

Algebraic Specification –
A Formalism to Specify
Sequential Programs

Markus Roggenbach

October 2002

Contents

1	Introduction	2
1.1	Formal Methods in Software Design	2
1.2	Elements of Formal Specifications	4
1.3	Bibliographic Remarks	8
1.4	Mathematical Preliminaries	8
2	Many-Sorted-Algebras	9
2.1	Signatures	9
2.2	Algebras	14
2.3	Terms	20

1 Introduction

1.1 Formal Methods in Software Design

Introduction 3

Formal Methods in Software Design

"Use of mathematics in software development"

main activities:

- **writing** formal specifications
- **proving** properties about formal specifications
- **constructing** a program by mathematical manipulating a formal specification
- **verifying** a program by mathematical argument

M. Roggenbach: Algebraic Specification, October 2002 118

Formal Methods in Software Design 4

Non Formal, Semi Formal, Formal

"It has been widely accepted that **syntax** can be mathematically defined for quite some time, but there has been more resistance to the mathematical definition of **semantics**."

(quoted freely from [?])

M. Roggenbach: Algebraic Specification, October 2002 118

Formal Methods in Software Design 5

non formal:
in natural language
(open to arbitrary new symbols)

formal:
in a (fixed) language with
mathematically defined **Syntax** and **Semantics**

M. Roggenbach: Algebraic Specification, October 2002 118

Formal Methods in Software Design 6

semi formal:
in a language with

- **Syntax** definition by mathematical methods
- **Semantics** definition in natural language or by tool

M. Roggenbach: Algebraic Specification, October 2002 118

Formal Methods in Software Design 7

Specifications

Specification: "description by properties"

Main question on specifications:
"What happens if . . ."

Specifications should be

- complete
- precise
- consistent (no contradictions)

M. Roggenbach: Algebraic Specification, October 2002 118

Formal Methods in Software Design 8

Why formal Specifications?

- formal specifications are **precise**
(non formal and sometimes even semi formal specifications are open to re-interpretation)
- syntactical and semantical **correctness**
independent of tools
- **mathematical methods**
(consistency, completeness)

M. Roggenbach: Algebraic Specification, October 2002 118

Formal Methods in Software Design 9

Limitations of Formal Methods

"The world is not a formal system."

I. Modelling means Abstraction
(only "essentials" are considered)

II. Errors within Formalisms.

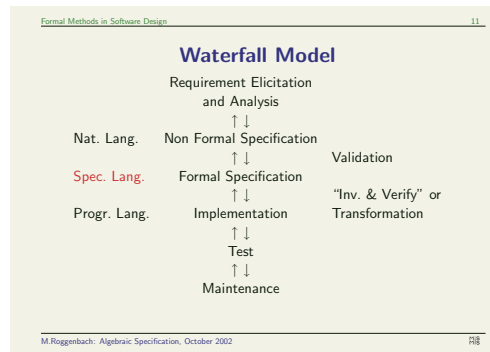
M. Roggenbach: Algebraic Specification, October 2002 118

Formal Methods in Software Design 10

III. Behaviour of a Program depends on

- Compiler
- Operating System
- Computer Hardware
- Embedding in a Technical Process
- Human Operator

M. Roggenbach: Algebraic Specification, October 2002 118



1.2 Elements of Formal Specifications

Formal Methods in Software Design 12

Specification Languages

"No single technique is adequate to address all issues of complex system development."

Classification of Specification Languages:

- Model-oriented: Z, VDM
- Property-oriented: Larch, OBJ, CASL
- Process algebras: CCS, CSP, π -calculus

M. Roggenbach: Algebraic Specification, October 2002 118

Introduction 13

Elements of Formal Specifications

```

graph TD
    A[Non Formal Specification] <--> B[Formal Specification]
    B <--> C[Implementation]
  
```

extract the "essentials" of

- a non formal specification
- all desirable implementations

M. Roggenbach: Algebraic Specification, October 2002 118

Elements of Formal Specifications 14

Running Example: Database

Non Formal Specification:

Write a Java program that implements a database with

- "Name" and
- "Telephone Number"

as entries.

