Algebraic Specification –
A Formalism to Specify
Sequential Programs

Markus Roggenbach

October 2002

Contents

1 Introduction

1.1 Formal Methods in Software Design

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

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 resitance to the mathematical definition of semantics."

(quoted freely from [?])

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

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

semi formal:
in a language with

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

Specifications

Specification: "description by properties"

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

Specifications should be
- complete
- precise
- consistent (no contradictions)

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)

Limitations of Formal Methods

"The world is not a formal system."

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

II. Errors within Formalisms.

III. Behaviour of a Program depends on
- Compiler
- Operating System
- Computer Hardware
- Embedding in a Technical Process
- Human Operator

Waterfall Model

	Requirement Elicitation and Analysis	
	↑↓	
Nat. Lang.	Non Formal Specification	
	↑↓	Validation
Spec. Lang.	Formal Specification	
	↑↓	"Inv. & Verify" or
Progr. Lang.	Implementation	Transformation
	↑↓	
	Test	
	↑↓	
	Maintenance	

1.2 Elements of Formal Specifications

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

Elements of Formal Specifications

Non Formal Specification
↑↓
Formal Specification
↑↓
Implementation

extract the "essentials" of
- a non formal specification
- all desirable implementations

Running Example: Database

Non Formal Specification:

Write a Java program that implements a database with
- "Name" and
- "Telephone Number"

as entries.

Implementation by Lists

Start Database

Show Source

Implementation by Sorted Binary Trees

Show Source

Compile Database

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)

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.

Interfaces and Programming Languages

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

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

Implementation by Lists – with Interface

Show Interface

Specifying the Interface in CASL

spec DATABASE =
  sorts $Database; String; Nat$
  ops $initial \quad : Database;$
    $look\_up \quad : Database \times String \to Nat;$
    $update \quad : Database \times String \times Nat \to Database$
end

A Wrong Implementation

Start Database

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 $\textsc{Database} =$
sort $Database; String; Nat$
ops $initial : Database;$
$0 : Nat;$
$look\_up : Database \times String \rightarrow Nat;$
$update : Database \times String \times Nat \rightarrow Database$

forall $s : Database; n : Nat; v, w : String$
- %[initial] $look\_up(initial, v) = 0$
- %[look_up_1] $v = w \Rightarrow look\_up(update(s, v, n), w) = n$
- %[look_up_2] $\neg\, v = w \Rightarrow look\_up(update(s, v, n), w) = look\_up(s, w)$

end

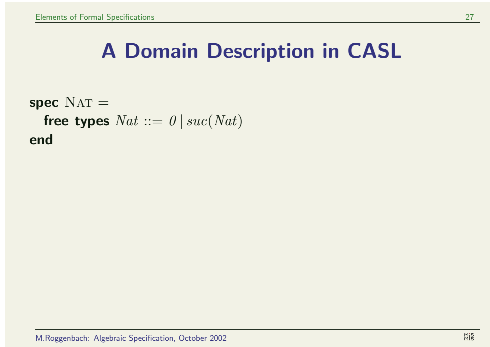
Useless Database

Start Database

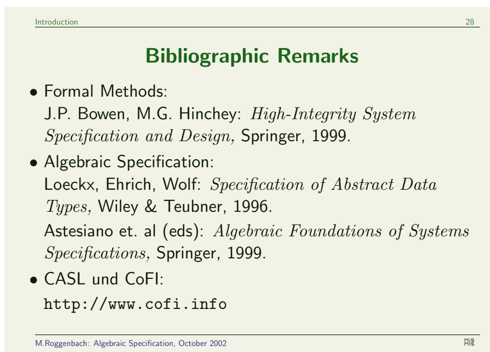
Formal Specifications = Abstract Datatypes

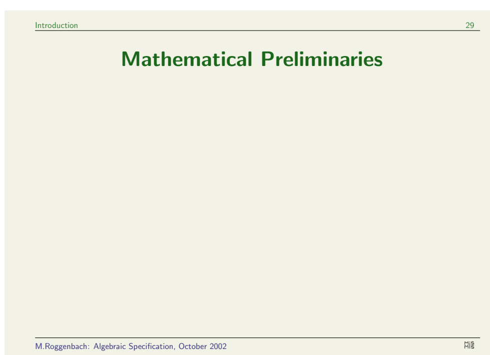
An Abstract Datatype consists of a
(i) Signature,
(ii) a description of the functions' properties,
(iii) a description of the domains.

