Algebraic Specification - Equational Logic

Specification techniques' requirements:

- Abstraction (refinement)
- Structuring mechanisms
  Partition-aggregation, combination, extension-instantiation
- Clear (explicit and plausible) semantics
- Support of the „verify while develop"-principle
- Expressiveness (all the partial recursive functions representable)
- Readability (adequacy) (suitability)

  ⋮

Algebraic Specification - Algebras

Specification of data types

Syntax	Equations	Programs
signature	$t_1 = t_2$	data operations
axiom	if φ then $t_1 = t_2$	directed application

Algebras

heterogeneous	order-sorted	homogeneous
(Many-Sorted)	(Many-Sorted)	(Single-Sorted)

Single-Sorted Algebras

Example 6.1. *a) Groups*
SORT:: g
SIG:: $\cdot : g, g \rightarrow g \qquad 1 : \rightarrow g \qquad {}^{-1} : g \rightarrow g$
EQN:: $x \cdot 1 = x \qquad x \cdot x^{-1} = 1 \qquad (x \cdot y) \cdot z = x \cdot (y \cdot z)$
All-quantified equations

Models are groups

Question: Which equations are valid in all groups,
i.e. $EQN \models t_1 = t_2$

$$1 \cdot x = x \qquad x^{-1} \cdot x = 1 \qquad (x^{-1})^{-1} = x$$

Single-Sorted Algebras

Equational Logic: Replace „equals“ with „equals“
Problem: cycles, non-termination
Solution: Directed equations $\rightsquigarrow$ Term rewriting systems

Find R „convergent“ with $\underset{EQN}{=} = \overset{*}{\underset{R}{\Longleftrightarrow}}$

$$
\begin{array}{ll}
x \cdot 1 \to x & 1 \cdot x \to x \\
x \cdot x^{-1} \to 1 & x^{-1} \cdot x \to 1 \\
1^{-1} \to 1 & (x^{-1})^{-1} \to x \\
(x \cdot y)^{-1} \to y^{-1} \cdot x^{-1} & (x \cdot y) \cdot z \to x \cdot (y \cdot z) \\
x^{-1} \cdot (x \cdot y) \to y & x \cdot (x^{-1} \cdot y) \to y
\end{array}
$$

Many-Sorted Algebras

b) Lists over nat-numbers

SIG: BOOL, NAT, LIST Sorts
true, false: $\to$ BOOL
0 $\to$ NAT
suc: NAT $\to$ NAT
$+$: NAT, NAT $\to$ NAT
eq: NAT, NAT $\to$ BOOL
nil: $\to$ LIST
. : NAT, LIST $\to$ LIST
app: LIST, LIST $\to$ LIST
rev: LIST $\to$ LIST

Many-Sorted Algebras

Axioms are all-quantified equations, i.e.
$\forall x_1, \ldots, x_n, y_1, \ldots, y_m : \quad t_1(x_1, \ldots, x_n) = t_2(y_1, \ldots, y_m)$ where

$t_1(x_1, \ldots, x_n), t_2(y_1, \ldots, y_m)$ Terms of the same sort over the signature.

$$\text{EQN}: \quad n + 0 = n \quad n + \text{suc}(m) = \text{suc}(n + m)$$

$$\text{eq}(0, 0) = \text{true} \quad \text{eq}(0, \text{suc}(n)) = \text{false}$$
$$\text{eq}(\text{suc}(n), 0) = \text{false}$$
$$\text{eq}(\text{suc}(n), \text{suc}(m)) = \text{eq}(n, m)$$

$$\text{app}(\text{nil}, l) = l \quad \text{app}(n.l_1, l_2) = n.\,\text{app}(l_1, l_2)$$

$$\text{rev}(\text{nil}) = \text{nil} \quad \text{rev}(n.l) = \text{app}(\text{rev}(l), n.\text{nil})$$

Many-Sorted Algebras

Terms of type BOOL, NAT, LIST as identifiers for elements. (standard definition!)

Which algebra is specified? How can we compute in this algebra?

Direct the equations $\rightsquigarrow$ term-rewriting system R. Evidently e.g.:

$$s^i(0) + s^j(0) \xrightarrow[R]{*} s^{i+j}(0)$$

$$\text{app}(3.1.\text{nil}, \text{app}(5.\text{nil}, 1.2.3.\text{nil})) \xrightarrow[R]{*} 3.1.5.1.2.3.\text{nil}$$

$$\begin{aligned}
\text{rev}(3.1.\text{nil}) &\to \text{app}(\text{rev}(1.\text{nil}), 3.\text{nil}) \\
&\to \text{app}(\text{app}(\text{rev}(\text{nil}), 1.\text{nil}), 3.\text{nil}) \\
&\to \text{app}(\text{app}(\text{nil}, 1.\text{nil}), 3.\text{nil}) \\
&\to \text{app}(1.\text{nil}, 3.\text{nil}) \xrightarrow{*} 1.3.\text{nil}
\end{aligned}$$

Question: Is $\text{app}(x.y.\text{nil}, z.\text{nil}) =_E \text{app}(x.\text{nil}, y.z.\text{nil})$ true?

Many-Sorted Algebras

Some equations are not valid in all the models of EQN$= E$.
e.g.

$$x + y \neq_E y + x$$
$$\mathrm{app}(x, \mathrm{app}(y, z)) \neq_E \mathrm{app}(\mathrm{app}(x, y), z)$$
$$\mathrm{rev}(\mathrm{rev}(x)) \neq_E x$$

The pairs of terms cannot be joined via rewriting.

Distinction:

- Equations that are valid in all the models of E.
- Equations that are valid in data models of E.

$x + y = y + x :: s^i 0 + s^j 0 = s^j 0 + s^i 0$ all i, j
$\mathrm{rev}(\mathrm{rev}(x)) = x$ for $x \equiv s^{i_1} 0.s^{i_2} 0. \ldots s^{i_n} 0.\mathrm{nil}$

Thesis: Data types are Algebras

ADT: Abstract data types. Independent of the data representation.

Specification of abstract data types:

Concepts from Logic/universal Algebra
Objective: common language for specification and implementation.

Methods for proving correctness:

Syntax, L formulae (P-Logic,Hoare,...)

Cl: Consequence closure (e.g. $\models$, $Th(A), \ldots$)

Consequence closure

$Cl : \mathbb{P}(L) \to \mathbb{P}(L)$ (subsets of L) with

a) $A \subset L \leadsto A \subset Cl(A)$
b) $A, B \subset L, A \subseteq B \leadsto Cl(A) \subseteq Cl(B)$ (Monotonicity)
c) $Cl(A) = Cl(Cl(A))$ (Maximality)

Important concepts:

Consistency: $A \subsetneq L$ A is consistent if $Cl(A) \subsetneq L$
Implementation: A (over L') implements B (over L) (Refinement)

$$L \subset L', Cl(B) \subseteq Cl(A)$$

Related to implication.

Signature - Terms

Definition 6.2. *a) Signature is a triple* $\mathrm{sig} = (S, F, \tau)$ *(abbreviated:* Σ*)*

- *S finite set of sorts*
- *F set of operators (function symbols)*
- *$\tau : F \to S^+$ arity function, i.e.*
  $\tau(f) = s_1 \cdots s_n\ s$, $n \geq 0$, *s_i argument's sorts, s target sort.*

Write: $f : s_1, \ldots, s_n \to s$

(Notice that $n = 0$) is possible, constants of sort S.

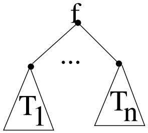
Signature - Terms

b) Term(F): Set of ground terms over sig and their tree presentation.

$$\text{Term}(F) := \dot{\bigcup_{s \in S}} \text{Term}_s(F)$$

recursive definition:

- $f :\to s$, so $f \in \text{Term}_s(F)$ representation: $\cdot f$
- $f : s_1, \ldots, s_n \to s$, $t_i \in \text{Term}_{s_i}(F)$ with rep. T_i so $f(t_1, \ldots, t_n) \in \text{Term}_s(F)$ with rep.



Consider the representation by ordered trees

Signature - Terms

c) $V = \dot{\bigcup}_{s \in S} V_s$ system of variables $V \cap F = \varnothing$.

Each $x \in V_s$ has arity $x :\rightarrow s$

Set: $\text{Term}(F, V) := \text{Term}(F \cup V)$.

Quotation: terms over sig in the variables V.
(F and τ extended with the set of variables and their sorts).

Intention: for variables it is allowed to use any object of the same sort, i.e. terms of this sort. "Placeholder" for an arbitrary object of this sort.

Strictness - Positions- Subterms

Definition 6.3. *a)* $s \in S$ *strict, if* $\text{Term}_s(F) \neq \emptyset$
If for each sort $s \in S$ *there is a constant of sort* S *or a function* $f : s_1, \ldots, s_n \rightarrow s$*, so that the* s_i *are strict. If all the sorts of the signature are strict.* $\rightsquigarrow$ *strict signatures (general assumption)*

b) Subterms $(t) = \{t_p \mid p$ *location (position) in* p*,* t_p *subterm in* $p\}$
The positions are represented by sequences over $\mathbb{N}$
(elements of $\mathbb{N}^*$*,* e *the empty sequence).*
$O(t)$ *Set of positions in* t*,*
For $p \in O(t)$ t_p *(or* $t|_p$*) subterm of* t *in position* p

- t *constant or variable:* $O(t) = \{e\}$ $t_e \equiv t$
- $t \equiv f(t_1, \ldots, t_n)$ *so*
  $O(t) = \{ip \mid 1 \leq i \leq n, p \in O(t_i)\} \cup \{e\}$
  $t_{ip} \equiv t_i|_p$ *and* $t_e \equiv t$.