M. Roggenbach: Algebraic Specification, October 2002 118

Elements of Formal Specifications 15

Implementation by Lists

[Start Database](#)

[Show Source](#)

M. Roggenbach: Algebraic Specification, October 2002 118

Elements of Formal Specifications 16

Implementation by Sorted Binary Trees

[Show Source](#)

[Compile Database](#)

M. Roggenbach: Algebraic Specification, October 2002 118

Elements of Formal Specifications 17

Essential for Programs

distinguish between functions of the

- **interface**
(functions that can be used "safely")
- **implementation**
(functions that make sense only in a particular realization)

M. Roggenbach: Algebraic Specification, October 2002 118

Elements of Formal Specifications 18

Formal Specification – First Element

Formal Specifications describe an **Interface** –
written down as **Signature**, i.e. a list consisting of the

- Name and
- Profile

of all functions.

M. Roggenbach: Algebraic Specification, October 2002 118

Elements of Formal Specifications 19

Interfaces and Programming Languages

PL supporting Interfaces:
C++, Modula, ML, Haskell, Java, Eiffel, . . .

PL not supporting Interfaces:
Fortran, Pascal, C, Lisp, . . .

M. Roggenbach: Algebraic Specification, October 2002 118

Elements of Formal Specifications 20

Implementation by Lists – with Interface

Show Interface

M. Roggenbach: Algebraic Specification, October 2002 118

Elements of Formal Specifications 21

Specifying the Interface in CASL

```
spec DATABASE =  
  sorts Database; String; Nat  
  ops initial : Database;  
    look-up : Database  $\times$  String  $\rightarrow$  Nat;  
    update : Database  $\times$  String  $\times$  Nat  $\rightarrow$  Database  
end
```

M. Roggenbach: Algebraic Specification, October 2002 118

Elements of Formal Specifications 22

A Wrong Implementation

Start Database

M. Roggenbach: Algebraic Specification, October 2002 118

Formal Specifications – Second Element

A formal specification includes beside the

- (i) signature
- (ii) a description of the functions' properties.

Programming Languages fail for (ii):

Expressing properties of a function involves implementation details.

Interface and Properties in CASL

```
spec DATABASE =
  sort Database; String; Nat
  ops initial : Database;
    0 : Nat;
    look_up : Database × String → Nat;
    update : Database × String × Nat → Database
  forall s : Database; n : Nat; v, w : String
  • %[initial] look_up(initial, v) = 0
  • %[look_up.1] v = w ⇒
    look_up(update(s, v, n), w) = n
  • %[look_up.2] ¬ v = w ⇒
    look_up(update(s, v, n), w) = look_up(s, w)
end
```

Useless Database

Start Database

Formal Specifications = Abstract Datatypes

An Abstract Datatype consists of a

- (i) Signature,
- (ii) a description of the functions' properties,
- (iii) a description of the domains.

1.3 Bibliographic Remarks

Elements of Formal Specifications
27

A Domain Description in CASL

```

spec NAT =
  free types Nat ::= 0 | suc(Nat)
end
          
```

M. Roggenbach: Algebraic Specification, October 2002
118

1.4 Mathematical Preliminaries

Introduction
28

Bibliographic Remarks

- Formal Methods:
J.P. Bowen, M.G. Hinchey: *High-Integrity System Specification and Design*, Springer, 1999.
- Algebraic Specification:
Loeckx, Ehrich, Wolf: *Specification of Abstract Data Types*, Wiley & Teubner, 1996.
Astesiano et. al (eds): *Algebraic Foundations of Systems Specifications*, Springer, 1999.
- CASL und CoFI:
<http://www.cofi.info>

M. Roggenbach: Algebraic Specification, October 2002
118

Introduction
29

Mathematical Preliminaries

M. Roggenbach: Algebraic Specification, October 2002
118

Definition 1 (Functions)

A, B : sets

1. *function*: relation $f \subseteq A \times B$ with:

for every $a \in A$ exists at most one $b \in B$ such that $(a, b) \in f$.

notations:

- $f(a) = b$ instead of $(a, b) \in f$
- $f(a)$ undefined iff $\forall b \in B : \neg f(a) = b$.
- $f(a)$: value of f for the argument a .

2. total function: $\forall a \in A : f(a)$ defined.