1.3 Bibliographic Remarks



1.4 Mathematical Preliminaries





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.

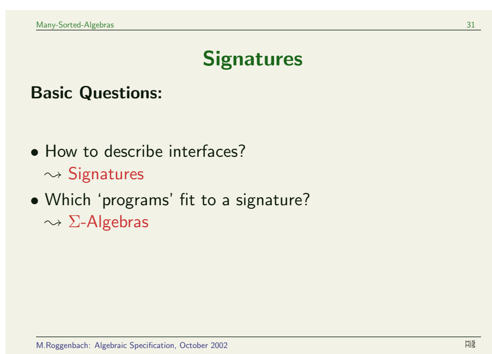
2. *total function:* $\forall a \in A : f(a)$ *defined.*
3. *partial function:* $\exists a \in A : f(a)$ *undefined.*

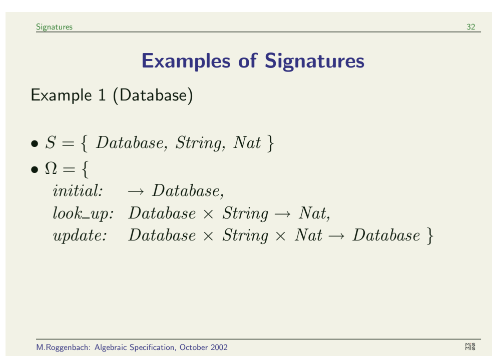
   notations:

   - *total function* $f : A \to B$
   - *partial function* $f A \to ?B$

2 Many-Sorted-Algebras

2.1 Signatures





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*
$\omega = n : s_1 \times \ldots \times s_k \to s\ ;\ k \geq 0, s_1, \ldots, s_k,\ s \in S.$

n	:	*operation name*
$s_1 \times \ldots \times s_k \to s$	:	*profile*
$s_1, \ldots, s_k$	:	*argument sorts*
s	:	*target sort*

Example 2 (Integer)

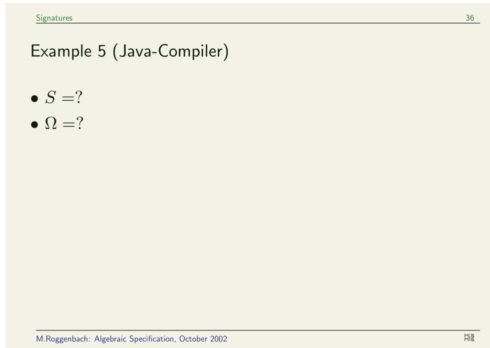
- $S = \{\ Int\ \}$
- $\Omega = \{\ \ -\ \ : Int \to Int,$
  $+\ \ : Int \times Int \to Int,$
  $-\ \ : Int \times Int \to Int\ \}$

Example 3 (Lists of Natural Numbers)

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

Example 4 (Editor)

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



Example 6 (Signatures)
Java-Compiler example (slide 37).
here a possible solution:

$$S \quad = \{\textit{Java-source, Class-file, Errors, Compiler-options}\}$$
$$\Omega \quad = \{\ \textit{javac} : \textit{Java-source} \times \textit{Compiler-options} \to \textit{Errors}$$
$$\textit{javac} : \textit{Java-source} \times \textit{Compiler-options} \to \textit{Class-file}\ \}$$

Remark 1 (Signatures)
$\omega = \omega' \Leftrightarrow \omega$ *and* ω' *got the same name and the same profile.*

Remark 2 (Signatures)
$k = 0 : n \to s$ *is a constant of sort* s.