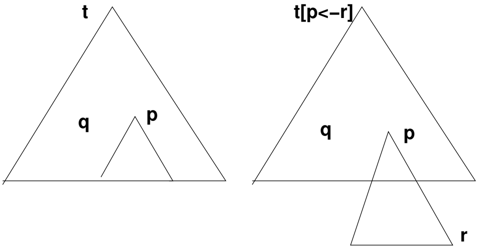
Term replacement

c) Term replacement: $t, r \in \text{Term}(F, V)$
$p \in O(t)$: with $r, t_p \in \text{Term}_s(F, V)$ for a sort s.

Then
$t[r]_p$, $t[p \leftarrow r]$ respectively t_p^r is the term, that is obtained from t by replacing subterm t_p by r.

So $t[p \leftarrow r]_q = t_q$ for $q \mid p$ and

$$t[p \leftarrow r]_p = r$$



Signatures - terms

Example 6.4. $S = (\text{BOOL}, \text{NAT}, \text{LIST})$, $F = \{true, false, \dots\}$,
$\tau : F \to S^* :: true :\to \text{BOOL},\ eq : \text{NAT}, \text{NAT} \to \text{BOOL}, \dots$

$$V = \underbrace{V_{\text{BOOL}}}_{\{b_i \,:\, i \in \mathbb{N}\}} \cup \underbrace{V_{\text{NAT}}}_{\{x_i \,:\, i \in \mathbb{N}\}} \cup \underbrace{V_{\text{LIST}}}_{\{l_i \,:\, i \in \mathbb{N}\}}$$

Ground terms:

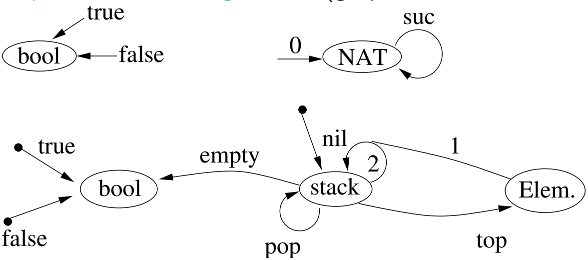
$true, false, eq(0, \text{suc}(0)) \in \text{Term}_{\text{BOOL}}(S)$
$0, \text{suc}(0), \text{suc}(0) + (\text{suc}(\text{suc}(0)) + 0) \in \text{Term}_{\text{NAT}}(S)$
$\text{app}(nil, \text{suc}(0).(\text{suc}(\text{suc}(0)).nil) \in \text{Term}_{\text{LIST}}(S)$
$0.\,\text{suc}(0), eq(true, false), \text{rev}(0)$ *no terms.*

General terms:

$eq(x_1, x_2) \in \text{Term}_{\text{BOOLE}}(F, V), suc(x_1) + (x_2 + \text{suc}(0)) \in \text{Term}_{\text{NAT}}(F, V)$
$\text{app}(l_1, x_1.l_0) \in \text{Term}_{\text{LIST}}(F, V)$
$\text{rev}(x_1.l) \in \text{Term}_{\text{LIST}}(F, V)$
$\text{app}(x_1, l_2)$ *no term.*

Signatures

Representation of signatures (graphical or standardized)



Notations:

<u>sig</u> . . .

<u>sorts</u> . . .

<u>ops</u> . . .

op: $W \to S$

$\mathrm{op}_1, \dots, \mathrm{op}_i : W \to S$

Interpretations: sig-Algebras

Definition 6.5. sig $= (S, F, \tau)$ *signature. A* *sig-Algebra* $\mathfrak{A}$ *is composed of*

1) *Set of support* $A = \bigcup_{s \in S} A_s, A_s \neq \emptyset$ *set of support of sort* s.
2) *Function system* $F_{\mathfrak{A}} = \{f_{\mathfrak{A}} : f \in F\}$ *with*
   $f_{\mathfrak{A}} : A_{s_1} \times \cdots \times A_{s_n} \to A_s$ *function and* $\tau(f) = s_1 \cdots s_n s$.

Notice: The $f_{\mathfrak{A}}$ are total functions.

The precondition $A_s \neq \emptyset$ is not mandatory.

Interpretations: sig-Algebras

Example 6.6. *a)* sig $\equiv$ BOOL, *true*, *false* $:\to$ BOOL

$$\left.\begin{array}{lcll} \mathfrak{A}_1 & \{0,1\} & true_{\mathfrak{A}_1} = 0 & false_{\mathfrak{A}_1} = 1 \\ \mathfrak{A}_2 & \{0,1\} & true_{\mathfrak{A}_2} = 0 & false_{\mathfrak{A}_2} = 0 \\ \mathfrak{A}_3 & \mathbb{N} & true_{\mathfrak{A}_3} = 4 & false_{\mathfrak{A}_3} = 5 \\ \mathfrak{A}_4 & \{true, false\} & true_{\mathfrak{A}_4} = true & false_{\mathfrak{A}_4} = false \end{array}\right\} \textit{bool-Alg.}$$

b) sig $\equiv$ NAT, $0, \mathrm{suc}$

$$\left\{\begin{array}{cccccc} A_{i_{\mathrm{NAT}}} & \mathbb{N} & \mathbb{Z} & \mathbb{N} & \{true, false\} & \{0, suc^i(0)\} \\ 0_{\mathfrak{A}_i} & 0 & 0 & 1 & true & 0 \\ \mathrm{suc}_{\mathfrak{A}_i} & \mathrm{suc}_{\mathbb{N}} & \mathrm{pred}_{\mathbb{Z}} & \mathrm{id}_{\mathbb{N}} & \mathrm{suc}(true) = false & \mathrm{suc}(0) = suc(0) \\ & & & & \mathrm{suc}(false) = true & \mathrm{suc}(suc^i(0)) = suc^{i+1}(0) \end{array}\right.$$

Free sig-algebra generated by V

Definition 6.7.

- $\mathfrak{A} = (A, F_{\mathfrak{A}})$ *with:* $A = \bigcup_{s \in S} A_s$ $A_s = \text{Term}_s(F, V)$, *i.e.* $A = \text{Term}(F, V)$
  $F \ni f : s_1, \ldots, s_n \to s$, $f_{\mathfrak{A}}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$

  $\mathfrak{A}$ *is sig-Algebra::* $T_{\text{sig}}(V)$
  the free termalgebra in the variables V generated by V
- $V = \emptyset$: $A_s = \text{Term}_s(F)$ *set of ground terms* ($A_s \neq \emptyset$, *because sig is strict).*

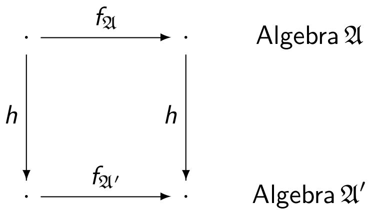
  $\mathfrak{A}$ *ground termalgebra::* T_{sig}

Homomorphisms

Definition 6.8 (sig-homomorphism). $\mathfrak{A}, \mathfrak{A}'$ *sig-algebras*
$h : \mathfrak{A} \rightarrow \mathfrak{A}'$ *family of functions*
$h = \{h_s : A_s \rightarrow A'_s : s \in S\}$ *is sig-homomorphism*
when

$$h_s(f_{\mathfrak{A}}(a_1, \ldots, a_n)) = f_{\mathfrak{A}'}(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$$

As always: injective, surjective, bijective, isomorphism



Canonical homomorphisms

Lemma 6.9. *$\mathfrak{A}$ sig-Algebra, T_{sig} ground term algebra*

a) *The family of canonical interpretation functions $h_s : \text{Term}_s(F) \to A_s$ defined through*

$$h_s(f(t_1, \ldots, t_n)) = f_{\mathfrak{A}}(h_{s_1}(t_1), \ldots, h_{s_n}(t_n))$$

with $h_s(c) = c_{\mathfrak{A}}$ is a sig-homomorphism.

b) *There is no other sig-homomorphism from T_{sig} to $\mathfrak{A}$.* *Uniqueness!*

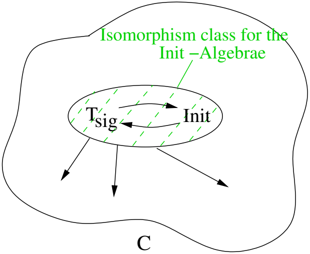
Proof: Just try!!

Initial algebras

Definition 6.10 (Initial algebras). *A sig-Algebra $\mathfrak{A}$ is called initial in a class C of sig-algebras, if for each sig-Algebra $\mathfrak{A}' \in C$ exists exactly one sig-homomorphism $h : \mathfrak{A} \to \mathfrak{A}'$.*

***Notice**: T_{sig} is initial in the class of all sig-algebras (Lemma 6.9).*

Fact: Initial algebras are isomorphic.