3. partial function: $\exists a \in A : f(a)$ undefined.

notations:

- total function $f : A \rightarrow B$
- partial function $f : A \rightarrow ?B$

2 Many-Sorted-Algebras

2.1 Signatures

Many-Sorted-Algebras
31

Signatures

Basic Questions:

- How to describe interfaces?
 \leadsto Signatures
- Which 'programs' fit to a signature?
 \leadsto Σ -Algebras

M. Roggenbach: Algebraic Specification, October 2002
118

Signatures
32

Examples of Signatures

Example 1 (Database)

- $S = \{ \text{Database}, \text{String}, \text{Nat} \}$
- $\Omega = \{$
 - $\text{initial}: \rightarrow \text{Database},$
 - $\text{look_up}: \text{Database} \times \text{String} \rightarrow \text{Nat},$
 - $\text{update}: \text{Database} \times \text{String} \times \text{Nat} \rightarrow \text{Database} \}$

M. Roggenbach: Algebraic Specification, October 2002
118

Definition 2 (Signature)

A signature Σ is a pair $\Sigma = (S, \Omega)$ of sets,

S : set of sorts

Ω : set of operations

Each operation $\omega \in \Omega$ consists of a tuple

$$\omega = n : s_1 \times \dots \times s_k \rightarrow s ; k \geq 0, s_1, \dots, s_k, s \in S.$$

n : operation name

$s_1 \times \dots \times s_k \rightarrow s$: profile

s_1, \dots, s_k : argument sorts

s : target sort

Signatures
33

Example 2 (Integer)

- $S = \{ Int \}$
- $\Omega = \{$
 - $- : Int \rightarrow Int,$
 - $+ : Int \times Int \rightarrow Int,$
 - $- : Int \times Int \rightarrow Int \}$

M. Roggenbach: Algebraic Specification, October 2002
118

Signatures
34

Example 3 (Lists of Natural Numbers)

- $S = \{ List, Nat \}$
- $\Omega = \{$
 - $nil : \rightarrow List,$
 - $:: : Nat \times List \rightarrow List,$
 - $++ : List \times List \rightarrow List,$
 - $! : List \times Nat \rightarrow Nat \}$

M. Roggenbach: Algebraic Specification, October 2002
118

Signatures
35

Example 4 (Editor)

- $S = \{ Text, Character, Position \}$
- $\Omega = \{$
 - $insert: Character \times Position \times Text \rightarrow Text,$
 - $delete: Position \times Text \rightarrow Text,$
 - $move: Position \times Position \times Position \times Text \rightarrow Text \}$

M. Roggenbach: Algebraic Specification, October 2002
118

Signatures
36

Example 5 (Java-Compiler)

- $S = ?$
- $\Omega = ?$

M. Roggenbach: Algebraic Specification, October 2002
7/8

Example 6 (Signatures)

Java-Compiler example (slide 37).

here a possible solution:

$$\begin{aligned}
 S &= \{ \text{Java-source, Class-file, Errors, Compiler-options} \} \\
 \Omega &= \{ \text{javac} : \text{Java-source} \times \text{Compiler-options} \rightarrow \text{Errors} \\
 &\quad \text{javac} : \text{Java-source} \times \text{Compiler-options} \rightarrow \text{Class-file} \}
 \end{aligned}$$

Remark 1 (Signatures)

$\omega = \omega' \Leftrightarrow \omega$ and ω' got the same name and the same profile.

Remark 2 (Signatures)

$k = 0 : n \rightarrow s$ is a constant of sort s .

Remark 3 (Signatures)

S and Ω are arbitrary sets.

$\rightarrow \Sigma = (\emptyset, \emptyset)$ is a signature.

$\rightarrow S, \Omega$ can be infinitive, e.g.

(i) sorts representing functions of all arities

$$\begin{aligned}
 S_0 &\quad \text{constants} \\
 S_1 &\quad : S_0 \rightarrow S_0 \\
 S_2 &\quad : S_0 \times S_0 \rightarrow S_0 \\
 &\vdots
 \end{aligned}$$

(ii) Fourier series:

$$\begin{aligned}
 &\cos(kx), \sin(kx), \quad k \in \mathbb{N}, k \geq 1 \\
 &\text{as elementary functions.}
 \end{aligned}$$

Signatures 37

CASL-Syntax

```

NAMED-SPEC ::= spec SPEC-NAME = BASIC-SPEC
              end/
              | spec SPEC-NAME = SPEC-NAME
                then ... then SPEC-NAME
                then BASIC-SPEC end/

BASIC-SPEC ::= BASIC-ITEMS ... BASIC-ITEMS

BASIC-ITEMS ::= SIG-ITEMS | ...