Remark 3 (Signatures)
S *and* Ω *are arbitrary sets.*

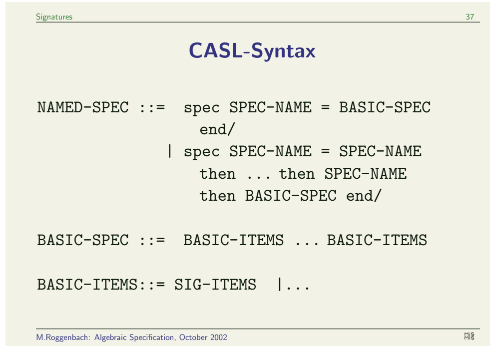
$\to \quad \Sigma = (\emptyset, \emptyset)$ *is a signature.*
$\to \quad S, \Omega$ *can be infinitive, e.g.*

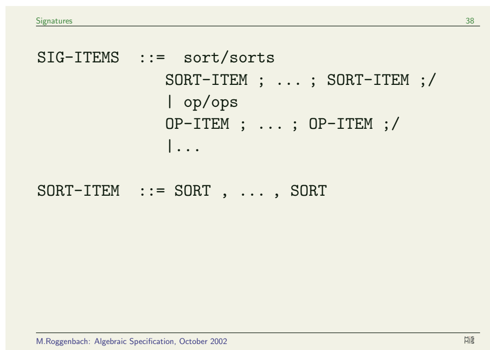
(i) sorts representing functions of all arities

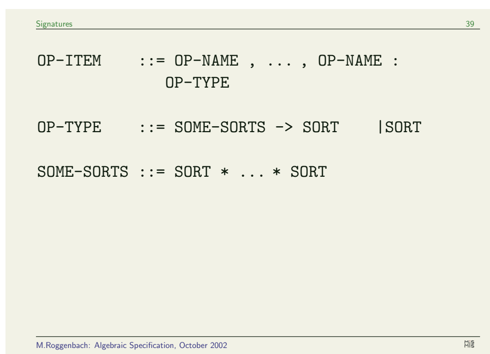
$$\begin{array}{lll} S_0 & & \textit{constants} \\ S_1 & : & S_0 \to S_0 \\ S_2 & : & S_0 \times S_0 \to S_0 \\ \vdots & & \end{array}$$

(ii) Fourier series:

$$\cos(kx), \sin(kx)\ ,\ k \in N, k \geq 1$$
as elementary functions.







Remark 4 (Signatures)
Operations

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

can be simulated by

$$\omega_1 = n_1 : s_1 \times \ldots \times s_k \rightarrow t_1$$
$$\vdots$$
$$\omega_l = n_l : s_1 \times \ldots \times s_k \rightarrow t_l$$

So it can also be written as

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

Concepts vs Constructs

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

Example 7 (Database)

- $S = \{\ \mathit{Database},\ \mathit{String},\ \mathit{Nat}\ \}$
- $\Omega = \{$
  $\mathit{initial}\colon\ \to \mathit{Database},$
  $\mathit{look\_up}\colon\ \mathit{Database} \times \mathit{String} \to \mathit{Nat},$
  $\mathit{update}\colon\ \mathit{Database} \times \mathit{String} \times \mathit{Nat} \to \mathit{Database}\ \}$

spec DATABASE =
  sorts *Database*; *String*; *Nat*
  ops $\mathit{initial} :\ \mathit{Database};$
    $\mathit{look\_up} : \mathit{Database} \times \mathit{String} \to \mathit{Nat};$
    $\mathit{update} :\ \mathit{Database} \times \mathit{String} \times \mathit{Nat} \to \mathit{Database}$
end

Parsing CASL

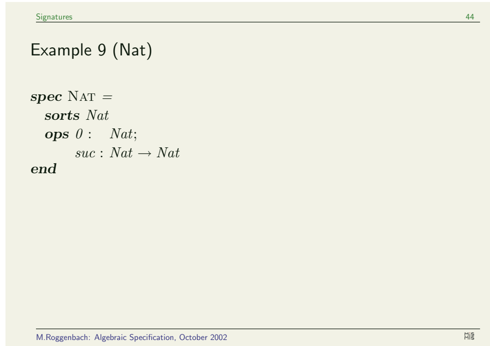
Show Database

Parse Database

Example 8 (Integer)

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

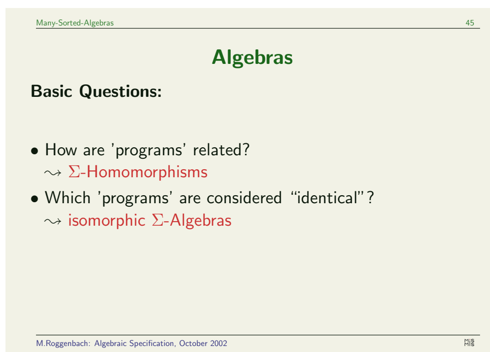
spec INTEGER =
  sorts *Int*
  ops $-\_\_ :\ \mathit{Int} \to \mathit{Int};$
    $\_\_ + \_\_,$
    $\_\_ - \_\_ : \mathit{Int} \times \mathit{Int} \to \mathit{Int}$
end



Remark 5 (CASL-Signatures)

S and Ω are finite in CASL-signatures.

