Algebraic Specification of Abstract Data Types

Wolfgang Ahrendt

Graduate Course, June 2003

Department of Computing Science

Chalmers University, Göteborg

Organisation

Lectures:

Mon to Fri, 10.15-12.00, in MD10, *except: Tue, MD9*

Exercises:

given in the lecture
from these, select a number k of exercises, with $k \gg 0$
solved individually, or in groups, where you want, in the early afternoon

Discussing Solutions and Further Questions:

Mon to Fri, 16.15-18.00, in MD10
active participation

Part 1: Introduction

Data Types

data types appear everywhere in the context of computing

a data type

represents

a collection of data,
sharing a similar structure,
providing similar services

Data Types

- programs make use of data types
  - provided by the programming language
  - user definable in the programming language
- programs realise data types
  here, the description of the data type serves as
  - a specification of the program
  - an interface between program modules

Abstract Data Types

abstract data types

- *abstract away* from the implementation of the data type
- (sometimes) even *abstract away* from a single data type,
  referring to a whole class of *similar* data types
- are *described* (only) *abstractly*

Formal Specification of Abstract Data Types

constitute formal description of abstract data type,
using a formal language

formal specification styles:

- *state oriented*
  examples: Z, VDM, B, OCL
  not in this course
- *axiomatic*
  formulas ('axioms') relate different operations of the data type to each other
  similar as in description of *algebras* in math
  ⇒'*algebraic specification*'
  in this course

Algebraic Specification of Abstract Data Types

classical example: the data type *stack*

(not precise syntax or semantics yet)

$$
\begin{aligned}
Isempty(Emptystack) &= True \\
Isempty(Push(n,s)) &= False \\
Pop(Push(n,s)) &= s \\
Top(Push(n,s)) &= n \\
Del(n, Emptystack) &= Emptystack \\
Del(n, Push(n,s)) &= s \\
n \neq m \rightarrow Del(n, Push(m,s)) &= Push(m, Del(n,s))
\end{aligned}
$$

intuitively: $Emptystack$ and $Push$ are constructors, $Isempty$ is a query,
Pop and Top are selectors, Del is a modifier

Exercise

Exercise 1.1.
describe any *data type you like, by giving axioms for it's operations*

- *mainly use equations*
- *if needed, use inequalities or other logical connectives* ($\neg, \vee, \rightarrow, \ldots$)
- *if needed, use quantifiers* ($\exists, \forall$)
- *don't use higher order logic*

Also, state one or two properties (i.e. formulas again), which intuitively follow from your specification.

Exercise (cont'd)

It is a *feature* of exercise 1.1 that things remain unclear
(no precise syntax or semantics yet).

Ideas for data types you might specify:

- trees (binary or arbitrary branching), arrays, sets, acyclic graphs, ...
- or look into the libraries you use frequently
- ...

Historical Remarks

algebraic specification:

- rooted in *universal algebra*
- first algebraic specification method:
  equational specifications with *initial semantics*
  [Goguen, Thatcher, Wagner, Wright 75]
  the 'initial approach' dominated the field for many years
  several specification languages
  strong connections to term rewriting
- an alternative approach: loose semantics [Guttag 75]
  less restrictive, less 'computational'
  got more popular in the 90s
  aims at 'requirement specification'
  several specification languages
- current trend: unifying the approaches
  language: CASL

CoFI and CASL

CoFI

CoFI is the 'Common Framework Initiative' for the development general-purpose algebraic specification language.

CASL

Is the outcome, the 'Common Algebraic Specification Language'.
Subsumes many existing languages.