The final algebras can be defined analogously.

Canonical homomorphisms

$\mathfrak{A}$ sig-Algebra, $h : T_{\text{sig}} \to \mathfrak{A}$ interpretation homomorphism.
$\mathfrak{A}$ sig-generated (term-generated) iff
$\forall s \in S \quad h_s : \text{Term}_s(F) \to A_s$ surjective

The ground termalgebra is sig-generated.

ADT requirements:

- Independent of the representation (isomorphism class)
- Generated by the operations (sig-generated)
  Often: constructor subset

Thesis: An ADT is the isomorphism class of an initial algebra.

Ground termalgebras as initial algebras are ADT.

Notice by the properties of free termalgebras : functions from V in $\mathfrak{A}$ can be extended to unique homomorphisms from $T_{sig}(V)$ in $\mathfrak{A}$.

Equational specifications

For Specification's formalisms:

Classes of algebras that have initial algebras.

$\leadsto$ Horn-Logic (See bibliography)

sig INT sorts int
ops $0 :\longrightarrow$ int
suc : int $\longrightarrow$ int
pred : int $\longrightarrow$ int

Equational specifications

Definition 6.11. $\text{sig} = (S, F, \tau)$ *signature, V system of variables.*

a) *Equation:* $(u, v) \in \text{Term}_s(F, V) \times \text{Term}_s(F, V)$

Write: $u = v$

Equational system E over sig, V: Set of equations E

b) *(Equational)-specification:* $\text{spec} = (\text{sig}, E)$

where E is an equational system over $F \cup V$.

Notation

Keyword eqns

spec INT	
sorts int	implicit
ops $0 :\to$ int	All-Quantification
suc, pred: int $\to$ int	often also a declaration
eqns $\mathrm{suc}(\mathrm{pred}(x)) = x$	of the sorts
$\mathrm{pred}(\mathrm{suc}(x)) = x$	of the variables

Semantics::

- loose all models (PL1)
- tight (special model initial, final)
- operational (equational calculus + induction principle)

Models of spec $= (\text{sig}, E)$

Definition 6.12. $\mathfrak{A}$ *sig-Algebra, $V(S)$- system of variables*

a) *Assignment function* φ *for* $\mathfrak{A}$: $\varphi_s : V_s \to A_s$ *induces a*

valuation $\varphi : \text{Term}(F, V) \to \mathfrak{A}$ *through*

$\varphi(f) = f_{\mathfrak{A}}$, f *constant*, $\varphi(x) := \varphi_s(x)$, $x \in V_s$
$\varphi(f(t_1, \ldots, t_n)) = f_{\mathfrak{A}}(\varphi(t_1), \ldots, \varphi(t_n))$

$$V_s \xrightarrow{\varphi_s} A_s$$
$$\text{Term}_s(F, V) \xrightarrow{\varphi_s} A_s$$
$$\text{Term}(F, V) \xrightarrow{\varphi} \mathfrak{A} \qquad \textit{homomorphism}$$

(Proof!)

Models of spec = (sig, E)

b) $s = t$ equation over sig, V
$\mathfrak{A} \models_{\varphi} s = t$: $\mathfrak{A}$ satisfies $s = t$ with assignment φ iff $\varphi(s) = \varphi(t)$,
equality in A.

c) $\mathfrak{A}$ satisfies $s = t$ or $s = t$ holds in $\mathfrak{A}$
$\mathfrak{A} \models s = t$: for each assigment φ
$\mathfrak{A} \models_{\varphi} s = t$

d) $\mathfrak{A}$ is model of spec = (sig, E)
iff $\mathfrak{A}$ satisfies each equation of E
$\mathfrak{A} \models E$ ALG(spec) class of the models of spec.

Examples

Example 6.13. *1)*

spec NAT
sorts nat
ops $0 :\to \text{nat}$
$s : \text{nat} \to \text{nat}$
$\_ + \_ : \text{nat}, \text{nat} \to \text{nat}$
eqns $x + 0 = x$
$x + s(y) = s(x + y)$

Examples

sig-algebras

a) $\mathfrak{A} = (\mathbb{N}, \hat{0}, \hat{+}, \hat{s})$
$\hat{0} = 0 \quad \hat{s}(n) = n + 1 \quad n \hat{+} m = n + m$

b) $\mathfrak{B} = (\mathbb{Z}, \hat{0}, \hat{+}, \hat{s})$
$\hat{0} = 1 \quad \hat{s}(i) = i \cdot 5 \quad i \hat{+} j = i \cdot j$

c) $\mathfrak{C} = (\{\text{true}, \text{false}\}, \hat{0}, \hat{+}, \hat{s})$
$\hat{0} = \text{false} \quad \hat{s}(\text{true}) = \text{false} \quad \hat{s}(\text{false}) = \text{true}$
$i \hat{+} j = i \vee j$

Examples

$\mathfrak{A}, \mathfrak{B}, \mathfrak{C}$ are models of spec NAT

$$\text{e.g.} \quad \mathfrak{B}: \quad \varphi(x) = a \quad \varphi(y) = b \quad a, b \in \mathbb{Z}$$

$$\varphi(x + 0) = a \hat{+} \hat{0} = a \cdot 1 = a = \varphi(x)$$

$$\begin{aligned} \varphi(x + s(y)) &= a \hat{+} \hat{s}(b) = a \cdot (b \cdot 5) \\ &= (a \cdot b) \cdot 5 = \hat{s}(a \hat{+} b) \\ &= \varphi(s(x + y)) \end{aligned}$$

Examples

2)

spec	LIST(NAT)
use	NAT
sorts	nat, list
ops	nil :$\rightarrow$ list
	$\_.\_$: nat, list $\rightarrow$ list
	app : list, list $\rightarrow$ list
eqns	$\text{app}(\text{nil}, q_2) = q_2$
	$\text{app}(x.q_1, q_2) = x.\,\text{app}(q_1, q_2)$

Examples

spec-Algebra

$$\mathfrak{A} \qquad \mathbb{N}, \mathbb{N}^*$$
$$\hat{0} = 0 \quad \hat{+} = + \quad \hat{s} = +1$$
$$\hat{\text{nil}} = e \qquad \text{(emptyword)}$$
$$\hat{.}\ (i, z) = i\ z$$
$$\widehat{\text{app}}(z_1, z_2) = z_1 z_2 \text{ (concatenation)}$$

Examples

3) spec INT $\quad suc(pred(x)) = x \quad pred(suc(x)) = x$

	1	2	3
A_{int}	$\mathbb{Z}$	$\mathbb{N}$	$\{true, false\}$
$0_{\mathfrak{A}_i}$	0	0	true
$suc_{\mathfrak{A}_i}$	$suc_{\mathbb{Z}}$	$suc_{\mathbb{N}}$	$\left\{ \begin{array}{c} true \rightarrow false \\ false \rightarrow true \end{array} \right\}$
$pred_{\mathfrak{A}_i}$	$pred_{\mathbb{Z}}$	$\left\{ \begin{array}{c} n+1 \rightarrow n \\ 0 \rightarrow 0 \end{array} \right\}$	$\left\{ \begin{array}{c} true \rightarrow false \\ false \rightarrow true \end{array} \right\}$
	+	−	+

Examples

	4	5	6
A_{int}	$\{a,b\}^* \cup \mathbb{Z}$	$\{1\}^+ \cup \{0\}^+ \cup \{z\}$	!
$0_{\mathfrak{A}_i}$	0	z	!
$\text{suc}_{\mathfrak{A}_i}$	$\text{suc}_{\mathbb{Z}}$	$\left\{ \begin{array}{c} 1^n \to 1^{n+1} \\ z \to 1 \\ 0^{n+1} \to 0^n \\ 0 \to z \end{array} \right\}$	id
$\text{pred}_{\mathfrak{A}_i}$	$\text{pred}_{\mathbb{Z}}$	$\left\{ \begin{array}{c} 1^{n+1} \to 1^n \\ 1 \to z \\ z \to 0 \\ 0^n \to 0^{n+1} \end{array} \right\}$	id
	$-$	$+$	$+$

Substitution

Definition 6.14 (sig, $\mathrm{Term}(F,V)$). σ:: $\sigma_s : V_s \to \mathrm{Term}_s(F,V)$, $\sigma_s(x) \in \mathrm{Term}_s(F,V)$, $x \in V_s$
$\sigma(x) = x$ *for almost every* $x \in V$

$D(\sigma) = \{x \mid \sigma(x) \neq x\}$ *finite::* *domain of* σ

Write $\sigma = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$

Extension to homomorphism $\sigma : \mathrm{Term}(F,V) \to \mathrm{Term}(F,V)$

$$\sigma(f(t_1,\ldots,t_n)) = f(\sigma(t_1),\ldots,\sigma(t_n))$$

Ground substitution: $t_i \in \mathrm{Term}_S(F) \quad x_i \in D(\sigma)_S$

Lose semantics

Definition 6.15. spec $= (\text{sig}, E)$
ALG(spec) $= \{\mathfrak{A} \mid$ *sig-Algebra*, $\mathfrak{A} \models E\}$ *sometimes alternatively*

