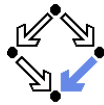


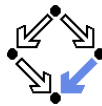
# Abstract Datatypes

Wolfgang Schreiner  
Wolfgang.Schreiner@risc.uni-linz.ac.at

Research Institute for Symbolic Computation (RISC)  
Johannes Kepler University, Linz, Austria  
<http://www.risc.uni-linz.ac.at>

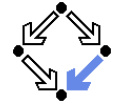


## Example



- $BOOL = (S_B, \Omega_B)$ .
  - $S_B = \{bool\}$ .
  - $\Omega_B = \{True : \rightarrow bool, False : \rightarrow bool, \neg : bool \rightarrow bool, \wedge : bool \times bool \rightarrow bool\}$ .
- $NATBOOL = (S_N, \Omega_N)$ .
  - $S_N = \{nat, bool\}$ .
  - $\Omega_N = \{0 : \rightarrow nat, Succ : nat \rightarrow nat, \leq : nat \times nat \rightarrow bool\}$ .
- $NATSTACK = (S, \Omega)$ .
  - $S = \{nat, bool, stack\}$ .
  - $\Omega = \{Emptystack : \rightarrow stack, Push : stack \times nat \rightarrow stack, Pop : stack \rightarrow stack, Top : stack \rightarrow nat\}$ .

# Signatures

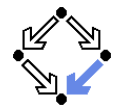


Our goal is to model abstract data types.

- A **signature**  $\Sigma = (S, \Omega)$ .
  - $S$  ... set of **sorts**.
  - $\Omega$  ... set of **operations** of form  $n : s_1 \times \dots \times s_k \rightarrow s$ .
    - $s_1, \dots, s_k, s \in S, k \geq 0$ .
    - **operation name**  $n$ .
    - **argument sorts**  $s_1 \times \dots \times s_k$ .
    - **target sort**  $s$ .
    - **arity**  $s_1 \times \dots \times s_k \rightarrow s$ .
    - Case  $k = 0$ : **constant**  $n : \rightarrow s$  of sort  $s$ .

A signature models the syntactic interface of an abstract data type.

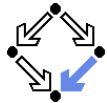
## Many-Sorted Algebras



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

- A (**many-sorted**) **algebra**  $A$  for  $\Sigma$  (a  $\Sigma$ -**algebra**  $A$ ):
  - A **carrier set**  $A(s)$ 
    - for each sort  $s \in S$ .
  - A **function**  $A(n : s_1 \times \dots \times s_k \rightarrow s) : A(s_1) \times \dots \times A(s_k) \rightarrow A(s)$ 
    - for each operation  $n : s_1 \times \dots \times s_k \rightarrow s \in \Omega$ .
    - (i.e., a carrier  $A(n : \rightarrow s)$  for each constant  $n : \rightarrow s \in \Omega$ ).
- An algebra assigns a meaning to a signature.
  - A set for each sort, a function for each operation.
- $Alg(\Sigma) := \{A : A \text{ is a } \Sigma\text{-algebra}\}$ .
  - The set of all  $\Sigma$ -algebras.

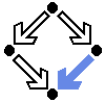
A  $\Sigma$ -algebra models a possible implementation of an abstract datatype.



## Example

- Signature  $\text{NAT} = (S_N, \Omega_N)$ .
  - $S_N = \{\text{nat}\}$ .
  - $\Omega_N = \{0 : \rightarrow \text{nat}, \text{Succ} : \text{nat} \rightarrow \text{nat}\}$ .
- NAT-algebra  $A$ :
  - $A(\text{nat}) = \mathbb{N}$ .
  - $A(0) = 0_{\mathbb{N}}$ .
  - $A(\text{Succ}) : \mathbb{N} \rightarrow \mathbb{N}$
  - $A(\text{Succ})(n) = n + 1$  (i.e.,  $A(\text{Succ}) = \lambda n.n + 1$ ).
- NAT-algebra  $B$ :
  - $B(\text{nat}) = \{\text{true}, \text{false}\}$ .
  - $B(0) = \text{false}$ .
  - $B(\text{Succ}) : \{\text{true}, \text{false}\} \rightarrow \{\text{true}, \text{false}\}$
  - $B(\text{Succ})(n) = \neg n$ .

Not all  $\Sigma$ -algebras behave in the “same” way.



## Homomorphisms

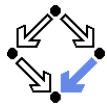
Take  $\Sigma$ -algebras  $A$  and  $B$  for signature  $\Sigma = (S, \Omega)$ .