Differences to (the Mainstream of) Logic

More restrictive semantics, which serve:

- our intuition of data types (certainly debatable)
- the needs of computer science applications
- expressiveness, while keeping the logic syntactically simple
- partly: 'uniqueness' og the 'meaning'
- partly: achieving (almost) executable specifications

The Course of The Course

1. Preliminaries
2. Many-Sorted Algebras
3. Particular Classes of Algebras
   (e.g. term algebras, generated algebras, initial algebras)
4. Equational Logics
5. Loose Specification
6. Initial Specification
7. CASL by example

Literature

J. Loeckx, H.-D. Ehrich, M. Wolf
Specification of Abstract Data Types
Wiley/Teubner, 1996

J. Loeckx, H.-D. Ehrich, M. Wolf
Algebraic specification of abstract data types
in: S. Abramski, D.M. Gabbay, T.S.E. Maibaum (Eds.)
Handbook of Logic in Computer Science
Oxford University Press 2000

M. Wirsing
Algebraic Specification
in: J. v. Leeuwen (Ed.)
Handbook of Theoretical Computer Science, Volume B
Elsevier and MIT Press 1990

Part 2: Preliminaries

Remark

what follows now:

- mathematical notation
- *not yet* syntax of a specification language

Sets

usual set notations, including:

empty set: $\emptyset$

subset: $A \subseteq B$

proper subset: $A \subset B$ for $A \subseteq B$ and $A \neq B$

power set of A: $\wp(A) = \{B \mid B \subseteq A\}$

Cartesian product: $A_1 \times \ldots \times A_k = \{(a_1, \ldots, a_k) \mid a_i \in A_i, 1 \leq k\}$
special case: $A^k = A \times \ldots \times A$

family of sets: $(A_i)_{i \in I}$
contains exactly (and exclusively) one A_i for each $i \in I$

Particular Sets

$\mathbb{N}_0 = \{0, 1, 2, \ldots\}$	the set of *natural numbers*
$\mathbb{N} = \{1, 2, \ldots\}$	
$\mathbb{Z} = \{\ldots, -1, 0, 1, \ldots\}$	the set of *integers*
$\mathbb{Q} = \{ \frac{n}{m} \mid n, m \in \mathbb{Z}$ and $m \neq 0\}$	the set of *rational numbers*
$\{\text{true}, \text{false}\}$	the set of *truth values*

Relations

"R is a relation between A and B"

$$:\Longleftrightarrow$$

$$R \subseteq A \times B$$

"R is a relation on A"

$$:\Longleftrightarrow$$

$$R \subseteq A \times A$$

'$a\ R\ b$' denotes $(a, b) \in R$

Relations

R (on A) is:

reflexive $:\Longleftrightarrow$ $(a,a) \in R$ for all $a \in A$

irreflexive $:\Longleftrightarrow$ $(a,a) \in R$ for *no* $a \in A$

symmetric $:\Longleftrightarrow$ $(a_1,a_2) \in R$ implies $(a_2,a_1) \in R$

antisymmetric $:\Longleftrightarrow$ $(a_1,a_2) \in R$ and $(a_2,a_1) \in R$ implies $a_1 = a_2$

transitive $:\Longleftrightarrow$ $(a_1,a_2) \in R$ and $(a_2,a_3) \in R$ implies $(a_1,a_3) \in R$

a *partial order* $:\Longleftrightarrow$ R is reflexive, antisymmetric, and transitive (e.g. $\leq$ on $\mathbb{N}_0$)

an *irreflexive partial order* $:\Longleftrightarrow$ R is irreflexive and transitive (e.g. $<$ on $\mathbb{N}_0$)

an *equivalence relation* $:\Longleftrightarrow$ R is reflexive, symmetric, and transitive

More on Partial Orders

Let $\sqsubseteq$ be a partial order on A.

$a \in A$ is called *minimal* (w.r.t. $\sqsubseteq$), if there is no $b \in A$ with $b \neq a$ and $b \sqsubseteq a$.