2.2 Algebras



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').

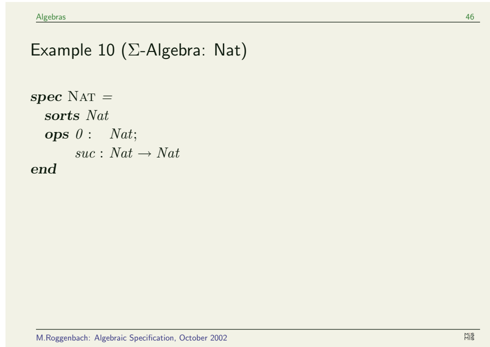
$\rightarrow$ *a total function*
$A(n : s_1 \times \ldots \times s_k \rightarrow s) : A(s_1) \times \ldots \times A(s_k) \rightarrow A(s)$
to each operation $(n : s_1 \times \ldots \times s_k \rightarrow s) \in \Omega$, $k \geq 0$.

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

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

Remark 6 (Class)

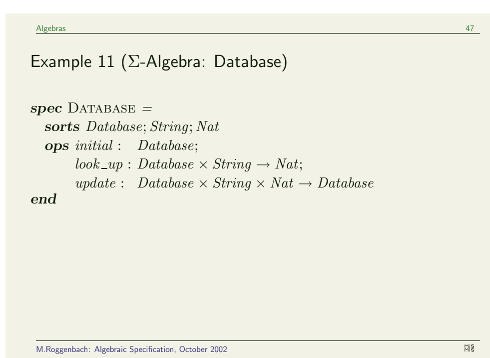
The mathematical concept of classes is subject of the tutorial.



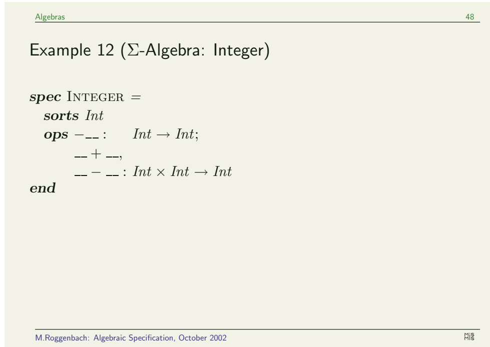
$$
\begin{aligned}
A_1(Nat) &= N \\
A_1(0) &= 0 \\
A_1(suc)(n) &= n+1 \;\; \forall n \in N
\end{aligned}
$$

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

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



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



$$
\begin{aligned}
A_1(Int) &= Z \\
A_1(-)(z) &= -z\ ;\ z \in Z \\
A_1(+)(z_1, z_2) &= z_1 + z_2 \ \ z_1, z_2 \in Z \\
A_1(-)(z_1, z_2) &= z_1 - z_2 \ \ z_1, z_2 \in Z
\end{aligned}
$$

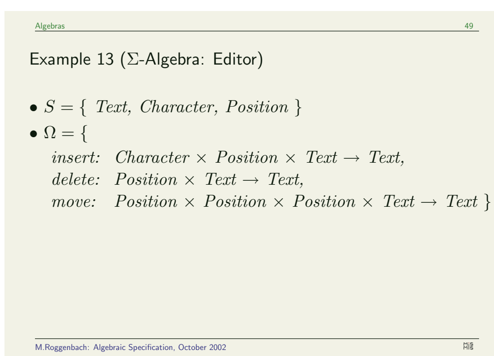
$$
\begin{aligned}
A_2(Int) &= \{z \in Z | -2^n + 1 \le z \le 2^n - 1\} =: int\ ;\ n \in N \\
A_2(-)(z) &= -z\ ;\ z \in int \\
A_2(+)(z_1, z_2) &= (z_1 + z_2)\ rem\ 2^n \\
A_2(-)(z_1, z_2) &= (z_1 - z_2)\ rem\ 2^n
\end{aligned}
$$

Note:

→ $|x\ quot\ y| = |x|\ quot\ |y|$

→ $x = (x\ quot\ y) * y + (x\ rem\ y)$

→ $|x\ rem\ y| = |x|\ rem\ |y|$



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

Remark 7 (Algebras in CASL)

CASL semantics:

carrier sets are non-empty.