$ALG_{TG}(spec) = \{\mathfrak{A} \mid$ *term-generated sig-Algebra*, $\mathfrak{A} \models E\}$

Find: Characterizations of equations that are valid in ALG(spec) or $\text{ALG}_{TG}(\text{spec})$.

a) *Semantical equality*: $E \models s = t$

b) *Operational equality*: $t_1 \underset{E}{\vdash\!\dashv} t_2$ *iff*
There is $p \in 0(t_1), s = t \in E$, *substitution* σ *with*
$t_1|_p \equiv \sigma(s),\ t_2 \equiv t_1[\sigma(t)]_p(t_1[p \leftarrow \sigma(t)])$
or $t_1|_p \equiv \sigma(t),\ t_2 \equiv t_1[\sigma(s)]_p$

$t_1 =_E t_2$ *iff* $t_1 \overset{*}{\underset{E}{\vdash\!\dashv}} t_2$
Formalization of replace equals $\leftrightarrow$ *equals*

Equality calculus

c) Equality calculus: Inference rules (deductive)

Reflexivity $\overline{t = t}$

Symmetry $\dfrac{t = t'}{t' = t}$

Transitivity $\dfrac{t = t', t' = t''}{t = t''}$

Replacement $\dfrac{t' = t''}{s[t']_p = s[t'']_p}$ $p \in 0(s)$

(frequently also with substitution σ)

Equality calculus

$E \vdash s = t$ iff there is a proof P for $s = t$ out of E, i.e.

$P =$ sequence of equations that ends with $s = t$, such that for $t_1 = t_2 \in P$.

i) $t_1 = t_2 \in \sigma(E)$ for a Substitution σ:

ii) $t_1 = t_2 \ldots$ out of precedent equations in P by application of one of the inference rules.

Properties and examples

Consequence 6.16 (Properties and Examples). a) *If either* $E \models s = t$ *or* $s =_E t$ *or* $E \vdash s = t$ *holds, then*

i) *If* σ *is a substitution, then also*

$E \models \sigma(s) = \sigma(t)$ / $\sigma(s) =_E \sigma(t)$ / $E \models \sigma(s) = \sigma(t)$
i.e. the induced equivalence relations on $\mathrm{Term}(F, V)$ *are stable w.r. to substitutions*

ii) $r \in \mathrm{Term}(F, V)$, $p \in 0(r)$, $r|_p$, $s, t \in \mathrm{Term}_{s'}(F, V)$ *then*

$E \models r[s]_p = r[t]_p$ / $r[s]_p =_E r[t]_p$ / $E \vdash r[s]_p = r[t]_p$
replacement property (monotonicity)

$\rightsquigarrow$ *Congruence on* $\mathrm{Term}(F, V)$ *which is stable.*

Congruences / Quotient algebras

b) $\mathfrak{A} = (A, F_{\mathfrak{A}})$ sig-Algebra. $\sim$ bin. relation on A is congruence relation over $\mathfrak{A}$, iff

i) $a \sim b \leadsto \exists s \in S : a, b \in A_s$ (sort compatible)

ii) $\sim$ is equivalence relation

iii) $a_i \sim b_i$ $(i = 1, \dots, n)$, $f_{\mathfrak{A}}(a_1, \dots, a_n)$ defined $\leadsto f_{\mathfrak{A}}(a_1, \dots, a_n) \sim f_{\mathfrak{A}}(b_1, \dots, b_n)$ (monotonic)

$\mathfrak{A}/\sim$ quotient algebra:
$A/\sim = \bigcup_{s \in S}(A_s/\sim)_s$ with $(A_s/\sim)_s = \{[a]_\sim : a \in A_s\}$ and $f_{\mathfrak{A}/\sim}$ with $f_{\mathfrak{A}/\sim}([a_1], \dots, [a_n]) = [f_{\mathfrak{A}}(a_1, \dots, a_n)]$

well defined, i.e. $\mathfrak{A}/\sim$ is sig-Algebra. Abbreviated $\mathfrak{A}_\sim$

$\varphi : \mathfrak{A} \to \mathfrak{A}_\sim$ with $\varphi_s(a) = [a]_\sim$ is a surjective homomorphism, the canonical homomorphism.

Connections between $\models, =_E, \vdash_E$

c) $\mathfrak{A}, \mathfrak{A}'$ sig-algebras $\varphi : \mathfrak{A} \to \mathfrak{A}'$ surjective homomorphism.
Then

$$\mathfrak{A} \models s = t \rightsquigarrow \mathfrak{A}' \models s = t$$

d) spec $= (\text{sig}, E)$:

$$s =_E t \ \text{ iff } \ E \vdash s = t$$

e) $\mathfrak{A}$ sig-Algebra, R a sort compatible bin. relation over $\mathfrak{A}$.
Then there is a smallest congruence $\equiv_R$ over $\mathfrak{A}$ that contains R, i.e. $R \subseteq \equiv_R$

$\equiv_R$ the congruence generated by R

Proofs: Don't give up...

Connections between $\models, =_E, \vdash_E$

f) $\mathfrak{A}$ sig-Algebra, E equational system over (sig, V).
E induces a relation $\underset{E,\mathfrak{A}}{\sim}$ on $\mathfrak{A}$ where
$a \underset{E,\mathfrak{A},s}{\sim} a'$ $(a, a' \in A_s)$ iff there is $t = t' \in E$ and an assignment $\varphi : V \rightarrow \mathfrak{A}$ with $\varphi(t) = a$, $\varphi(t') = a'$
This relation is sort compatible.
Fact: Let $\equiv$ be a congruence over $\mathfrak{A}$ that contains $\underset{E,\mathfrak{A}}{\sim}$, then $\mathfrak{A}/\equiv$ is a spec $= (\text{sig}, E)$-Algebra, i.e. model of E.

g) Existence: $\mathfrak{A} = T_{\text{sig}}$ the (ground) term algebra, then $=_E$ is on T_{sig} the smallest congruence that contains $\underset{E,\mathfrak{A}}{\sim}$.
In particular $T_{\text{sig}}/=_E$ is a term-generated model of E.

example

spec :: INT with $\text{pred}(\text{suc}(x)) = x$, $\text{suc}(\text{pred}(x)) = x$

$$
\begin{aligned}
(T_{\text{INT}}/=_E)_{\text{int}} = \quad & \{[0] = \{0, \text{pred}(\text{suc}(0)), \text{suc}(\text{pred}(0)), \ldots \\
& [\text{suc}(0)] = \{\text{suc}(0), \text{pred}(\text{suc}(\text{suc}(0))), \ldots \\
& [\text{suc}(\text{suc}(0))] = \{\cdots \\
& [\text{pred}(0)] = \{\text{pred}(0), \text{suc}(\text{pred}(\text{pred}(0))) \ldots
\end{aligned}
$$

$$
\begin{aligned}
\text{suc}_{T_{\text{INT}}/=_E} \quad & ([\text{pred}(\text{suc}(0))]) = [\text{suc}(\text{pred}(\text{suc}(0)))] \\
& = [\text{suc}(0)] \\
& = \text{suc}_{T_{\text{INT}}/=_E}([0])
\end{aligned}
$$

Birkhoff's Theorem

Theorem 6.17 (Birkhoff). *For each specification spec* $= (sig, E)$ *the following holds*

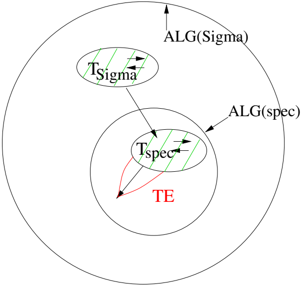
$$E \models s = t \quad iff \quad E \vdash s = t \quad (\text{i. e. } s =_E t)$$

Definition 6.18. *Initial semantics*
Let $spec = (sig, E)$, *sig strict.*
The algebra $T_{sig}/=_E$ *(Quotient term algebra)*
($=_E$ *the smallest congruence relation on* T_{sig} *generated by* E*)*
is defined as initial algebra semantics of $spec = (sig, E)$.

It is term-generated and initial in $ALG(spec)$*!*

Initial Algebra semantics

Initial Algebra semantics assigns to each equational specification spec the isomorphism class of the (initial) quotient term algebra $T_{\text{sig}}/=_E$.
Write: T_{spec} or $I(E)$



$$\text{sig} = \Sigma, \ \text{spec} = (\Sigma, E)$$

Quotient term algebras

Quotient term algebras are ADT.

Example 7.1. *(Continuation)* $spec = INT$

A^i_{int}	$\mathbb{Z}$	$\{true, false\}$	$\{1\}^+ \cup \{0\}^+ \cup \{z\}$
0_{A^i}	0	$true$	z
suc_{A^i}	$\text{suc}_{\mathbb{Z}}$	not	$\dots$
pred_{A^i}	$\text{pred}_{\mathbb{Z}}$	not	$\dots$

$T_{\text{INT}}/=_E$ $\quad [0] \mapsto true \quad [\text{suc}^{2n}(0)] \mapsto true$

$[\text{suc}^{2n+1}(0)] \mapsto false \quad [\text{pred}^{2n+1}(0)] \mapsto false$

$[\text{pred}^{2n}(0)] \mapsto true$

Initial algebra

spec $= (\text{sig}, E)$ Initial algebra T_{spec} $(I(E))$

Questions:

- Is T_{spec} computable?
- Is the word problem $(T_{\text{sig}}, =_E)$ solvable?
- Is there an "operationalization" of T_{spec}?
- Which (PL1-) properties are valid in T_{spec} ?
- How can we prove these properties? Are there general methods?