$a \in A$ is called *least* (w.r.t. $\sqsubseteq$), if $a \sqsubseteq b$ for all $b \in A$.

a partial order $\sqsubseteq$ on A is *well-founded*
$:\Longleftrightarrow$
every non-empty subset of A contains at least one minimal element (w.r.t. $\sqsubseteq$)

More on Partial Orders

Notation ($\sqsubseteq$, $\sqsubset$)
Let $\sqsubseteq$ be a partial order on A.
Then $\sqsubset$ denotes the relation on A with: $a \sqsubset b$ iff $a \sqsubseteq b$ and $a \neq b$. $\star$

Theorem 2.1.
For each partial order $\sqsubseteq$, the corresponding relation $\sqsubset$ is an irreflexive partial oder. $\star$

a partial order $\sqsubseteq$ on A is *Noetherian*
$:\Longleftrightarrow$
there exists *no* infinite sequence $a_1, a_2, \ldots$ with $a_{i+1} \sqsubset a_i$, f.a. $i \in \mathbb{N}$.

Exercises

Exercise 2.1.
Prove theorem 2.1.

More on Partial Orders

Theorem 2.2.
Let $\sqsubseteq$ be a partial order on A.
Then the following two statements are equivalent:

- *$\sqsubseteq$ is Noetherian*
- *$\sqsubseteq$ is well-founded*

$\star$

Noetherian Induction

Theorem 2.3.
Let $\sqsubseteq$ be a Noetherian partial order on A.

In order to prove that a property P holds for all $a \in A$,
it is sufficient to show for all $a \in A$:

$$\boxed{\text{if } P \text{ holds for all } b \in A \text{ with } b \sqsubset a, \text{ then } P \text{ holds for } a}$$

$\star$

Exercises

Exercise 2.2.
Prove theorem 2.3, using theorem 2.2.

Equivalence Classes

Let $\equiv$ be an equivalence relation on A.

The equivalence class $[a]_\equiv$ is defined to be: $\{b \in A \mid a \equiv b\}$

Simplified Notation for Families of Relations

Let $A = (A_i)_{i \in I}$ and $B = (B_i)_{i \in I}$ be families of sets (same I):

$R \subseteq A \times B$ denotes family $R = (R_i)_{i \in I}$ of relations $R_i \subseteq A_i \times B_i$, $i \in I$.

A family R is said to have a property if each R_i, $i \in I$ has the property.

Example: "$(R_i)_{i \in I}$ is an equivalence relation".

Functions

Let A and B be sets:

A relation $f \subseteq A \times B$ is called *function from A to B* if:
For each $a \in A$ exists at most one $b \in B$ with $(a, b) \in f$ (functionality)

Notations

$f : A \longrightarrow B$ denotes a function f from A to B

for $f : A \longrightarrow B$, A is called *domain*, B is called *codomain*

$f(a) = b$ denotes $(a, b) \in f$

if $f(a) = b$, then $f(a)$ denotes b

$f(a)$ *is defined* denotes that there is a $b \in B$ with $f(a) = b$

f is called a *predicate* if the codomain is $\{$true, false$\}$

Properties of Functions

A function $f : A \longrightarrow B$ is:

- *injective* $:\Longleftrightarrow$ for every $b \in B$ there is *at most* one $a \in A$
- *surjective* $:\Longleftrightarrow$ for every $b \in B$ there is *at least* one $a \in A$
- *bijective* $:\Longleftrightarrow$ f is injective and surjective
- *total* $:\Longleftrightarrow$ $f(a)$ is defined for all $a \in A$

More on Functions

The *identity function on A* is the function $1_A : A \longrightarrow A$ with $1_A(a) = a$ f.a. $a \in A$.

The *composition* of total functions $f : A \longrightarrow B$ and $g : B \longrightarrow C$ yields the total function $g \circ f : A \longrightarrow C$ with: $(g \circ f)(a) = g(f(a))$ f.a. $a \in A$.

Be $f : A \longrightarrow B$ total and *bijective*, then the *inverse of f* is the function $f^{-1} : B \longrightarrow A$ with $f^{-1}(f(a)) = a$ f.a. $a \in A$.

