Program Verification: Lecture 4

José Meseguer

Computer Science Department
University of Illinois at Urbana-Champaign

Definition of Many-Sorted Algebras

For $\Sigma = (S, F, G)$ a many-sorted signature, a many-sorted Σ-algebra is a pair $\mathbb{A} = (A, \_\__{\mathbb{A}})$, where:

1. A is a sort symbol interpretation function, choosing for each sort/type symbol $s \in S$ a corresponding data set A_s interpreting that sort. Therefore, if $S = \{s_1, \ldots, s_n\}$, then A is a function:

$$A : \{s_1, \ldots, s_n\} \ni s_i \mapsto A_{s_i} \in \{A_{s_1}, \ldots, A_{s_n}\}, \quad 1 \leq i \leq n$$

where the $A_{s_1}, \ldots, A_{s_n}$ need not be different sets.

Notation. We denote the sort interpretation function A as $A = \{A_s\}_{s \in S}$, call A an S-indexed set, and think of it as a parametric family of sets, parameterized by $s \in S$.

2. $\_\_{\mathbb{A}}$ is a function symbol interpretation function, choosing for each:

   - constant $a : \epsilon \rightarrow s$ in G an element $a_{\mathbb{A}} \in A_s$
   - function symbol $f : s_1 \ldots s_n \rightarrow s$ in G, $n \geq 1$, a function $f_{\mathbb{A}} : A_{s_1} \times \ldots \times A_{s_n} \rightarrow A_s$.

   Notation: if $w = s_1 \ldots s_n$, we write $A^w = A_{s_1} \times \ldots \times A_{s_n}$. For $f : s_1 \ldots s_n \rightarrow s$ we then write $f_{\mathbb{A}} : A^w \rightarrow A_s$.

In summary, for $\Sigma = (S, F, G)$, a Σ-algebra $\mathbb{A} = (A, \_\_{\mathbb{A}})$ interprets:

- each sort/type symbol $s \in S$ as a data set A_s
- each (typed) function symbol f as a constant or function $f_{\mathbb{A}}$ that respects its typing information in G.

Examples of Many-Sorted Algebras

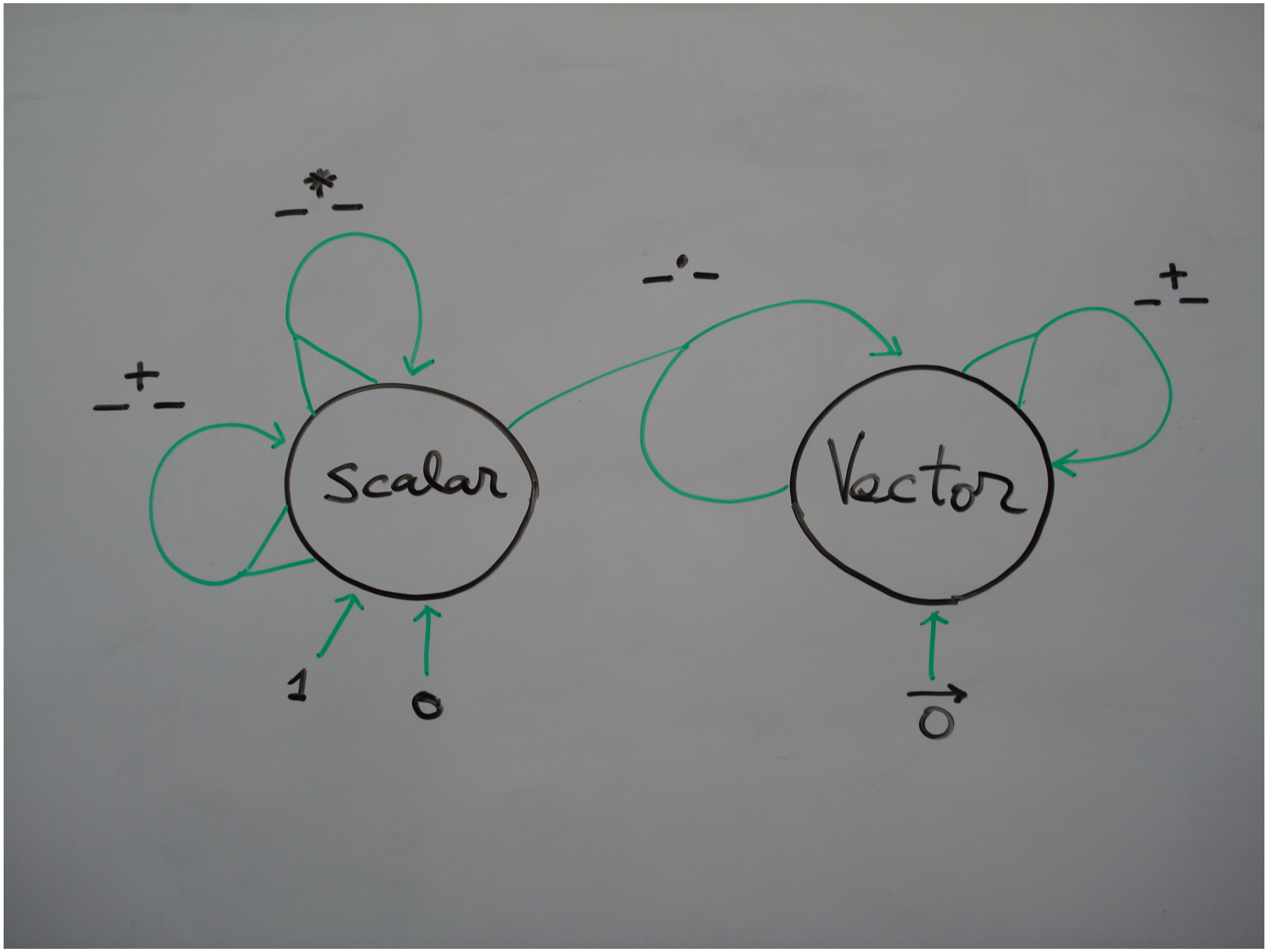
For Σ the signature of the module `NAT-LIST` we can define several algebras:

1. (Strings of naturals). We interpret the sort `Natural` as the set $\mathbb{N}$ of natural numbers, and the sort `List` as the set of strings $\mathbb{N}^*$. The interpretation function for the constants and operations is then as follows: (i) all operations in the submodule `NAT-MIXFIX` are intepreted as the algebra $\mathbb{N}$ of natural numbers; (ii) `nil` is interpreted as the empty string; (iii) `_._` is interpreted as the function that concatenates a natural number on the left of a string; and (iv) `length` is interpreted as the function measuring the length of a string.

2. (Sets of naturals). We interpret the sort `Natural` as the set $\mathbb{N}$ of natural numbers, and the sort `List` as the set $\mathcal{P}_{fin}(\mathbb{N})$ of

finite subsets of $\mathbb{N}$. The interpretation function for the constants and operations is then as follows: (i) all operations in the submodule `NAT-MIXFIX` are intepreted as the algebra $\mathbb{N}$ of natural numbers; (ii) `nil` is interpreted as the empty set $\emptyset$; (iii) `_._` is interpreted as the function inserting a natural number on a set of naturals; and (iv) `length` is interpreted as the cardinality function $|\_| : \mathcal{P}_{fin}(\mathbb{N}) \ni U \mapsto |U| \in \mathbb{N}$.

For another series of examples, consider the many-sorted signature Σ in the picture below.



The following are then examples of Σ-algebras:

1. (n-dimensional rational, real, and complex vector spaces). The sort `Scalar` is interpreted by, resp., $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$. The sort `Vector` by, resp., $\mathbb{Q}^n$, $\mathbb{R}^n$, $\mathbb{C}^n$. The operations of sort `Scalar` are interpreted on, resp., $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$, as done for the signature of `NAT-MIXFIX`. The constant 1 is intepreted as the number 1 in all cases. Vector addition is intepreted in all three cases as:

$$(x_1, \ldots, x_n) + (y_1, \ldots, y_n) =_{def} (x_1 + y_1, \ldots, x_n + y_n)$$

The constant $\vec{0}$ is interpreted as the zero vector $(0, \overset{n}{\ldots}, 0)$. The operation symbol $\_.\_$ is intepreted by the definition: $\lambda.(x_1, \ldots, x_n) =_{def} (\lambda * x_1, \ldots, \lambda * x_n)$.

2. (n-dimensional integer modules). Exactly as above, but using $\mathbb{Z}$ as scalars, and $\mathbb{Z}^n$ as vectors.