Equational theory / Inductive (equational-) theory

Definition 7.2. *Properties of equations*

a) $TH(E) = \{s = t : E \models s = t\}$ *Equational theory*
Equations that are valid in all spec-algebras.

b) $ITH(E) = \{s = t : T_{spec} \models s = t\}$ *inductive (=)-theory*
Equations that are valid in all term generated spec-algebras.

Equational theory / Inductive (equational-) theory

Consequence 7.3. *Basic properties*

a) $TH(E) \subseteq ITH(E)$, *since* T_{spec} *is a model of* E.

b) *Generally* $TH(E) \subsetneq ITH(E)$ / $=$ Hence *E is ω-complete*

$\rightsquigarrow$ *proofs by consistency inductionless induction*

E recursively enumerable (r.e.), so $TH(E)$ r.e., but $ITH(E)$ generally not r.e.

c) $T_{spec} \models s = t$ *iff* $\sigma(s) =_E \sigma(t)$ *for each ground substitution of the Var. in* s, t. $\rightsquigarrow$ *inductive proof methods, coverset induction*

d) $E : x + 0 = x \qquad x + s(y) = s(x + y)$

$\rightsquigarrow x + y = y + x \in ITH(E) - TH(E)$

$(x + y) + z = x + (y + z)$ *Proof !*

Examples

Example 7.4. *Basic examples*

a) <u>*spec*</u> BOOL
<u>sorts</u> bool
<u>ops</u> $true, false : \to \text{bool}$
$not : \text{bool} \to \text{bool}$
$and, or, \text{impl}, \text{eqv} : \text{bool}, \text{bool} \to \text{bool}$
$\text{if}\_\text{then}\_\text{else}\_ : \text{bool}, \text{bool}, \text{bool} \to \text{bool}$

Example (Cont.)

$$
\begin{aligned}
\underline{\text{eqns}} \quad & \text{not}(\text{true}) = \text{false} \\
& \text{not}(\text{false}) = \text{true} \\
& \text{and}(\text{true}, b) = b \\
& \text{and}(\text{false}, b) = \text{false} \\
& \text{or}(b, b') = \text{not}(\text{and}(\text{not}(b), \text{not}(b'))) \\
& \text{impl}(b, b') = \text{or}(\text{not}(b), b') \\
& \text{eqv}(b, b') = \text{and}(\text{impl}(b, b'), \text{impl}(b', b)) \\
& \text{if true } b' \text{ else } b'' = b' \\
& \text{if false } b' \text{ else } b'' = b''
\end{aligned}
$$

$(T_{\text{BOOL}})_{\text{bool}} = \{[\text{true}], [\text{false}]\}$ (Proof!)

$\rightsquigarrow$ Defined- and constructor-functions.

Example (Cont.)

b) <u>spec</u> SET-OF-CHARACTERS

<u>sorts</u> char, set

<u>ops</u> $a, b, c, \cdots :\longrightarrow \text{char}$

$\varnothing :\longrightarrow \text{set}$

$\text{insert} : \text{char}, \text{set} \rightarrow \text{set}$

<u>eqns</u> $\text{insert}(x, \text{insert}(x, s)) = \text{insert}(x, s)$

$\text{insert}(x, \text{insert}(y, s)) = \text{insert}(y, \text{insert}(x, s))$

$(T_{\text{soc}})_{\text{char}} = \{a, b, c, \ldots\}$

$(T_{\text{soc}})_{\text{set}} = \{[\varnothing], [\text{insert}(a, \varnothing)], \ldots$

“ “

$\{\varnothing\}\{\text{insert}(a, \text{insert}(a, \ldots, \text{insert}(a, \varnothing)\}$

Example (Cont.)

c)

<u>spec</u> NAT
<u>sorts</u> nat
<u>ops</u> $0 :\rightarrow \text{nat}$
$\text{suc} : \text{nat} \rightarrow \text{nat}$
$\_+\_, \_*\_ : \text{nat}, \text{nat} \rightarrow \text{nat}$
<u>eqns</u> $x + 0 = x$
$x + \text{suc}\, y = \text{suc}(x + y)$
$x * 0 = 0$
$x * \text{suc}(y) = (x * y) + x$

$$(T_{\text{NAT}})_{\text{nat}} = \{ \; [0, 0+0, 0*0, \ldots \\ [\text{suc}\,0, 0+\text{suc}\,0, \ldots \\ [\text{suc}(\text{suc}(0)), \ldots$$

Example (Cont.)

d) Binary tree

<u>spec</u> BIN-TREE
<u>sorts</u> nat, tree
<u>ops</u> $0 :\to nat$
$suc : nat \to nat$
$max : nat, nat \to nat$
$leaf :\to tree$
$left : tree \to tree$
$right : tree \to tree$
$both : tree, tree \to tree$
$height : tree \to nat$
$dleft : tree \to tree$
$dright : tree \to tree$

example

Continuation of d) binary tree.

eqns
$$\max(0, n) = n$$
$$\max(n, 0) = n$$
$$\max(\text{suc}(m), \text{suc}(n)) = \text{suc}(\max(m, n))$$
$$\text{height}(\text{leaf}) = 0$$
$$\text{height}(\text{both}(t, t')) = \text{suc}(\max(\text{height}(t), \text{height}(t')))$$
$$\text{height}(\text{left}(t)) = \text{suc}(\text{height}(t))$$
$$\text{height}(\text{right}(t)) = \text{suc}(\text{height}(t))$$

Correctness

Definition 7.5. *A specification spec* $= (sig, E)$ *is sig-correct for a sig-Algebra* $\mathfrak{A}$ *iff* $T_{spec} \cong \mathfrak{A}$ *(i.e. the unique homomorphism is a bijection).*

Example 7.6. *Application:*
INT correct for $\mathbb{Z}$, *BOOL correct for* $\mathbb{B}$

Note: The concept is restricted to initial semantics!

Restrictions/Forgetful functors

Definition 7.7. *Restrictions/Forget-images*

a) $sig = (S, F, \tau)$, $sig' = (S', F', \tau')$ signatures with $sig \subseteq sig'$, i.e. $(S \subseteq S', F \subseteq F', \tau \subseteq \tau')$.

For each sig'*-algebra* $\mathfrak{A}$ *let the sig-part* $\mathfrak{A}|_{sig}$ *of* $\mathfrak{A}$ *be the sig-Algebra with*

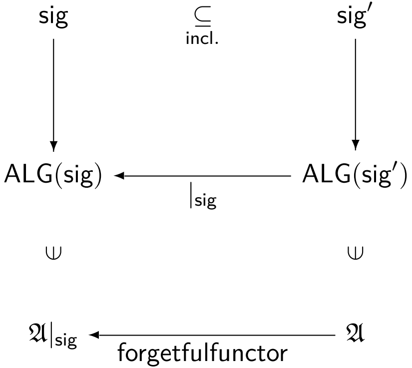
i) $(\mathfrak{A}|_{sig})_s = A_s$ *for* $s \in S$
ii) $f_{\mathfrak{A}|_{sig}} = f_{\mathfrak{A}}$ *for* $f \in F$

Note: $\mathfrak{A}|_{\text{sig}}$ is sig - algebra. The restriction of $\mathfrak{A}$ to the signature sig.

$\mathfrak{A}|_{\text{sig}}$ is also called forget-image of $\mathfrak{A}$ (with respect to sig).

Restrictions/Forgetful functors

$\mathfrak{A}|_{\mathrm{sig}}$ forget-image of $\mathfrak{A}$ (w.r.t. to sig). The forget image induces consequently a mapping (functor) between classes of algebras in the following way:



Restrictions/Forgetful functor

b) A specification $\text{spec} = (\text{sig}', E)$ with $\text{sig} \subseteq \text{sig}'$ is correct for a sig-algebra $\mathfrak{A}$ iff

$$(T_{\text{spec}})|_{\text{sig}} \cong \mathfrak{A}$$

c) A specification $\text{spec}' = (\text{sig}', E')$ implements a specification $\text{spec} = (\text{sig}, E)$ iff

$$\text{sig} \subseteq \text{sig}' \text{ and } (T_{\text{spec}'})|_{\text{sig}} \cong T_{\text{spec}}$$

Note:

- A consistency-concept is not necessary for $=$-specification. ((initial) models always exist !).
- The general implementation concept ($CI(\text{spec}) \subseteq CI(\text{spec}')$) reduces here to $=$ of the valid equations in the smaller language. „complete“ theories.

Problems

Verification of $s = t \in Th(E)$ or $\in ITH(E)$.

For $Th(E)$ find $=_E$ an equivalent, convergent term rewriting system (see group example).