$f_a^b(c)$ denotes the function with: $f_a^b(c) = \begin{cases} b & \text{if } c = a \\ f(c) & \text{otherwise} \end{cases}$

Let $f : A \longrightarrow B$ and $C \subseteq A$, then: $f(C)$ denotes $\{f(c) \mid c \in C\}$

Simplified Notation for Families of Functions

Let $A = (A_i)_{i \in I}$ and $B = (B_i)_{i \in I}$ be families of sets (same I):

$f : A \longrightarrow B$ denotes family $f = (f_i)_{i \in I}$ of relations $f_i : A_i \longrightarrow B_i$, $i \in I$.

A family f is said to have a property if each f_i, $i \in I$ has the property.

Example: "f is bijective".

(Motivation: Homomorphisms on mutli-sorted algebras are such families.)

Classes
(not so important)

classes: a set-theoretic notion, no precise definition here

historically motivated by the appearance of paradoxes in set theory

rough description:

- *class*: any collection of elements
- *set*: a class which is element of some other class

examples: - 'the class of all sets' is not a set
- 'the class of all Σ-algebras' (see below) is not a set

Part 3: Many-Sorted Algebras

Signatures

a *signature* is a pair (S, Ω) of sets.

elements of S resp. Ω called *sorts* reps. *operations*

an *operation* is a $(k+2)$-tuple $(n, s_1, \ldots, s_k, s)$, written '$n : s_1 \times \ldots \times s_k \to s$'

with $s_1, \ldots, s_k, s \in S$ and $k \geq 0$

n called *operation name*

$s_1 \times \ldots \times s_k \to s$ called *arity*

$s_1, \ldots, s_k$ resp. s are called *argument sorts* resp. *target sort*

if $k = 0$, an operation $n : \to s$ is called *constant of sort s*.

(often, we write n instead of $n : s_1 \times \ldots \times s_k \to s$)

Examples of Signatures

SEMIGROUP $= (S_{SEMIGROUP}, \Omega_{SEMIGROUP})$
with:

$$\begin{array}{lcl} S_{SEMIGROUP} & = & \{ \; dom \; \} \\ \Omega_{SEMIGROUP} & = & \{ + : dom \times dom \to dom \} \end{array}$$

MONOID $= (S_{SEMIGROUP}, \Omega_{MONOID})$
with:

$$\Omega_{MONOID} = \Omega_{SEMIGROUP} \cup \{ 0 :\to dom \}$$

GROUP $= (S_{SEMIGROUP}, \Omega_{GROUP})$
with:

$$\Omega_{GROUP} = \Omega_{MONOID} \cup \{ - : dom \to dom \}$$

Examples of Signatures

BOOL $= (S_{BOOL}, \Omega_{BOOL})$

with:

$$S_{BOOL} \;=\; \{ \; bool \; \}$$

$$\begin{array}{lcllcl} \Omega_{BOOL} & = & \{ & True & : & \to bool, \\ & & & False & : & \to bool, \\ & & & \neg & : & bool \to bool, \\ & & & \wedge & : & bool \times bool \to bool \; \} \end{array}$$

Examples of Signatures

NAT $= (S_{NAT}, \Omega_{NAT})$

with:

$$S_{NAT} \;=\; \{ \; nat \; \}$$

$$\begin{array}{lcllcl} \Omega_{NAT} & = & \{ & 0 & : & \to nat, \\ & & & Succ & : & nat \to nat \; \} \end{array}$$

NAT$^{+} = (S_{NAT}, \Omega_{NAT} \cup \{ + : nat \times nat \to nat \})$

NAT$^{infiniteConst} = (S_{NAT}, \Omega_{NAT} \cup \{ 1 :\to nat, \; 2 :\to nat, \; \ldots \})$

Examples of Signatures

NATBOOL $= (S_{NATBOOL}, \Omega_{NATBOOL})$

with:

$$S_{NATBOOL} \;=\; \{ \; nat, bool \; \}$$

$$\begin{array}{lcl} \Omega_{NATBOOL} & = & \Omega_{BOOL} \cup \Omega_{NAT} \\ & & \cup \; \{ \;\; + \; : \; nat \times nat \to nat, \\ & & \qquad\; * \; : \; nat \times nat \to nat, \\ & & \qquad\; \leq \; : \; nat \times nat \to bool \;\; \} \end{array}$$