```

M. Roggenbach: Algebraic Specification, October 2002 37

Signatures 38

```

SIG-ITEMS ::= sort/sorts
            SORT-ITEM ; ... ; SORT-ITEM ;/
            | op/ops
            OP-ITEM ; ... ; OP-ITEM ;/
            | ...

SORT-ITEM ::= SORT , ... , SORT

```

M. Roggenbach: Algebraic Specification, October 2002 38

Signatures 39

```

OP-ITEM ::= OP-NAME , ... , OP-NAME :
           OP-TYPE

OP-TYPE ::= SOME-SORTS -> SORT | SORT

SOME-SORTS ::= SORT * ... * SORT

```

M. Roggenbach: Algebraic Specification, October 2002 39

Remark 4 (Signatures)

Operations

$$\omega = n : s_1 \times \dots \times s_k \rightarrow t_1 \times \dots \times t_l$$

can be simulated by

$$\begin{aligned}
\omega_1 &= n_1 : s_1 \times \dots \times s_k \rightarrow t_1 \\
&\vdots \\
\omega_l &= n_l : s_1 \times \dots \times s_k \rightarrow t_l
\end{aligned}$$

So it can also be written as

$$\omega = (\omega_1, \dots, \omega_l).$$

Signatures 40

Concepts vs Constructs

- **Concept:**
mathematical definition
- **Construct:**
phrase in CASL –
has the concept as its 'semantics'

M. Roggenbach: Algebraic Specification, October 2002 218

Signatures 41

Example 7 (Database)

- $S = \{ \text{Database}, \text{String}, \text{Nat} \}$
- $\Omega = \{$
 $\text{initial}: \rightarrow \text{Database},$
 $\text{look_up}: \text{Database} \times \text{String} \rightarrow \text{Nat},$
 $\text{update}: \text{Database} \times \text{String} \times \text{Nat} \rightarrow \text{Database} \}$

spec DATABASE =
sorts Database; String; Nat
ops $\text{initial} : \text{Database};$
 $\text{look_up} : \text{Database} \times \text{String} \rightarrow \text{Nat};$
 $\text{update} : \text{Database} \times \text{String} \times \text{Nat} \rightarrow \text{Database}$
end

M. Roggenbach: Algebraic Specification, October 2002 218

Signatures 42

Parsing CASL

Show Database

Parse Database

M. Roggenbach: Algebraic Specification, October 2002 218

Signatures 43

Example 8 (Integer)

- $S = \{ \text{Int} \}$
- $\Omega = \{$
 $- : \text{Int} \rightarrow \text{Int},$
 $+ : \text{Int} \times \text{Int} \rightarrow \text{Int},$
 $\times : \text{Int} \times \text{Int} \rightarrow \text{Int} \}$

spec INTEGER =
sorts Int
ops $- : \text{Int} \rightarrow \text{Int};$
 $+ : \text{Int} \times \text{Int} \rightarrow \text{Int};$
 $\times : \text{Int} \times \text{Int} \rightarrow \text{Int}$
end

M. Roggenbach: Algebraic Specification, October 2002 218

Signatures 44

Example 9 (Nat)

```

spec NAT =
  sorts Nat
  ops 0 : Nat;
       suc : Nat → Nat
end

```

M. Roggenbach: Algebraic Specification, October 2002 718

Remark 5 (CASL-Signatures)

S and Ω are finite in CASL-signatures.

2.2 Algebras

Many-Sorted-Algebras 45

Algebras

Basic Questions:

- How are 'programs' related?
 \leadsto Σ -Homomorphisms
- Which 'programs' are considered "identical"?
 \leadsto isomorphic Σ -Algebras

M. Roggenbach: Algebraic Specification, October 2002 718

Definition 3 (Σ -Algebra)

$\Sigma = (S, \Omega)$ signature. A Σ -Algebra assigns

\rightarrow a set $A(s)$ to each sort $s \in S$
 ('carrier set').

\rightarrow a total function
 $A(n : s_1 \times \dots \times s_k \rightarrow s) : A(s_1) \times \dots \times A(s_k) \rightarrow A(s)$
 to each operation $(n : s_1 \times \dots \times s_k \rightarrow s) \in \Omega$, $k \geq 0$.

Note: $k = 0 \Rightarrow A(n : \rightarrow s) \in A(s)$

$\text{Alg}(\Sigma)$: class of all Σ -algebras.

Remark 6 (Class)

The mathematical concept of classes is subject of the tutorial.