For $ITH(E)$ induction's methods:

s, t induce functions to T_{spec}. If $x_1, \ldots, x_n$ are the variables in s and t, types $s_1, \ldots, s_n$.
$s : (T_{\text{spec}})_{s_1} \times \cdots \times (T_{\text{spec}})_{s_n} \to (T_{\text{spec}})_s$
$s = t \in ITh(E)$ iff s and t induce the same functions $\rightsquigarrow$ prove this by induction on the construction of the ground terms.
NAT $\quad 0, \text{suc}, + \quad x + y = y + x \quad \in ITH$
$\qquad\qquad\qquad\quad 0 + x = x$

Problems

- $0 + 0 = 0 \qquad \text{Ass.}: 0 + a = a$
  $0 + Sa =_E S(0 + a) =_I S(a)$
- $x + 0 = 0 + x \qquad \text{Ass.}: x + a = a + x$
  $x + Sa =_E S(x + a) =_I S(a + x) =_E a + Sx \stackrel{?}{=} Sa + x$
- $x + Sy = Sx + y$
  $x + S0 =_E S(x + 0) =_E Sx =_E Sx + 0$
  $x + SSa =_E S(x + Sa) =_I S(Sx + a) =_E Sx + Sa$

spec(sig, E)
Equations only often do not suffice

$P_{\text{spec}}(\text{sig}, E, \mathit{Prop})$
Properties that should hold!
$\rightsquigarrow$ Verification tasks

Structuring mechanisms

Horizontal:	- Decomposition, - Combination,
	- Extension, - Instantiation
Vertical:	- Realisation, - Information hiding,
	- Vertical composition

Here:

Combination, Enrichment, Extension, Modularisation, Parametrisation

$\rightsquigarrow$ **Reusability.**

Structuring mechanisms

BIN-TREE

1) spec NAT
   sorts nat
   ops $0 :\rightarrow nat$
   $suc : nat \rightarrow nat$

2) spec NAT1
   use NAT
   ops $max : nat, nat \rightarrow nat$
   eqns $max(0, n) = n$
   $max(n, 0) = n$
   $max(s(m), s(n)) = s(max(m, n))$

Structuring mechanisms

BIN-TREE (Cont.)

3) spec BINTREE1
   sorts bintree
   ops leaf :$\longrightarrow$ bintree

   left, right : bintree $\longrightarrow$ bintree
   both : bintree, bintree $\longrightarrow$ bintree

4) spec BINTREE2
   use NAT1, BINTREE1
   ops height : bintree $\longrightarrow$ nat
   eqns $\vdots$

Combination

Definition 7.8 (Combination). *Let* $spec_1 = (sig_1, E_1)$*, with* $sig_1 = (S_1, F_1, \tau_1)$ *be a signature and* $sig_2 = [S_2, F_2, \tau_2]$ *a triple,* E_2 *set of equations.*

$\mathrm{comb} = spec_1 + (sig_2, E_2)$ *is called combination*
iff
$spec = ((S_1 \cup S_2), (F_1 \cup F_2), (\tau_1 \cup \tau_2)), E_1 \cup E_2)$ *is a specification.*

In particular $((S_1 \cup S_2), (F_1 \cup F_2), (\tau_1 \cup \tau_2))$ *is a signature and* E_2 *contains „syntactically correct“ equations.*

The semantics of comb: $T_{\mathrm{comb}} := T_{spec}$

The semantics of comb

$T_{\text{comb}} := T_{\text{spec}}$

Typical cases:

$S_2 = \varnothing$, F_2 new function symbols with arities τ_2 (in old sorts).

S_2 new sorts, F_2 new function symbols.
τ_2 arities in new + old sorts.

E_2 only „new“ equations.

Notations: use, include (protected)

Example

Example 7.9. a) *Step-by-step design of integer numbers*

		semantics
spec	INT1	
sorts	int	$T_{\text{INT1}} \cong (\mathbb{N}, 0, \text{suc}_{\mathbb{N}})$
ops	$0 :\longrightarrow$ int	
	suc : int $\longrightarrow$ int	
	$\cap$	$\cap$
spec	INT2	
use	INT1	$T_{\text{INT2}} \cong (\mathbb{Z}, 0, \text{suc}_{\mathbb{Z}}, \text{pred}_{\mathbb{Z}})$
ops	pred : int $\longrightarrow$ int	
eqns	$\text{pred}(\text{suc}(x)) = x$	
	$\text{suc}(\text{pred}(x)) = x$	

Example (Cont.)

Question: Is the INT1-part of T_{INT2} equal to T_{INT1}??
Does INT2 implement INT1?

$$(T_{\mathrm{INT2}})|_{\mathrm{INT1}} \cong T_{\mathrm{INT1}}$$

$$(\mathbb{Z}, 0, \mathrm{suc}_{\mathbb{Z}}, \mathrm{pred}_{\mathbb{Z}})|_{\mathrm{INT1}}$$

$$\shortparallel$$

$$(\mathbb{Z}, 0, \mathrm{suc}_{\mathbb{Z}}) \quad \ncong \quad (\mathbb{N}, 0, \mathrm{suc}_{\mathbb{N}})$$

Caution: Not always the proper data is specified!
Here new data objects of sort int were introduced.

Example (Cont.)

b) spec NAT2
use NAT
eqns $\text{suc}(\text{suc}(x)) = x$

$$(T_{\text{NAT2}})|_{\text{NAT}} = (\mathbb{N} \bmod 2)|_{\text{NAT}} = \mathbb{N} \bmod 2 \quad \not\cong \quad \mathbb{N} = T_{\text{NAT}}$$

Problem: Adding new or identifying old elements.

Problems with the combination

Let

$$\text{comb} = \text{spec}_1 + (\text{sig}, E)$$

$$\left.\begin{array}{r} (T_{\text{comb}})|_{\text{spec}_1} \text{ is spec}_1 \text{ Algebra} \\ T_{\text{spec}_1} \text{ is initial spec}_1 \text{ algebra} \end{array}\right\} \leadsto$$

$\exists!$ homomorphism $h : T_{\text{spec}_1} \to (T_{\text{comb}})|_{\text{spec}_1}$

Properties of

h: not injective / not surjective / bijective.

e.g. $(T_{\text{BINTREE2}})|_{\text{NAT}} \cong T_{\text{NAT}}$.

Extension and enrichment

Definition 7.10. a) *A combination* comb $= spec_1 + (sig, E)$ *is an extension iff*

$$(T_{\text{comb}})|_{spec_1} \cong T_{spec_1}$$

b) *An extension is called enrichment when sig does not include new sorts, i.e.* $sig = [\varnothing, F_2, \tau_2]$

- *Find sufficient conditions (syntactical or semantical) that guarantee that a combination is an extension*

Parameterisation

Definition 7.11 (Parameterised Specifications). *A parameterised specification Parameter=(Formal, Body) consist of two specifications: formal and body with formal $\subseteq$ body.*

i.e. $Formal{=}(sig_F, E_F)$, $Body{=}(sig_B, E_B)$, *where*
$sig_F \subseteq sig_B \qquad E_F \subseteq E_B$.

Notation: Body[Formal]

Syntactically: Body $=$ Formal $+(sig', E')$ *is a combination.*

Note: In general it is not required that Formal or Body[Formal] have an initial semantics.
It is not necessary that there exist ground terms for all the sorts in Formal.
Only until a concrete specification is "substituted", this requirement will be fulfilled.

Example

Example 7.12.

spec	ELEM	$(T_{spec})_{elem} = \emptyset$
sorts	elem	
ops	next : elem $\rightarrow$ elem	

spec	STRING[ELEM]	$(T_{spec})_{string} = \{[empty]\}$
use	ELEM	
sorts	string	
ops	empty :$\rightarrow$ string	
	unit : elem $\rightarrow$ string	
	concat : string, string $\rightarrow$ string	
	ladd : elem, string $\rightarrow$ string	
	radd : string, elem $\rightarrow$ string	

Example (Cont.)

$$
\begin{aligned}
\text{eqns}\quad & \text{concat}(s, \text{empty}) = s \\
& \text{concat}(\text{empty}, s) = s \\
& \text{concat}(\text{concat}(s_1, s_2), s_3) = \text{concat}(s_1, \text{concat}(s_2, s_3)) \\
& \text{ladd}(e, s) = \text{concat}(\text{unit}(e), s) \\
& \text{radd}(s, e) = \text{concat}(s, \text{unit}(e))
\end{aligned}
$$

Parameter passing: ELEM $\rightarrow$ NAT

$$\text{STRING[ELEM]} \rightarrow \text{STRING[NAT]}$$

Assignment: formal parameter $\rightarrow$ current parameter

$$
\begin{aligned}
S_F &\rightarrow S_A \\
Op &\rightarrow Op_A
\end{aligned}
$$

Mapping of the sorts and functions, semantics?

Signature morphisms - Parameter passing

Definition 7.13. a) *Let $sig_i = (S_i, F_i, \tau_i)$ $i = 1, 2$ be signatures. A pair of functions $\sigma = (g, h)$ with $g : S_1 \to S_2, h : F_1 \to F_2$ is a signature morphism, in case that for every $f \in F_1$*

$$\tau_2(hf) = g(\tau_1 f)$$

(g extended to $g : S_1^* \to S_2^*$).