NATBOOL constitutes the signature of *Peano arithmetic*

Examples of Signatures

NATSTACK $= (S_{NATSTACK}, \Omega_{NATSTACK})$

with:

$$\begin{array}{lcl} S_{NATSTACK} & = & \{ \; nat, bool, stack \; \} \\ \\ \Omega_{NATSTACK} & = & \Omega_{NATBOOL} \\ & & \cup \; \{ \quad \begin{array}{lcl} Emptystack & : & \rightarrow stack, \\ Push & : & stack \times nat \rightarrow stack, \\ Pop & : & stack \rightarrow stack, \\ Top & : & stack \rightarrow nat \quad \} \end{array} \end{array}$$

Algebras

an *algebra* assigns a *meaning to a signature*,
by assigning

- to each sort a *set*
- to each operation a *function*

note: such algebras are 'many-sorted': "to each sort a set"

Algebras formally

Let $\Sigma = (S, \Omega)$ be signature. An *algebra for* Σ (or Σ*-algebra*) assigns:

- to each $s \in S$ a no-empty set $A(s)$, called *carrier set* of s
- to each operation $(n : s_1 \times \ldots \times s_k \rightarrow s) \in \Omega$
  a total function $A(n : s_1 \times \ldots \times s_k \rightarrow s) : A(s_1) \times \ldots \times A(s_k) \longrightarrow A(s)$

note: when $k = 0$, $A(n :\rightarrow s)$ denotes an element of the carier set $A(s)$

note: our algebras are *total*

$Alg(\Sigma)$ denotes the class of all Σ-algebras

Examples of Algebras

A is the "classical" **NAT**-algebra

with:

$$\begin{array}{rcl} A(nat) & = & \mathbb{N}_0 \\ \\ A(0) & = & 0 \\ A(Succ) & : & \mathbb{N}_0 \longrightarrow \mathbb{N}_0 \quad \text{with} : A(Succ)(n) = n + 1 \end{array}$$

note: different '0' used

you might say: 'of course'

be careful: this is only one of many **NAT**-algebras

Examples of Algebras

another **NAT**-algebra: B

with:

$$\begin{array}{rcl} B(nat) & = & \{\mathsf{true}, \mathsf{false}\} \\ \\ B(0) & = & \mathsf{false} \\ B(Succ) & : & \{\mathsf{true}, \mathsf{false}\} \longrightarrow \{\mathsf{true}, \mathsf{false}\} \quad \text{with :} \\ & & B(Succ)(\mathsf{true}) = \mathsf{false} \\ & & B(Succ)(\mathsf{false}) = \mathsf{true} \end{array}$$

(*note:* true $\in B(nat)$, but $True \in \Omega_{BOOL}$)

Examples of Algebras

another **NAT**-algebra: C

with:

$$\begin{array}{rcl} C(nat) & = & \{\#\} \\ \\ C(0) & = & \# \\ C(Succ) & : & \{\#\} \longrightarrow \{\#\} \quad \text{with : } C(Succ)(n) = \# \end{array}$$

Relate different Algebras

two algebras can be related by *mappings which "respect" their functions*

'respecting' the functions roughly means:

it should not matter of we apply the mapping

- before or
- after

we apply the corresponding function

$\Rightarrow$ *'Homomorphisms'*

Homomorphisms

Let A and B be two Σ-Algebras. $\Sigma = (S, \Omega)$.

A *homomorphisms $h : A \longrightarrow B$ from A to B* is a family $(h_s)_{s \in S}$ of functions

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

such that for any $\omega \in \Omega$ with $\omega = (n : s_1 \times \ldots \times s_k \rightarrow s)$:

$$h_s(A(\omega)(a_1, \ldots, a_k)) = B(\omega)(h_{s_1}(a_1), \ldots, h_{s_k}(a_k))$$

('homomorphism condition')

for all $(a_1, \ldots, a_k) \in A(s_1) \times \ldots \times A(s_k)$.