3. (n-dimensional natural semi-modules). Exactly as above, but using $\mathbb{N}$ as scalars, and $\mathbb{N}^n$ as vectors.

Definition of Order-Sorted Algebras

Given an order-sorted signature $\Sigma = ((S, <), F, G)$ an order-sorted Σ-algebra is defined as a many-sorted (S, F, G)-algebra $\mathbb{A} = (A, \_\_{\mathbb{A}})$ such that:

- In $A = \{A_s\}_{s \in S}$, if $s < s'$ then $A_s \subseteq A_{s'}$
- if f is subsort overloaded, so that we have, $f : s_1 \ldots s_n \to s$, and $f : s'_1 \ldots s'_n \to s'$, with $s_i \equiv_{\leq} s'_i$, $1 \leq i \leq n$, and $s \equiv_{\leq} s'$, then:
  - if $n = 0$, so that $s_1 \ldots s_n = s'_1 \ldots s'_n = \epsilon$, then f is a constant and we have $f_{\mathbb{A}}^{\epsilon,s} = f_{\mathbb{A}}^{\epsilon,s'}$ (subsort overloaded constants coincide)
  - otherwise, if $w = s_1 \ldots s_n$ and $w' = s'_1 \ldots s'_n$, if $(a_1, \ldots, a_n) \in A^w \cap A^{w'}$, then $f_{\mathbb{A}}^{w,s}(a_1, \ldots, a_n) = f_{\mathbb{A}}^{w',s'}(a_1, \ldots, a_n)$ (subsort overloaded operations agree on shared data)

Examples of Order-Sorted Algebras

For Σ the signature of `NAT-LIST-II` we can define, among others, two different order-sorted algebra structures:

1. Interpret the sort **`NzNatural`** as $\mathbb{N}_{>0}$, **`Natural`** as $\mathbb{N}$, **`s`**, **`p`**, and $\_+\_$ in the usual way, **`NeList`** as $\mathbb{N}^+$, **`List`** as $\mathbb{N}^*$, **`nil`** as the empty string ϵ, $\_.\_$ as left concatenation with a natural, and **`first`**, **`rest`** and **`length`** in the usual way.

2. We can instead interpret both **`NzNatural`** and **`Natural`** as $\mathbb{Z}$, **`s`**, **`p`**, and $\_+\_$ as those functions extended to all integers, **`NeList`** as $\mathbb{Z}^+$, **`List`** as $\mathbb{Z}^*$, **`nil`** as the empty string ϵ, $\_.\_$ as left concatenation with an integer, **`first`**, **`rest`** and **`length`** in the usual way.

Order-Sorted Term Algebras

For $((S,<),F,G)$ an order-sorted signature, an obvious Σ-algebra is the term algebra $\mathbb{T}_\Sigma = (T_\Sigma, \_{\mathbb{T}_\Sigma})$, where the family of data sets $T_\Sigma = \{T_{\Sigma,s}\}_{s\in S}$ and its symbol interpretation function $\_{\mathbb{T}_\Sigma}$ are mutually defined in an inductive way by:

- for each $a : \epsilon \to s$ in Σ, $a_{\mathbb{T}_\Sigma} = a \in T_{\Sigma,s}$
- for each $f : w \to s$ in Σ, with $w = s_1 \ldots s_n$, $n > 0$, the function $f_{\mathbb{T}_\Sigma} : T_{\Sigma,s_1} \times \ldots \times T_{\Sigma,s_n} \to T_{\Sigma,s}$ maps the tuple $(t_1,\ldots,t_n) \in T_\Sigma^w$ to the expression (called a term) $f(t_1,\ldots,t_n) \in T_{\Sigma,s}$
- if $s < s'$, then $T_{\Sigma,s} \subseteq T_{\Sigma,s'}$

Examples of Terms for the **NATURAL** Specification

$T_{\texttt{NATURAL},\texttt{NzNatural}} =$
$\{\texttt{s 0}, \texttt{s s 0}, \texttt{s s s 0}, \texttt{s p s 0}, \texttt{s(0 + s 0)}, \ldots\}$

$T_{\texttt{NATURAL},\texttt{Natural}} = T_{\texttt{NATURAL},\texttt{NzNatural}} \cup \{\texttt{0}, \texttt{p s 0}, (0+0), \ldots\}.$

Although the mathematical definition of terms uses prefix notation, Maude allows general mixfix notation. This is just a (very useful) parsing and pretty-printing facility. If one insists (by giving the command `set print mixfix off .`) Maude can print even mixfix terms in prefix notation. For example, `s_(_+_(0,s_(0)))` instead of `s(0 + s 0)`.

