t_n)!_E \in T_\Omega.$$

Again, this has a clear, very intuitive meaning: it just means that for any $t_1, \ldots, t_n \in T_\Omega$, $f_{\mathbb{C}_{\Sigma/E}}(t_1, \ldots, t_n)$ is the result returned by the Maude command `red f(t1,...,tn) .`

Getting to Use Maude

You should begin writing functional modules of your own with syntax as exemplified in the examples in lectures. An easy and reusable way is to write such modules in files and reading them in with Maude's `in` command.

Download Maude from the Maude web page `http://maude.cs.uiuc.edu`. Read Setion 1.7 of "All About Maude" for suggestions on how beginners can become acquainted with Maude as soon as possible.

To enter a module into Maude can use cut and paste, or the "in filename" command inside Maude, and can change or list directories using Unix commands.

Some Common Mistakes

- not ending declarations for sorts, operators, etc. with a space followed by a period, e.g.,

```
sort Natural
op 0 : -> Natural.
op s : Natural -> Natural
```

- not putting enough parentheses to disambiguate expressions, e.g., `p s s 0 + 0`

- not leaving spaces between a mixfix operator and its arguments, e.g., `0+0`

Readings and Exercises

Before the next lecture try to:

- Follow the reading suggestions for beginners in 1.7 of "All About Maude," and try to get as deep as possible this way into Chapter 4.

- Continue playing with Maude. Define other functions on commonly used data types. For example, define binary trees that have natural numbers in their leaves, and define three functions: (i) tree reverse, (ii) max and min (give the biggest, resp. smallest, number stored in the tree), and (iii) insert, which inserts a number in the tree, so that numbers to its left in the tree will be smaller.

Readings and Exercises (II)

Ex.3.1. Give examples of Maude functional modules such that:

1. The module is not terminating.

2. The module is terminating, but not uniquely so; that is, one can choose a term such that, depending on the order in which the equations are applied, the simplified term obtained from simplifying the chosen term may be different.

3. The module is not sufficiently complete; that is, one can choose a non-constructor term whose simplified form is also a non-constructor term.

Readings and Exercises (III)

Ex.3.2. Let Σ be the signature:

```
sort Natural .
op 0 : -> Natural .
op s : Natural -> Natural .
```

And let $A = \{a, b, c\}$. How many different Σ-algebra structures can be defined on the set A? That is, how many different Σ-algebras of the form $\mathbb{A} = (A, \_\_{\mathbb{A}})$ are there? (Explain, and also state the total number of such algebras). Can you justify why the number comes out that way? For example, can your supposed justification predict (without having to explicitly construct them) exactly how many such algebras will there be on A if we add to the above Σ a binary function, say,

```
op _+_ : Natural Natural -> Natural .
```