My Advice

during the course week:

read the previous slide each night before you fall asleep

Homomorphisms

if $\omega = (n :\rightarrow s)$ is a constant, the homomorphism condition states $(k = 0)$:

$$h_s(A(\omega)) = B(\omega)$$

Isomorphisms

A *bijective homomorphism* is called *isomorphism*.

two Σ-Algebras A and B are *isomorphic* (written $A \simeq B$)
$:\Longleftrightarrow$
there exists an isomorphism from A to B

Intuition: A and B are 'identical up to renaming'.

Example: Relate different BOOL-Algebras by Homomorphisms

remind:

BOOL $= (S_{BOOL}, \Omega_{BOOL})$

with:

$$\begin{array}{lcl} S_{BOOL} & = & \{\ bool\ \} \\ \Omega_{BOOL} & = & \{\ \begin{array}{lcl} True & : & \rightarrow bool, \\ False & : & \rightarrow bool, \\ \neg & : & bool \rightarrow bool, \\ \wedge & : & bool \times bool \rightarrow bool\ \} \end{array} \end{array}$$

A Classical BOOL-Algebra

BOOL-Algebra A

with:

$$
\begin{aligned}
A(bool) &= \{\text{true}, \text{false}\} \\
A(True) &= \text{true} \\
A(False) &= \text{false} \\
A(\neg)(n) &= \begin{cases} \text{true} & \text{if } n = \text{false} \\ \text{false} & \text{otherwise} \end{cases} \\
A(\wedge)(n, m) &= \begin{cases} \text{true} & \text{if } n = m = \text{true} \\ \text{false} & \text{otherwise} \end{cases}
\end{aligned}
$$

Other BOOL-Algebras

BOOL-Algebra B

with:

$$
\begin{aligned}
B(bool) &= \{\#\} \\
B(True) &= \# \\
B(False) &= \# \\
B(\neg)(\#) &= \# \\
B(\wedge)(\#, \#) &= \#
\end{aligned}
$$

Other BOOL-Algebras

BOOL-Algebra C

with:

$$
\begin{aligned}
C(bool) &= \{0, 1\} \\
C(True) &= 1 \\
C(False) &= 0 \\
C(\neg)(n) &= \begin{cases} n + 1 & \text{if } n \text{ is even} \\ n - 1 & \text{otherwise} \end{cases} \\
C(\wedge)(n, m) &= n * m
\end{aligned}
$$

Other BOOL-Algebras

BOOL-Algebra D

with:

$$
\begin{aligned}
D(bool) &= \mathbb{N}_0 \\
D(True) &= 1 \\
D(False) &= 0 \\
D(\neg)(n) &= \begin{cases} n + 1 & \text{if } n \text{ is even} \\ n - 1 & \text{otherwise} \end{cases} \\
D(\wedge)(n, m) &= n * m
\end{aligned}
$$

Other BOOL-Algebras

BOOL-Algebra E

with:

$$\begin{aligned} E(bool) &= \mathbb{N}_0 \\ E(True) &= 1 \\ E(False) &= 0 \\ E(\neg)(n) &= n+1 \\ E(\wedge)(n,m) &= n+m \end{aligned}$$

Remark

These were **BOOL**-Algebras, i.e. Algebras for the signature **BOOL**.

BOOL-Algebras are not necessarily *Boolean algebras*. (Explanation later.)

Facts about these BOOL-Algebras

Here, A, B, C, D, and E refer to the **BOOL**-Algebras above.