The Algebra Defined by a Functional Module

Consider a functional module `fmod` (Σ, E) `endfm` with (Σ, E) order-sorted and $\Omega \subseteq \Sigma$ the constructor subsignature.

In the unsorted case we saw that, under reasonable assumptions on E, the meaning (i.e., semantics) of `fmod` (Σ, E) `endfm` is its canonical term algebra $\mathbb{C}_{\Sigma/E}$. We can now explain the more general case when (Σ, E) is order-sorted.

As before, the constructors Ω define the data elements of `fmod` (Σ, E) `endfm` belonging to the constructor term algebra $\mathbb{T}_\Omega = (T_\Omega, \_\mathbb{T}_\Omega)$. Instead, all the Σ-terms belong to the term algebra $\mathbb{T}_\Sigma = (T_\Sigma, \_\mathbb{T}_\Sigma)$. In `fmod` (Σ, E) `endfm`, Σ-terms should evaluate to constructor terms (data values) in T_Ω. But, under what conditions on E can we define $\mathbb{C}_{\Sigma/E}$?

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 three properties of E:

(1). Unique Termination. For any Σ-term t, repeatedly applying the equations E to t as left-to-right simplification rules in any order always terminates with a unique result, denoted $t!_E$. I.e., the Maude command "`red` t ." always terminates.

(2). Sufficient Completeness. Simplification of any Σ-term t always terminates in a constructor term $t!_E \in T_{\Omega}$.

(3). Sort Preservation. If $t \in T_{\Sigma,s}$, $s \in S$, then $t!_E \in T_{\Omega,s}$. This property holds automatically in the unsorted and many-sorted cases, but may fail for (Σ, E) order-sorted.

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

Properties (1)–(3) will allow us to define $\mathbb{C}_{\Sigma/E}$. To see why this is so, we need the notion of an S-indexed function:

Given two S-indexed sets $A = \{A_s\}_{s \in S}$, and $B = \{B_s\}_{s \in S}$, an S-indexed function f from A to B is an S-indexed set $f = \{f_s\}_{s \in S}$ such that for each $s \in S$, f_s is a function $f_s : A_s \longrightarrow B_s$. We then write $f : A \longrightarrow B$.

By Unique Termination, Sufficient Completeness and Sort Preservation, for each $s \in S$ we have a function $\_!_{E,s} : T_{\Sigma,s} \ni t \mapsto t!_E \in T_{\Omega,s}$. That is, an S-indexed function:

$$\_!_E : T_\Sigma \to T_\Omega$$

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

Defining $\mathbb{C}_{\Sigma/E}$ (II)

Let `fmod` (Σ, E) `endfm` be a functional module with order-sorted signature Σ and constructor subsignature Ω, were the E satisfy properties (1)–(3). Thus, we have an S-indexed function $\_!_E : T_\Sigma \to T_\Omega$. Assume $\forall t \in T_\Omega,\ t!_E = t$. 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 $a :\to s$ in Σ to $a_{\mathbb{C}_{\Sigma/E}} = a!_E \in T_{\Omega,s}$.
- any $f : w \longrightarrow s$ in Σ, $|w| = n \geq 1$, to the function:

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

Therefore, for any $(t_1, \ldots, t_n) \in T_\Omega^w$, $f_{\mathbb{C}_{\Sigma/E}}(t_1, \ldots, t_n)$ is the result returned by the Maude command `red f(t1,...,tn) .` For $\Sigma =$ `NAT-LIST`, $\mathbb{C}_{\Sigma/E}$ is the algebra defined in pg. 3 (1).

Maude Programming = Mathematical Modeling

The slogan:

$$\text{Maude Programming} = \text{Computable Mathematical Modeling}$$

sounds good. But what does it really mean? Is it really true?

Yes, it is true. When you write a Maude functional module `fmod` (Σ, E) `endfm` meeting conditions (1)–(3), what you do is exactly to define a mathematical model, namely, the Σ-algebra $\mathbb{C}_{\Sigma/E}$. This model is furthermore computable using Maude's `red` command: is a computable algebra.