Algebras 46

Example 10 (Σ -Algebra: Nat)

```

spec NAT =
  sorts Nat
  ops 0 : Nat;
      suc : Nat → Nat
end

```

M. Roggenbach: Algebraic Specification, October 2002 718

$$\begin{aligned}
 A_1(\text{Nat}) &= N \\
 A_1(0) &= 0 \\
 A_1(\text{suc})(n) &= n + 1 \quad \forall n \in N
 \end{aligned}$$

$$\begin{aligned}
 A_2(\text{Nat}) &= Z \\
 A_2(0) &= 0 \\
 A_2(\text{suc})(z) &= -z - 1 ; z > 0 \\
 &\quad -z + 1 ; z \leq 0
 \end{aligned}$$

$$\begin{aligned}
 A_3(\text{Nat}) &= \{42\} \\
 A_3(0) &= 42 \\
 A_3(\text{suc})(42) &= 42
 \end{aligned}$$

Algebras 47

Example 11 (Σ -Algebra: Database)

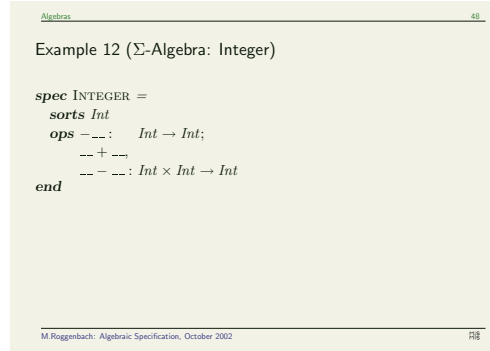
```

spec DATABASE =
  sorts Database; String; Nat
  ops initial : Database;
      look_up : Database × String → Nat;
      update : Database × String × Nat → Database
end

```

M. Roggenbach: Algebraic Specification, October 2002 718

Semantics of Java-programms are Σ -algebras for the signature.



$$A_1(Int) = Z$$

$$A_1(-)(z) = -z ; z \in Z$$

$$A_1(+)(z_1, z_2) = z_1 + z_2 \quad z_1, z_2 \in Z$$

$$A_1(-)(z_1, z_2) = z_1 - z_2 \quad z_1, z_2 \in Z$$

$$A_2(Int) = \{z \in Z \mid -2^n + 1 \leq z \leq 2^n - 1\} =: int ; n \in N$$

$$A_2(-)(z) = -z ; z \in int$$

$$A_2(+)(z_1, z_2) = (z_1 + z_2) \text{ rem } 2^n$$

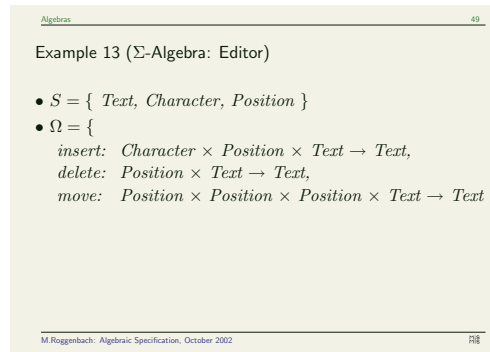
$$A_2(-)(z_1, z_2) = (z_1 - z_2) \text{ rem } 2^n$$

Note:

$$\rightarrow |x \text{ quot } y| = |x| \text{ quot } |y|$$

$$\rightarrow x = (x \text{ quot } y) * y + (x \text{ rem } y)$$

$$\rightarrow |x \text{ rem } y| = |x| \text{ rem } |y|$$



Take your favorite editor (emacs, vi, ...).

Remark 7 (Algebras in CASL)

CASL semantics:

carrier sets are non-empty.

$$[[\text{CASL-Spec with Sort- and Op-Items}]] = \text{Alg}'(\Sigma)$$

$$\Sigma = (S, \Omega)$$

S : set of all declared sorts

Ω : set of all declared operations

$$\text{Alg}'(\Sigma) = \{A \in \text{Alg}(\Sigma) \mid \forall s \in S. A(s) \neq \emptyset\}$$

Definition 4 (Σ -Homomorphism)

Let $\Sigma = (S, \Omega)$ be a signature and

let A, B be Σ -algebras.

A Σ -homomorphism is a family

$$h = (h_s)_{s \in S}$$

with

$$h_s : A(s) \rightarrow B(s)$$

and

$$\begin{array}{ccccc} A(s_1) \times \dots \times A(s_k) & \xrightarrow{A(\omega)} & A(s) \\ | & & | \\ h_{s_1} & & h_{s_k} \quad // \quad h_s \\ \downarrow & & \downarrow \\ B(s_1) \times \dots \times B(s_k) & \xrightarrow{B(\omega)} & B(s) \end{array}$$

for all $\omega \in \Omega$.

note: $k = 0 \Rightarrow h_s(A(\omega)) = B(\omega)$

Definition 5 (Isomorphism)

(i) A bijective homomorphism is called isomorphism.