Fact 3.1.
The function $h : A \longrightarrow B$ with $h(\text{true}) = h(\text{false}) = \#$ is a homomorphism. ⋆

Fact 3.2.
The function $g : A \longrightarrow C$ with $g(\text{true}) = 1$ and $g(\text{false}) = 0$ is an isomorphism. ⋆

Fact 3.3.
(i) There exists a homomorphism $k : A \longrightarrow D$.
(ii) There exists a homomorphism $l : D \longrightarrow A$.
(iii) (But) A and D are not isomorphic. ⋆

Fact 3.4.
There exists no homomorphism from A to E, nor from E to A. ⋆

Exercises

Exercise 3.1.
Prove some the facts 3.1, 3.2, 3.3, 3.4 (if not proved during the lecture).

Exercise 3.2.
*Pairwise examine the **NAT**-Algebras A, B, and C (pages 43, 44, 45) for the existence of homomorphisms, in each direction.*

Properties of Homomorphisms

Theorem 3.5.
The composition of two Σ-homomorphisms yields a Σ-homomorphism. $\star$

Theorem 3.6.
Let $\Sigma = (S, \Omega)$ be a signature and $h : A \longrightarrow B$ a Σ-isomorphism.
Then $h^{-1} = (h_s^{-1})_{s \in S}$ is a Σ-isomorphism from B to A. $\star$

Theorem 3.7.
The relation $\simeq$ on $Alg(\Sigma)$ is an equivalence relation.
(We deliberately carry over the definition of equivalence relations from sets to classes.) $\star$

Exercises

Exercise 3.3.
Prove some of the theorems 3.5, 3.6, 3.7 (if not proved during the lecture).

Initiality

Let $\mathcal{C} \subseteq Alg(\Sigma)$ be a class of Σ-Algebras for some signature Σ.

an algebra $A \in \mathcal{C}$ is *initial in the class $\mathcal{C}$*
$:\Longleftrightarrow$
for each $B \in \mathcal{C}$ there exists exactly one homomorphism from A to B

Example and Exercise

Consider the **BOOL**-Algebras A, B, C, and D.

Fact 3.8.
A is initial in $\{A, B\}$. $\star$

Exercise 3.4.
How many initial algebras does the set $\{A, B, C, D\}$ have? (zero, one, two, ...?)

Initaility of Isomorphic Algebras

Theorem 3.9.
Let $\mathcal{C} \subseteq Alg(\Sigma)$ be a class of Σ-Algebras for some signature Σ.
Assume A is initial in $\mathcal{C}$. Then:

$$B \text{ is initial in } \mathcal{C} \quad \Longleftrightarrow \quad A \simeq B$$

$\star$

Example: reconsider Fact 3.8 and Exercise 3.4.

Exercise

Exercise 3.5.
Prove theorem 3.9.

Closure under Isomorphism

Observation: Isomorphic algebras are 'similar' in the sense that they only differ in how the carrier elements are 'called'. The 'structure' of isomorphic algebras is the same.

Look ahead: Moreover, isomorphic algebras cannot be distinguished by logic.

Therefore, only certain classes are interesting:

A class $\mathcal{C} \subseteq Alg(\Sigma)$ is *closed under isomorphism*, if the following condition is satisfied: $A \in \mathcal{C}$ and $A \simeq B$ implies $B \in \mathcal{C}$ (f.a. Σ-Algebras A, B).

Abstract Data Types

An *abstract data type* for a signature Σ is any class of Σ-Algebras that is closed under isomorphism.

An abstract data type is called *monomorphic*, if its algebras are all isomorphic to each other. Otherwise, it is called *polymorphic*.

Informally:

- A *monomorphic abstract data type* stands for a 'single' data type.
- A *polymorphic abstract data type* stands for 'several' data types, which typically correspond to an 'incomplete' specification (see below).

Example

Consider the **BOOL**-Algebras A, B, and C.

Fact 3.10.
$\{J \in Alg(\boldsymbol{BOOL}) \mid J \simeq A\}$ *is an abstract data type, which is monomorphic and contains* C. $\star$

Fact 3.11.
$\{J \in Alg(\boldsymbol{BOOL}) \mid J \simeq A \ or \ J \simeq B\}$ *is a polymorphic abstract data type.* $\star$