$\mathbb{C}_{\Sigma/E}$ is precisely the model you had in mind when you wrote `fmod` (Σ, E) `endfm`. You wanted to define some data and some functions on that data. That's exactly what $\mathbb{C}_{\Sigma/E}$ is.

Sensible Signatures

A signature Σ can be intrinsically ambiguous, so that a term may denote two completely different things. Consider for example the following many-sorted signature:

```
sorts A B C D .
op a : -> A .
op f : A -> B .
op f : A -> C .
op g : B -> D .
op g : C -> D .
```

The term `g(f(a))` is an ambiguous term of sort `D` denoting two completely different things.

A mild condition ruling this out, yet allowing ad-hoc overloading, is the notion of a sensible signature, namely one such that whenever we have $f : s_1 \ldots s_n \longrightarrow s$ and $f : s'_1 \ldots s'_n \longrightarrow s'$, then $(s_1 \equiv_{\leq} s'_1 \wedge \ldots \wedge s_n \equiv_{\leq} s'_n) \Rightarrow s \equiv_{\leq} s'$.

Sensible Signatures (II)

Lemma. If Σ is a sensible order-sorted signature, then for any term t in T_Σ we have,

$$t \in T_{\Sigma,s} \ \wedge \ t \in T_{\Sigma,s'} \quad \Rightarrow \quad s \equiv_{\leq} s'$$

Proof: By induction on the depth of t.

We define the depth of a term as follows: constants have depth 0, and terms of the form $f(t_1, \ldots, t_n)$ have depth $1 + max(depth(t_1), \ldots, depth(t_n))$.

For depth 0, $t = a$ is a constant, and $a \in T_{\Sigma,s}$ iff there is $a : nil \to s''$ in Σ with $s'' \leq s$. Similarly, if $a \in T_{\Sigma,s'}$ there is $a : nil \to s'''$ in Σ with $s''' \leq s'$. By Σ sensible we have $s'' \equiv_{\leq} s'''$, and therefore, $s \equiv_{\leq} s'$.

Sensible Signatures (III)

Assuming the result true for depth $\leq n$, let $t = f(t_1, \ldots, t_n)$ have depth $n+1$. If we have $t \in T_{\Sigma,s} \ \wedge \ t \in T_{\Sigma,s'}$, this forces the existence of $f : w'' \longrightarrow s''$ and $f : w''' \longrightarrow s'''$, with $s'' \leq s$ and $s''' \leq s'$ and such that $(t_1, \ldots, t_n) \in T_{\Sigma}^{w''} \cap T_{\Sigma}^{w'''}$.

By the induction hypothesis this forces $w'' \equiv_{\leq} w'''$, where if $w'' = s''_1 \ldots s''_n$ and $w''' = s'''_1 \ldots s'''_n$, the notation $w'' \equiv_{\leq} w'''$ abbreviates the conjunction $s''_1 \equiv_{\leq} s'''_1 \wedge \ldots \wedge s''_n \equiv_{\leq} s'''_n$. And by Σ sensible this forces $s'' \equiv_{\leq} s'''$, and therefore, $s \equiv_{\leq} s'$. q.e.d.

Preregular Signatures

A sensible order-sorted signature $\Sigma = ((S, <), F, G)$ is called preregular iff for each Σ-term t (possibly with variables X), the set of sorts

$$Sorts(t) = \{s \in S \mid t \in T_{\Sigma(X),s}\}$$

includes a least element of such set in the poset $(S, <)$, called the least sort of t and denoted $ls(t)$. That is:

$$ls(t) \in Sorts(t) \ \ \wedge \ \ \forall s' \in Sorts(t), \quad ls(t) \leq s'.$$

Maude automatically checks the preregularity of the signature Σ of any module entered by the user and issues a warning if Σ is not preregular.

Kind-Complete Order-Sorted Signatures

Terms in an order-sorted signature Σ are given the benefit of the doubt if we extend Σ to a signature $\Sigma^{\square}$ by: (i) adding a new sort $[s]$, called a kind, to each connected component $[s]$, with, $(\forall s' \in [s])\ [s] > s'$, and (ii) lifting each operator $f : s_1 \ldots s_n \to s$, $n \geq 1$, to the kind level as: $f : [s_1] \ldots [s_n] \to [s]$.