In the example g :: elem $\to$ nat h :: next $\to$ suc
Also $\sigma : sig_{\text{BOOL}} \to sig_{\text{NAT}}$ with

$$\begin{array}{llll} g :: & bool \to \text{nat} & & \\ h :: & true \to 0 & \text{not} \to \text{suc} & and \to \text{plus} \\ & false \to 0 & & or \to \text{times} \end{array}$$

is a signature morphism.

Signature morphisms - Parameter passing

b) spec = Body[Formal] parameterised specification and *Actual* a standard specification (i.e. with an initial semantics).
A parameter passing is a signature morphism $\sigma : \text{sig(Formal)} \rightarrow \text{sig(Actual)}$ in which Actual is called the current parameter specification.

(Actual, σ) defines a specification VALUE through the following syntactical changes to Body:

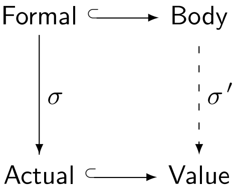
1) Replace Formal with Actual: Body[Actual].
2) Replace in the arities of $op : s_1 \ldots s_n \rightarrow s_0 \in$ Body, which are not in Formal, $s_i \in$ Formal with $\sigma(s_i)$.
3) Replace in each not-formal equation $L = R$ of Body each $o_P \in$ Formal with $\sigma(o_P)$.
4) Interprete each variable of a type s with $s \in$ Formal as variable of type $\sigma(s)$.
5) Avoid name conflicts between actual and Body/Formal by renaming properly.

Parameter passing

Notation:

$$\text{Value} = \text{Body}[\text{Actual}, \sigma]$$

Consequently for $\sigma : \text{sig}(\text{Formal}) \rightarrow \text{sig}(\text{Actual})$ we get a a signature morphism
$\sigma' : \text{sig}(\text{Body}[\text{Formal}]) \rightarrow \text{sig}(\text{Body}[\text{Actual}, \sigma]$ with



$$\sigma'(x) = \begin{cases} \sigma(x) & x \in \text{Formal} \\ x' & x \notin \text{Formal} \end{cases}$$

Where x' is a renaming, if there are naming conflicts.

Signature morphisms (Cont.)

Definition 7.14. *Let $\sigma : sig' \rightarrow sig$ be a signature morphism.*

Then for each sig-Algebra $\mathfrak{A}$ define $\mathfrak{A}|_\sigma$ a sig'-Algebra, in which for $sig' = (S', F', \tau')$

$(\mathfrak{A}|_\sigma)_s = A_{\sigma(s)}$ $s \in S'$ and $f_{\mathfrak{A}|_\sigma} = \sigma(f)_{\mathfrak{A}}$ $f \in F'$.

$\mathfrak{A}|_\sigma$ *is called forget-image of $\mathfrak{A}$ along σ*

Hence $|_\sigma$ is a "mapping" from sig-Algebras into sig'-Algebras.

(Special case: $sig' \subseteq sig :\hookrightarrow$) $|_{sig'}$

Example

Example 7.15. $\mathfrak{A} = T_{\mathrm{NAT}}$ *(with* 0, suc, plus, times*)*
$sig' = sig(\mathrm{BOOL}) \quad sig = sig(\mathrm{NAT})$
$\sigma : sig' \rightarrow sig$ *the one considered previously.*

$$
\begin{aligned}
((T_{\mathrm{NAT}})|_\sigma)_{\mathrm{bool}} &= (T_{\mathrm{NAT}})_{\sigma(\mathrm{bool})} = (T_{\mathrm{NAT}})_{\mathrm{nat}} \\
&= \{[0], [\mathrm{suc}(0)], \ldots\}
\end{aligned}
$$

$$
\begin{array}{lclcl}
true_{(T_{\mathrm{NAT}})|_\sigma} & = & \sigma(true)_{T_{\mathrm{NAT}}} & = & [0] \\
false_{(T_{\mathrm{NAT}})|_\sigma} & = & \sigma(false)_{T_{\mathrm{NAT}}} & = & [0] \\
not_{(T_{\mathrm{NAT}})|_\sigma} & = & \sigma(not)_{T_{\mathrm{NAT}}} & = & \mathrm{suc}_{T_{\mathrm{NAT}}} \\
and_{(T_{\mathrm{NAT}})|_\sigma} & = & \sigma(and)_{T_{\mathrm{NAT}}} & = & \mathrm{plus}_{T_{\mathrm{NAT}}} \\
or_{(T_{\mathrm{NAT}})|_\sigma} & = & \sigma(or)_{T_{\mathrm{NAT}}} & = & \mathrm{times}_{T_{\mathrm{NAT}}}
\end{array}
$$

Forget images of homomorphisms

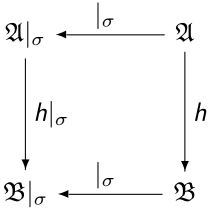
Definition 7.16. *Let $\sigma : sig' \to sig$ a signature morphism, $\mathfrak{A}, \mathfrak{B}$ sig-algebras and $h : \mathfrak{A} \to \mathfrak{B}$ a sig-homomorphism, then*

$h|_\sigma := \{h_{\sigma(s)} \mid s \in S'\}$ *(with $sig' = (S', F', \tau')$) is a sig'-homomorphism from $\mathfrak{A}|_\sigma \to \mathfrak{B}|_\sigma$ by setting*

$$\begin{array}{lccc} (h|_\sigma)_s = h_{\sigma(s)} : & A_{\sigma(s)} & \to & B_{\sigma(s)} \\ & \| & & \| \\ & (\mathfrak{A}|_\sigma)_s & \to & (\mathfrak{B}|_\sigma)_s \end{array}$$

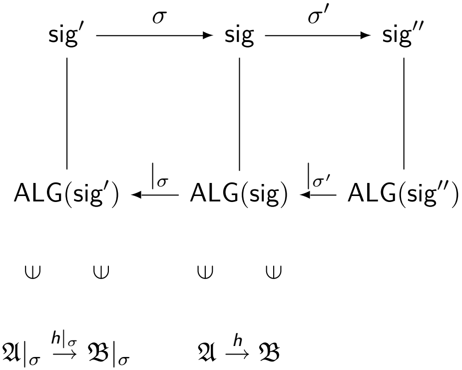
$h|_\sigma$ is called the forget image of h along σ

Forgetful functors



Forgetful functors

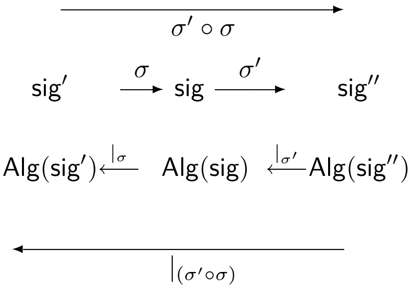
Properties of $h|_\sigma$ (forget image of h along σ)



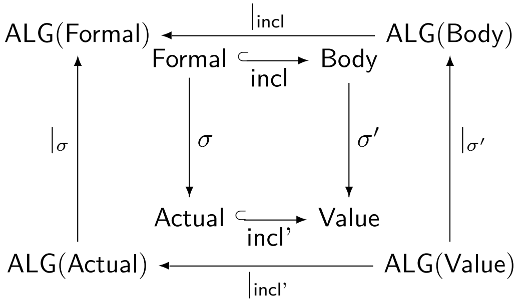
Compatible with identity, composition and homomorphisms.

Forgetful functors

Let $\sigma : \text{sig}' \to \text{sig}$, $\mathfrak{A}, \mathfrak{B}$, sig-algebras, $h : \mathfrak{A} \to \mathfrak{B}$, sig-homomorphism.
$h|_\sigma = \{h_{\sigma(s)} \mid s \in S'\}$, $\text{sig}' = (S', F', \tau')$, with
$h|_\sigma : A|_\sigma \to B|_\sigma$ forget image of h along σ.



Parameter Specification *Body*[*Formal*]



Semantics of parameter passing (only signature)

Definition 7.17. *Let Body[Formal] be a parameterized specification.* σ : Formal $\rightarrow$ Actual *signature morphism.*

Semantics of the the "instantiation" i.e. parameter passing [Actual, σ].

$$\sigma : \text{Formal} \rightarrow \text{Actual}$$
$$\downarrow$$
$$\text{initial semantics of value. i. e.}$$
$$T_{\text{Body[Actual},\sigma]}$$

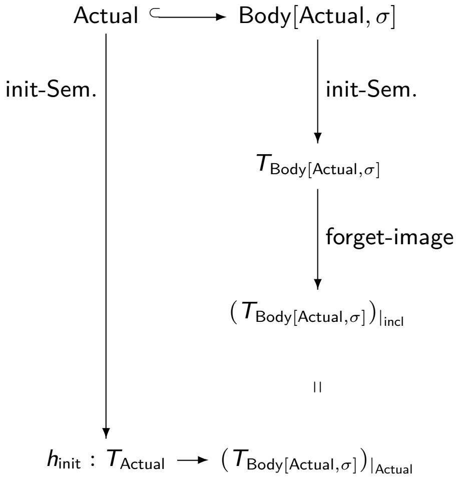
Can be seen as a mapping : $S :: (T_{\text{Actual}}, \sigma) \mapsto T_{\text{Body[Actual},\sigma]}$