- A  $\Sigma$ -homomorphism  $h : A \rightarrow B$  from  $A$  to  $B$ :
  - $h = (h_s)_{s \in S}$ .
    - A function for every sort in the signature.
  - $h_s : A(s) \rightarrow B(s)$ .
    - The function maps carrier set of  $A$  to corresponding carrier set of  $B$ .
  - $h_s(A(\omega)(a_1, \dots, a_k)) = B(\omega)(h_{s_1}(a_1), \dots, h_{s_k}(a_k))$ .
    - for every operation  $\omega = (n : s_1 \times \dots \times s_k \rightarrow s) \in \Omega$
    - and every tuple  $(a_1, \dots, a_k) \in A(s_1) \times \dots \times A(s_k)$ .

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

- For constant  $\omega$  ( $k = 0$ ):  $h_s(A(\omega)) = B(\omega)$ .

Homomorphism condition: the mappings are “compatible”.

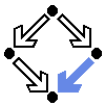


## Homomorphisms

How to interpret the existence of a homomorphism  $h : A \rightarrow B$ ?

- Functions  $A(\omega)$  and  $B(\omega)$  are “compatible”.
  - May first apply  $A(\omega)$  to arguments and then map the result to  $B$ .
  - Or may first map the arguments to  $B$  and then apply  $B(\omega)$ .
  - Both methods yield the same  $B$ -value.
- Carrier set of  $A$  has (at least) as much structure as carrier set of  $B$ .
  - If the  $B$ -counterparts  $b_1$  and  $b_2$  of the  $A$ -values  $a_1$  and  $a_2$  are different, then also  $a_1$  and  $a_2$  are different.
    - If  $b_1 = h(a_1) \neq b_2 = h(a_2)$ , we have  $h(a_1) \neq h(a_2)$ , and thus  $a_1 \neq a_2$ .
  - Nevertheless, different  $A$ -values  $a_1$  and  $a_2$  may have identical  $B$ -counterparts  $b_1$  and  $b_2$ .
    - Also if  $a_1 \neq a_2$ , it may be the case that  $h(a_1) = h(a_2)$ .

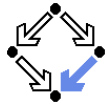
Guidelines for the intuition about homomorphism relation.



## Isomorphisms

- An  $\Sigma$ -isomorphism is a bijective  $\Sigma$ -homomorphism.
  - Bijective: one-to-one mapping between  $A$  and  $B$ .
    - Surjective and injective.
  - Surjective:  $\forall b \in B(s) : \exists a \in A(s) : h_s(a) = b$ .
    - Every value of  $B(s)$  is the counterpart of some value of  $A(s)$ .
  - Injective:  $\forall a, a' \in A(s) : h_s(a) = h_s(a') \Rightarrow a = a'$ .
    - Different values of  $A(s)$  are mapped to different values of  $B(s)$ .
- Two  $\Sigma$ -algebras  $A$  and  $B$  are **isomorphic** ( $A \simeq B$ ):
  - There exists a  $\Sigma$ -isomorphism between  $A$  and  $B$ .
- The isomorphism-relation  $\simeq$  is an equivalence relation.
  - Has reflexivity, symmetry, transitivity.

Isomorphic  $\Sigma$ -algebras are “identical up to renaming”.



## Example

- Signature  $BOOL = (\{bool\}, \{True \rightarrow bool, False \rightarrow bool, \neg : bool \rightarrow bool, \wedge : bool \times bool \rightarrow bool\})$ .
- BOOL-algebra  $A$ :
 
$$A(bool) = \{true, false\}$$

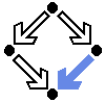
$$A(True) = true$$

$$A(False) = false$$

$$A(\neg)(n) := not(n) = \begin{cases} false, & \text{if } n = true \\ true, & \text{if } n = false \end{cases}$$

$$A(\wedge)(n, m) := and(n, m) = \begin{cases} true, & \text{if } n = m = true \\ false, & \text{otherwise} \end{cases}$$

The "classical" BOOL-algebra.



## Example (Contd)

- BOOL-algebra  $B$ :
 
$$B(bool) = \{\#\}$$

$$B(True) = B(False) = B(\neg)(\#) = B(\wedge)(\#, \#) = \#.$$
- BOOL-algebra  $C$ :
 
$$C(bool) = \{0, 1\}$$

$$C(True) = 1$$

$$C(False) = 0$$

$$C(\neg)(n) = 1 - n$$

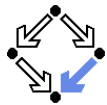
$$C(\wedge)(n, m) = n * m$$
- BOOL-algebra  $D$ :
 
$$D(bool) = \mathbb{N}$$

$$D(True) = 1$$

$$D(False) = 0$$

$$D(\neg)(n) = \begin{cases} n + 1, & \text{if } n \text{ is even} \\ n - 1, & \text{otherwise} \end{cases}$$

$$D(\wedge)(n, m) = n * m$$

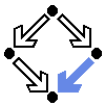


## Example (Contd'2)

How can all these BOOL-algebras be related?