Example. Let Σ have sorts $NzNat$ and Nat with $NzNat < Nat$, constant 0 of sort Nat and operators $s : Nat \to NzNat$ and $p : NzNat \to Nat$. The term $p(p(s(s(0))))$ does not parse in Σ. But it parses in its kind completion $\Sigma^{\square}$, that adds: (i) a kind $[Nat]$, with $[Nat] > Nat$, and operators $s : [Nat] \to [Nat]$ and $p : [Nat] \to [Nat]$.

Σ is called kind-complete if it has already been completed that way, i.e., if is of the form: $\Sigma = \Sigma_0^{\square}$ for some $\Sigma_0 \subseteq \Sigma$.

Variables

Note that in our definition of Σ-terms we only allowed constants and terms built up from them by other operation symbols, so-called ground terms. Therefore, terms with variables, such as those appearing in the equations

```
vars N M : Natural .
eq N + 0 = N .
eq N + s M = s(N + M) .
```

do not seem to fall within our definition. What can we say about such terms? First, note that `N` and `M` are variables in the mathematical sense, not at all in the sense of variables in an imperative language. Second, we can reduce the notion of terms with variables to that of terms without variables (ground terms) in an extended signature.

A Sample Extended Signature

We can extend the signature of our above example by adding the variables as additional constants to get the new signature,

```
sort Natural .
op 0 : -> Natural .
op N : -> Natural .
op M : -> Natural .
op s_ : Natural -> Natural .
op _+_ : Natural Natural -> Natural .
```

in which a term such as `s(N + M)` is now a well-defined term of sort `Natural`.

The Extended Signature $\Sigma(X)$

The general way of extending a signature $\Sigma = ((S, <), F, G)$ with variables is as follows. We assume a family $X = \{X_s\}_{s \in S}$ of sets of variables for the different sorts $s \in S$ in the signature Σ. Such that:

- variables of different sorts are different, i.e., $X_s \cap X_{s'} = \emptyset$ if $s \neq s'$
- the variables in X are different from the constants in Σ, i.e., $(\cup_{s \in S} X_s) \cap \{a \mid \exists s \in S,\ (a : \epsilon \to s) \in G\} = \emptyset$.

Then we define $\Sigma(X) = ((S, <), F(X), G(X))$, where: $F(X) = F \uplus X$, and $G(X) = G \uplus \{x : \epsilon \to s \mid x \in X_s \wedge s \in S\}$. I.e., we just add to Σ each $x \in X_s$ as a constant $x : \epsilon \to s$.

The Term Algebra $\mathbb{T}_{\Sigma(X)}$

Therefore, Σ-terms with variables in X are the elements of the term algebra $\mathbb{T}_{\Sigma(X)}$ associated to the extended signature $\Sigma(X)$.

Note that if Σ is a sensible signature, then it is trivial to check that $\Sigma(X)$ is also, by construction, a sensible signature. Therefore, all the results holding for ground terms in sensible signatures do hold likewise for terms with variables.

One can likewise prove that if Σ is preregular, then $\Sigma(X)$ is also preregular.

Substitutions

For an order-sorted signature $\Sigma = ((S, <), F, G)$ and S-indexed families of variables $X = \{X_s\}_{s \in S}$, and $Y = \{Y_s\}_{s \in S}$, a substitution is an S-indexed family of functions of the form:

$$\theta : X \longrightarrow T_{\Sigma(Y)}$$

For example, for Σ an unsorted signature of arithmetic expressions, $X = \{x, y, z\}$, and $Y = \{x, y, z, x', y', z'\}$, a particular θ can be the assignment:

- $x \mapsto (x + y') * z$
- $y \mapsto (x' - y')$
- $z \mapsto z' * z'$

Notation: $\theta = \{x \mapsto (x + y') * z,\ y \mapsto (x' - y'),\ z \mapsto z' * z'\}$.

Substitutions Extend to Terms

If Σ is a sensible signature, a substitution $\theta : X \longrightarrow T_{\Sigma(Y)}$ extends in a unique way to an S-indexed function:

$$\_\theta : T_{\Sigma(X)} \longrightarrow T_{\Sigma(Y)}$$

defined recursively by:

- $x\theta = \theta(x)$
- $f(t_1, \ldots, t_n)\theta = f(t_1\theta, \ldots, t_n\theta)$

For example, for the above θ we have,

$$(x + (y * z))\theta = ((x + y') * z) + ((x' - y') * (z' * z')).$$