This mapping between initial algebras can be interpreted as correspondence between formal algebras $\rightarrow$ *body-algebras.*

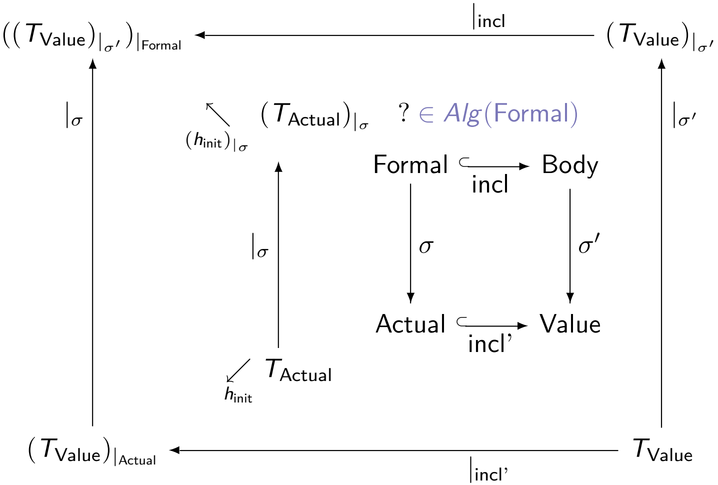
$$(T_{\text{Actual}})|_{\sigma} \mapsto (T_{\text{Body[Actual},\sigma]})|_{\sigma'}$$

Semantics parameter passing

$$(T_{\text{Actual}})|_{\sigma} \mapsto (T_{\text{Body}[\text{Actual},\sigma]})|_{\sigma'}$$



Mapping between initial algebras



Properties of the signature morphism

Formal	$\xrightarrow{\sigma}$	Actual
sorts elem	elem $\rightarrow$ nat	sorts nat
ops $a, b :\rightarrow$ elem	$a \rightarrow 0$	ops $0, 1 :\rightarrow$ nat
eqns $a = b$	$b \rightarrow 1$	eqns

$\mathfrak{A} = T_{\text{Actual}} \quad A_{\text{nat}} = \{0, 1\}$

$\mathfrak{A}|_\sigma \in \text{Alg}(\text{sig Formal}) \ (A|_\sigma)_{\text{elem}} = \{0, 1\}$

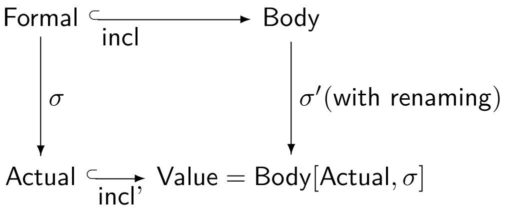
$a|_{\mathfrak{A}|_\sigma} = 0 \neq 1 = b|_{\mathfrak{A}|_\sigma}$

Equation from Formal is not fulfilled! i.e. $\mathfrak{A}|_\sigma \notin \text{Alg}(\text{Formal})$.

Parameter passing (Actual, σ)

Body[Formal]

$\sigma : \text{sig}(\text{Formal}) \rightarrow \text{sig}(\text{Actual})$
signature morphism



Precondition: sig(Actual) and sig(Value) strict.

Parameter passing (Actual, σ)

Forgetful functor: $|_\sigma : \text{Alg}(\text{sig}) \to \text{Alg}(\text{sig}')$

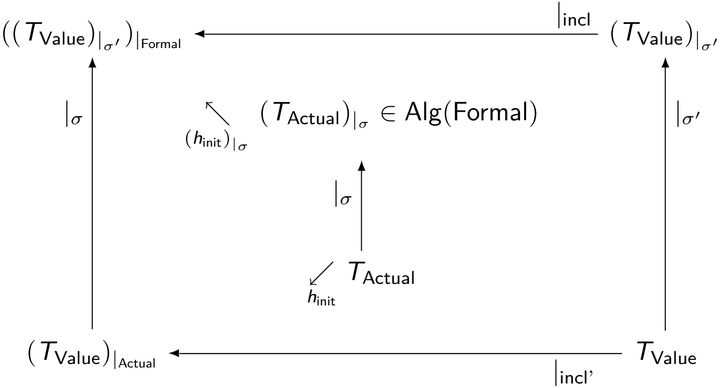
$$\mathfrak{A}|_\sigma \text{ for } \sigma : \text{sig}' \to \text{sig}$$

$h : \mathfrak{A} \to \mathfrak{B}$ sig-homomorphism

$$h|_\sigma : \mathfrak{A}|_\sigma \to \mathfrak{B}|_\sigma$$

sig′-homomorphism

Parameter passing (Actual, σ)



Problems: 1) $(T_{\text{Actual}})|_{\sigma} \notin \text{Alg}(\text{Formal})$, 2) h_{init} is not a bijection.

Specification morphisms

Definition 7.18. *Let $spec' = (sig', E'), spec = (sig, E)$ (general) specifications.*
A signature morphism $\sigma : sig' \to sig$ is called a specification morphism, if $\sigma(s) = \sigma(t) \in Th(E)$ for every $s = t \in E'$ holds.

Write: $\sigma : spec' \to spec$

Fact: If $\mathfrak{A} \in \mathrm{Alg}(spec)$ *then* $\mathfrak{A}|_\sigma \in \mathrm{Alg}(spec')$
i.e. $|_\sigma : \mathrm{Alg}(spec) \to \mathrm{Alg}(spec')$ *!*

Often „only"the weaker condition $\sigma(s) = \sigma(t) \in ITh(E)$ is demanded in above definition. More spec morphisms!

Semantically correct parameter passing

Definition 7.19. *A parameter passing for Body[Formal] is a pair* (Actual, σ)*: Actual an equational specification and* $\sigma : \text{Formal} \to \text{Actual}$ *a specification morphism.*

Hence:: $(T_{\text{Actual}})|_{\sigma} \in \text{Alg}(\text{Formal})$

- Demand also h_{init} bijection. Proof tasks become easier.

There are syntactical restrictions that guarantee this.

Algebraic Specification languages

CLEAR, Act-one, -Cip-C, Affirm, ASL, Aspik, OBJ, ASF, $\underset{+}{\leadsto}$ newer languages: - Spectrum, - Troll.

Example

Example 7.20.

$$\text{Formal} :: \begin{cases} \textit{spec} & \text{ELEMENT} \\ \text{use} & \text{BOOL} \\ \text{sorts} & \text{elem} \\ \text{ops} & . \leq . : \text{elem}, \text{elem} \to \text{bool} \\ \text{eqns} & x \leq x = \textit{true} \\ & \text{imp}(x \leq y \textit{ and } y \leq z, x \leq z) = \textit{true} \\ & x \leq y \textit{ or } y \leq x = \textit{true} \end{cases}$$

Example (Cont.)

spec	LIST[ELEMENT]
use	ELEMENT
sorts	list
ops	nil $:\rightarrow$ list
	$.$: elem, list $\rightarrow$ list
	insert : elem, list $\rightarrow$ list
	insertsort : list $\rightarrow$ list
	case : bool, list, list $\rightarrow$ list
	sorted : list $\rightarrow$ bool

Example (Cont.)

eqns $\text{case}(\text{true}, l_1, l_2) = l_1$
$\text{case}(\text{false}, l_1, l_2) = l_2$

$\text{insert}(x, \text{nil}) = x.\text{nil}$
$\text{insert}(x, y.l) = \text{case}(x \leq y, x.y.l, y.\,\text{insert}(x, l))$

$\text{insertsort}(\text{nil}) = \text{nil}$
$\text{insertsort}(x.l) = \mathit{insert}(x, \mathit{insertsort}(l))$

$\text{sorted}(\text{nil}) = \text{true}$
$\text{sorted}(x.\text{nil}) = \text{true}$
$\text{sorted}(x.y.l) = \text{if } x \leq y \text{ then sorted}(y.l) \text{ else false}$

Property: $\text{sorted}(\text{insertsort}(l)) = \text{true}$

Example (Cont.)

ACTUAL $\equiv$ BOOL

$$\begin{aligned} \sigma: \quad & \text{elem} \rightarrow \text{bool}, \text{bool} \rightarrow \text{bool} \\ & . \leq . \rightarrow \text{impl} \end{aligned}$$

The equations of ELEMENT are in $Th(\text{BOOL})$

$\rightsquigarrow$ Specification morphism

Example (Cont.)

$$
\begin{array}{lll}
\text{ACTUAL} \equiv \text{NAT} & & \\
\sigma: & \text{bool} \longrightarrow \text{nat} & \text{elem} \longrightarrow \text{nat} \\
 & \text{true} \longrightarrow \text{suc}(0) & \text{not allowed} \\
 & \text{false} \longrightarrow 0 & \\
 & \text{not} \longrightarrow \text{suc} & \\
 & \text{or} \longrightarrow \text{plus} & \\
 & \text{and} \longrightarrow \text{times} & \\
 & \vdots & \\
 & . \leq . \longrightarrow \cdots &
\end{array}
$$

is not a specification morphism

$\text{not}(\text{false}) = \text{true}$

$\text{not}(\text{true}) = \text{false}$ does not hold!.