(ii) Let Σ be a signature.

Σ -algebras A, B are called isomorphic

if there exists a Σ -isomorphism from A to B .

(notation: $A \simeq B$)

Example 14 (Homomorphism)

$$\begin{array}{lll} A(\text{Nat}) & = & N \\ A(0) & = & 0 \\ A(\text{suc})(n) & = & n + 1 \end{array} \qquad \begin{array}{lll} B(\text{Nat}) & = & N \\ B(0) & = & 1 \\ B(\text{suc})(n) & = & n + 1 \end{array}$$

(i) $g : A \rightarrow B$
 $n \mapsto n + 1$ is homomorphism.

$$\rightarrow g(A(0)) = g(0) = 1 = B(0)$$

$$\rightarrow g(A(suc)(n)) = g(n + 1) = n + 2$$

$$B(suc)(g(n)) = B(suc)(n + 1) = n + 2$$

(ii) is there a homomorphism $h : B \rightarrow A$?

$$\begin{aligned} h(B(suc)(0)) &= h(1) = 0 \\ A(suc)(h(0)) &= h(0) + 1 \quad \quad \quad ! \\ &\quad \quad \quad \underline{\quad} \end{aligned}$$

$$h(B(0)) = h(1) = A(0) = 0$$

but $n + 1 = 0$ has no solution in N

$$\begin{aligned} (iii) \quad C(Nat) &= \{42\} \\ C(0) &= 42 \\ C(suc)(42) &= 42 \end{aligned}$$

$$h : A \rightarrow C$$

$$h(n) = 42 \text{ is a homomorphism.}$$

$$\rightarrow h(A(0)) = h(42) = C(0)$$

$$\rightarrow h(A(suc)(n)) = h(n + 1) = 42$$

$$C(suc)(h(n)) = C(suc)(42) = 42$$

Theorem 1 (Homomorphism)

The composition of Σ -homs yields a Σ -hom.

proof: exercise

Theorem 2 (Homomorphism)

$$\Sigma = (S, \Omega) \quad \text{signature}$$

$$h : A \rightarrow B \quad \text{a } \Sigma\text{-isomorphism.}$$

$$\Rightarrow h^{-1} = (h_s^{-1})_{s \in S} \text{ is a } \Sigma\text{-isomorphism.}$$

proof: let $s \in S$.

a) h_s bijective $\Rightarrow h_s^{-1}$ bijective

b) hom.-condition :

let $\omega \in \Omega$, i.e. $\omega = (n : s_1 \times \dots \times s_k \rightarrow s)$

$$h_s^{-1}(B(\omega)(b_1, \dots, b_k)) \stackrel{!}{=} A(\omega)(h_s^{-1}(b_1), \dots, h_s^{-1}(b_k))$$

h is hom.

$$\begin{aligned} \Rightarrow h_s(A(\omega)(h_{s_1}^{-1}(b_1), \dots, h_{s_k}^{-1}(b_k))) \\ = B(\omega)(h_{s_1} \circ h_{s_1}^{-1}(b_1), \dots, h_{s_k} \circ h_{s_k}^{-1}(b_k)) \\ = B(\omega)(b_1, \dots, b_k) \end{aligned}$$

$$\begin{aligned} h_s^{-1}(B(\omega)(b_1, \dots, b_k)) \\ = \underbrace{h_s^{-1}(h_s(A(\omega)(h_{s_1}^{-1}(b_1), \dots, h_{s_k}^{-1}(b_k))))}_{= A(\omega)((h_{s_1}^{-1}(b_1), \dots, h_{s_k}^{-1}(b_k)))} \\ = A(\omega)((h_{s_1}^{-1}(b_1), \dots, h_{s_k}^{-1}(b_k))) \end{aligned}$$

Remark 8 (Homomorphism)

Relation of isom. is an equivalence relation.

(r) identity is an isomorphism.

(s) theorem 2

(t) theorem 1

Algebras
50

Definition 6 (Abstract Datatype)
A *datatype* is a combined entity consisting of domains and operations on these domains.
A datatype is called

- *abstract* when viewed primarily in terms of the properties of the operations and domains, and
- *concrete*, when the emphasis is put on the representation in a programming language.