- Homomorphism  $h : A \rightarrow B$ :
  - $h(true) = h(false) = \#$ .
- No homomorphism from  $B$  to  $A$ .
 

Assume homomorphism  $h : B \rightarrow A$ .  
Then  $h(\#) = h(B(\neg)(\#)) = A(\neg)(h(\#)) = not(h(\#)) \neq h(\#)$ .
- Isomorphism  $g : A \rightarrow C$ :
  - $g(true) = 1, g(false) = 0$ .
  - $g^{-1}(1) = true, g^{-1}(0) = false$ .
- $A$  and  $D$  are not isomorphic.
  - No bijection between  $\{true, false\}$  and  $\mathbb{N}$ .
- Homomorphisms  $k : A \rightarrow D$  and  $l : D \rightarrow A$ .
  - $k(true) = 1, k(false) = 0$ .
  - $l(n) = (n \text{ is even})$ .



## Example (Contd'3)

- BOOL-algebra  $E$ :
 
$$E(bool) = \mathbb{N}$$

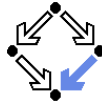
$$E(True) = 1$$

$$E(False) = 0$$

$$E(\neg)(n) = n + 1$$

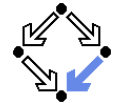
$$E(\wedge)(n, m) = n + m$$
- Neither a homomorphism from  $A$  to  $E$  nor one from  $E$  to  $A$ .
  - Assume homomorphism  $h : A \rightarrow E$ .  
Then  $h(false) = h(A(\neg)(true)) = E(\neg)(h(true)) = h(true) + 1$ .  
Also  $h(true) = h(A(\neg)(false)) = E(\neg)(h(false)) = h(false) + 1$ .  
But then  $h(false) = h(true) + 1 = (h(false) + 1) + 1 = h(false) + 2$ .
  - Assume homomorphism  $g : E \rightarrow A$ .  
Then  $g(1) = g(E(\neg)(0)) = A(\neg)(g(0)) = not(g(0))$ .  
Also  $g(1) = g(E(\wedge)(1, 0)) = A(\wedge)(g(1), g(0)) = and(not(g(0)), g(0)) = false$ .  
Also  $g(2) = g(E(\neg)(1)) = A(\neg)(g(1)) = not(g(1)) = true$ .  
But also  $g(2) = g(E(\wedge)(1, 1)) = A(\wedge)(g(1), g(1)) = and(false, false) = false$ .

# Abstract Data Types



- A **datatype**:
  - An equivalence class of isomorphic  $\Sigma$ -algebras.
    - A class  $[A] = \{B \in Alg(\Sigma) : B \simeq A\}$  (for some  $\Sigma$ -algebra  $A$ ).
  - The elements of such a class are identical up to renaming.
    - Thus we do not consider individual  $\Sigma$ -algebras as datatypes.
- An **abstract data type (ADT)**:
  - A class of  $\Sigma$ -algebras closed under isomorphism.
    - A class  $C \subseteq Alg(\Sigma)$ .
    - If  $A \in C$  and  $A \simeq B$ , then  $B \in C$  (for any  $\Sigma$ -algebras  $A$  and  $B$ ).
  - Every ADT  $C$  can be decomposed into datatypes:
    - $C = \bigcup \{[A] : A \in C\}$ .
    - All the datatypes that can implement the ADT.
  - An ADT is **monomorphic** if all its elements are isomorphic.
    - The ADT can be implemented by a single datatype.
  - A non-monomorphic ADT is **polymorphic**.
    - The ADT can be implemented by multiple datatypes.

# Example



Take the BOOL-algebras of the previous example.

- ADT  $\mathcal{A} := \{J \in Alg(BOOL) : J \simeq A\}$ 
  - All algebras isomorphic to the classical the BOOL-algebra  $A$ .
  - A monomorphic ADT with a single datatype  $[A]$  containing  $C$ .
- ADT  $\mathcal{B} := \{J \in Alg(BOOL) : J \simeq A \vee J \simeq B\}$ 
  - A polymorphic ADT with two datatypes  $[A]$  and  $[B]$ .

We need a language to specify abstract datatypes in a convenient way.