[| *CASL-Spec with Sort- and Op-Items* |] $= Alg'(\Sigma)$

$\Sigma = (S, \Omega)$

S : *set of all declared sorts*

Ω : *set of all declared operations*

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

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

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

and

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

for all $\omega \in \Omega$.
note: $k = 0 \Rightarrow h_s(A(\omega)) = B(\omega)$

Definition 5 (Isomorphism)

(i) *A bijective homomorphism is called* *<u>isomorphism</u>.*

(ii) *Let* Σ *be a signature.*
Σ*-algebras* A*,* B *are called* *<u>isomorphic</u>*
if there exists a Σ*-isomorphism from* A *to* B*.*
(notation: $A \simeq B$*)*

Example 14 (Homomorphism)

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

(i) $g : \quad A \to B$

$\quad n \mapsto n+1 \qquad$ *is homomorphism.*

$\to \quad g(A(0)) = g(0) = 1 = B(0)$

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

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

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

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

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

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

(iii) $C(Nat) = \{42\}$

$C(0) = 42$

$C(suc)(42) = 42$

$h : A \to C$

$h(n) = 42$ *is a homomorphism.*

$\to \quad h(A(0)) = h(42) = C(0)$

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

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

Theorem 1 (Homomorphism)

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

proof: exercise

Theorem 2 (Homomorphism)

$\Sigma = (S, \Omega) \qquad$ *signature*

$h : A \to B \qquad$ *a* Σ*-isomorphism.*

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

proof: *let* $s \in S$.

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

b) *hom.-condition :*

let $\omega \in \Omega$, *i.e.* $\omega = (n : s_1 \times \ldots \times s_k \to s)$

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

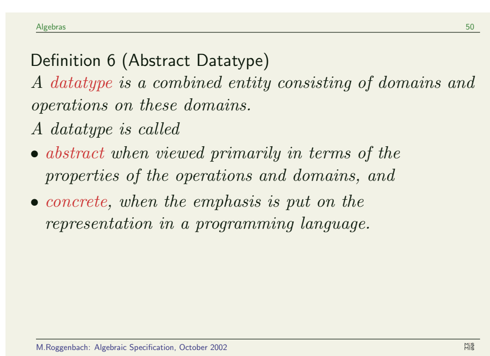
h is hom.

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

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

Remark 8 (Homomorphism)
Relation of isom. is an equivalence relation.

(r) *identity is an isomorphism.*
(s) *theorem 2*
(t) *theorem 1*



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)

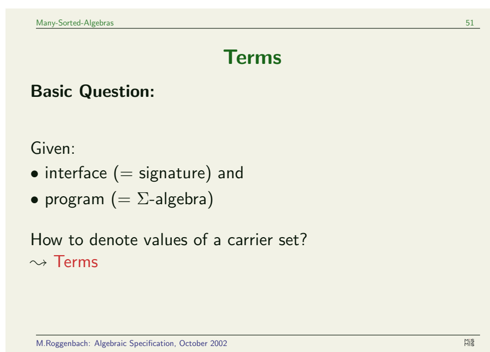
$$\Sigma = (\{Nat\}, \{0 : Nat, suc : Nat \rightarrow Nat\})$$

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

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

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

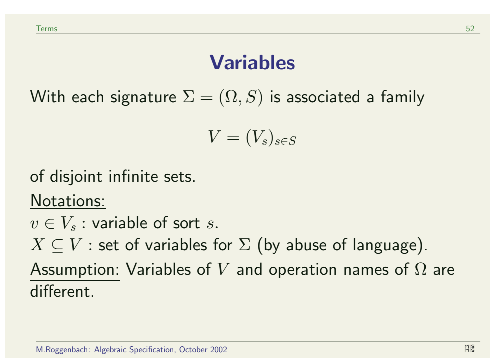
2.3 Terms



Remark 9 (Variables)

V : *"Universe" of variables*

$X \subseteq V$: *the variables one is "working with".*



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 $n :\to s \in \Omega$.
(iii) $n(t_1, \ldots, t_k) \in T_{\Sigma(X),s}$, if
$n : s_1 \times \ldots \times s_k \to s \in \Omega$, $k \geq 1$, $t_i \in T_{\Sigma(X),s_i}, 1 \leq i \leq k$.