M. Roggenbach: Algebraic Specification, October 2002
118

Definition 7 (Abstract Datatype)

An abstract datatype for a signature Σ is a class C of Σ -algebras closed under isomorphism, i.e.:

$$A \in C \wedge B \simeq A \Rightarrow B \in C$$

Example 15 (Abstract Datatype)

$$\Sigma = (\{Nat\}, \{0 : Nat, suc : Nat \rightarrow Nat\})$$

(i) $Alg(\Sigma)$ is ADT.

(ii) $\{A \in Alg(\Sigma) \mid |A(Nat)| = 1\}$ is ADT.

(iii) $\{D \in Alg(\Sigma) \mid D \simeq A \vee D \simeq B\}$ where A and B are the algebras of Example 2.2.

2.3 Terms

Many-Sorted Algebras
51

Terms

Basic Question:

Given:

- interface (= signature) and
- program (= Σ -algebra)

How to denote values of a carrier set?

\leadsto **Terms**

M. Roggenbach: Algebraic Specification, October 2002
118

Remark 9 (Variables)

V : “Universe” of variables
 $X \subseteq V$: the variables one is “working with”.

Terms
52

Variables

With each signature $\Sigma = (\Omega, S)$ is associated a family

$$V = (V_s)_{s \in S}$$

of disjoint infinite sets.

Notations:

$v \in V_s$: variable of sort s .
 $X \subseteq V$: set of variables for Σ (by abuse of language).

Assumption: Variables of V and operation names of Ω are different.

M. Roggenbach: Algebraic Specification, October 2002
118

53

Terms

Let $\Sigma = (S, \Omega)$ be a signature.
 Let $X = (X_s)_{s \in S}$ be a family of variables for Σ .
 The *terms over Σ* ,

$$T_{\Sigma(X)} = (T_{\Sigma(X),s})_{s \in S},$$

are the smallest family of sets such that

- (i) $X_s \subseteq T_{\Sigma(X),s}$,
- (ii) $n \in T_{\Sigma(X),s}$, if $n : \rightarrow s \in \Omega$.
- (iii) $n(t_1, \dots, t_k) \in T_{\Sigma(X),s}$, if
 $n : s_1 \times \dots \times s_k \rightarrow s \in \Omega$, $k \geq 1$, $t_i \in T_{\Sigma(X),s_i}$, $1 \leq i \leq k$.

M. Roggenbach: Algebraic Specification, October 2002
718

Example 16 (Terms in Nat)

$$\begin{aligned} Nat &= (\{Nat\}, \{0 : \rightarrow Nat, \\ &\quad suc : Nat \rightarrow Nat\}) \end{aligned}$$

$$\begin{aligned} X &= (X_s)_{s \in \{Nat\}} = (X_{Nat}) \\ X_{Nat} &= \{x\} \end{aligned}$$

$$\begin{aligned} (i) \quad & \{x\} \subseteq T_{\Sigma(X),Nat} \\ (ii) \quad & 0 \in T_{\Sigma(X),Nat} \\ (iii) \quad & suc(t) \in T_{\Sigma(X),Nat} \quad \text{if} \quad t \in T_{\Sigma(X),Nat} \end{aligned}$$

$$\Rightarrow T_{\Sigma(X),Nat} = \{x\} \cup \{0\} \cup \{suc^n(t) \mid n \geq 1, t = 0 \vee t = x\}$$

$$T_{\Sigma,Nat} = \{0\} \cup \{suc^n(0) \mid n \geq 1\}$$

54

Notations

$Var(t)$: set of all variables occurring in term t .
 t is called *ground term* if $Var(t) = \emptyset$.
 $T_{\Sigma,s}$: set of all ground terms of sort s .
 $T_{\Sigma} = (T_{\Sigma,s})_{s \in S}$: family of all ground terms.