Example 16 (Terms in Nat)

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

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

(i) $\{x\} \subseteq T_{\Sigma(X),Nat}$

(ii) $0 \in T_{\Sigma(X),Nat}$

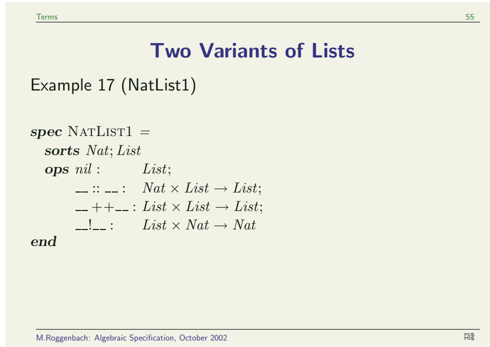
(iii) $suc(t) \in T_{\Sigma(X),Nat}$ *if* $t \in T_{\Sigma(X),Nat}$

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

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

Notations

$Var(t)$: set of all variables occuring 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.

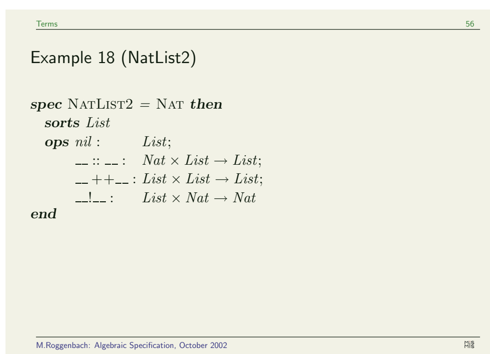


ground terms

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

$$\begin{aligned} T_{\Sigma,Nat} &= \emptyset \\ T_{\Sigma,List} &= L(G) \end{aligned}$$

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



NatList2 has ground terms of sort Nat:

$$\begin{aligned} T_{\Sigma,Nat} &= \{0\} \cup \{suc^n(0) \mid \; n \geq 1\} \\ T_{\Sigma,List} &= \text{really complicated} \end{aligned}$$

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:

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

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 $\alpha_s : X_s \to A(s)$.
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

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 $t = n$, $\omega = (n :\to s) \in \Omega$.
(iii) $A(\alpha)(t) = A(\omega)\,(A(\alpha)(t_1), \ldots, A(\alpha)(t_k))$, if $\omega = (n : s_1 \times \ldots \times s_k \to s) \in \Omega$, $k \geq 1$, $t_i \in T_{\Sigma(X),s_i}, 1 \leq i \leq k$.

M.Roggenbach: Algebraic Specification, October 2002

Example 19 (Semantic of terms)

$$
\begin{array}{lll}
Nat & = & (\{Nat\}, \; \{0 :\to Nat, \\
& & \qquad\qquad suc : Nat \to Nat\})
\end{array}
$$

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

$$
\begin{array}{lll}
A(Nat) & = & N \qquad\qquad \alpha_{Nat}(x) = 42 \\
A(0) & = & 0 \\
A(suc)(n) & = & n + 1
\end{array}
$$

$$
\begin{array}{lll}
\textit{(i)} & A(\alpha)(x) = \alpha(x) = 42 & \\
\textit{(ii)} & A(\alpha)(0) = A(0) = 0 & \\
\textit{(iii)} & A(\alpha)(suc(x)) & = A(suc)(A(\alpha)(x)) \\
& & = A(suc)(42) \\
& & = 42 + 1 = 43
\end{array}
$$

Theorem 3 (Semantic of terms)

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

if $\alpha(x) = \beta(x)$ *for all* $x \in Var(t)$
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.*

$h : A \to B$ Σ*-hom.*

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

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

proof: exercise!