M. Roggenbach: Algebraic Specification, October 2002
718

Terms
55

Two Variants of Lists

Example 17 (NatList1)

```

spec NATLIST1 =
  sorts Nat; List
  ops nil :      List;
      _ :: _ :   Nat × List → List;
      _ ++ _ :  List × List → List;
      _ ! _ :   List × Nat → Nat
  end

```

M. Roggenbach: Algebraic Specification, October 2002
118

ground terms

$$T_{\Sigma} = (T_{\Sigma, Nat}, T_{\Sigma, List})$$

$$T_{\Sigma, Nat} = \emptyset$$

$$T_{\Sigma, List} = L(G)$$

$$\underline{\mathcal{G}} : z ::= nil \mid (z ++ z)$$

Terms
56

Example 18 (NatList2)

```

spec NATLIST2 = NAT then
  sorts List
  ops nil :      List;
      _ :: _ :   Nat × List → List;
      _ ++ _ :  List × List → List;
      _ ! _ :   List × Nat → Nat
  end

```

M. Roggenbach: Algebraic Specification, October 2002
118

NatList2 has ground terms of sort *Nat*:

$$T_{\Sigma, Nat} = \{0\} \cup \{suc^n(0) \mid n \geq 1\}$$

$$T_{\Sigma, List} = \text{really complicated}$$

some examples:

$$\rightarrow nil$$

$$\rightarrow suc^7(0) :: suc(0) :: nil ++ nil ++ suc^{42}(0)$$

$$\rightarrow suc^{111}(0) ! nil$$

Remark 10 (Semantic of terms)

Semantic of terms:

$$\begin{array}{ll}
t & \in T_{\Sigma(X),s} & \text{Signature } \Sigma = (S, \Omega) \\
\downarrow & & \\
a & \in A(s) & \Sigma\text{-algebra } A.
\end{array}$$

Terms
57

Assignment

$\Sigma = (S, \Omega)$: signature
 $X = (X_s)$: family of variables
 A : Σ -algebra.
An *assignment* of X for A is a family

$$\alpha = (\alpha_s)_{s \in S}$$

of total functions $\alpha_s : X_s \rightarrow A(s)$.

Remark:

- no assignment from X to A if $X_s \neq \emptyset$ and $A(s) = \emptyset$.
- in CASL holds: $A(s) \neq \emptyset$.

M. Roggenbach: Algebraic Specification, October 2002
118

Terms
58

Semantics of Terms

The *value* $A(\alpha)(t)$ of a term $t \in T_{\Sigma(X),s}$ for an assignment α is defined by induction on the term structure:

- (i) $A(\alpha)(t) = \alpha_s(x)$, if $t = x$, $x \in X_s$.
- (ii) $A(\alpha)(t) = A(\omega)$, if $t = n$, $\omega = (n : \rightarrow s) \in \Omega$.
- (iii) $A(\alpha)(t) = A(\omega) (A(\alpha)(t_1), \dots, A(\alpha)(t_k))$, if
 $\omega = (n : s_1 \times \dots \times s_k \rightarrow s) \in \Omega$, $k \geq 1$,
 $t_i \in T_{\Sigma(X),s_i}$, $1 \leq i \leq k$.

M. Roggenbach: Algebraic Specification, October 2002
118

Example 19 (Semantic of terms)

$$\begin{aligned}
Nat &= (\{Nat\}, \{0 : \rightarrow Nat, \\
&\quad suc : Nat \rightarrow Nat\})
\end{aligned}$$

$$X_{Nat} = \{x\} \quad \alpha = (\alpha_{Nat})$$

$$A(Nat) = N \quad \alpha_{Nat}(x) = 42$$

$$A(0) = 0$$

$$A(suc)(n) = n + 1$$

$$(i) \quad A(\alpha)(x) = \alpha(x) = 42$$

$$(ii) \quad A(\alpha)(0) = A(0) = 0$$

$$\begin{aligned}
(iii) \quad A(\alpha)(suc(x)) &= A(suc)(A(\alpha)(x)) \\
&= A(suc)(42) \\
&= 42 + 1 = 43
\end{aligned}$$

Theorem 3 (Semantic of terms)

let $\alpha, \beta : X \rightarrow A$ be assignments
 $t \in T_{\Sigma(X)}$ be a term.

if $\alpha(x) = \beta(x)$ for all $x \in \text{Var}(t)$
 then $A(\alpha)(t) = A(\beta)(t)$

proof: exercise.

Corollary 1 (Semantic of terms)

The value of a ground term does not depend on the assignment.

Theorem 4 (Semantic of terms)

$\Sigma = (S, \Omega)$ signature.

A, B Σ -algebras.

$h : A \rightarrow B$ Σ -hom.

$$\Rightarrow h(A(\alpha)(t)) = B(h \circ \alpha)(t)$$

for each term $t \in T_{\Sigma(X)}$ and assignment $\alpha : X \rightarrow A$

proof: exercise!