4. Equational Partial Algebras

A heterogeneous partial **algebra** consists of several carrier sets and several operations between these sets: $A = ((A_s\colon s \in S), (\omega_A\colon \omega \in \Omega))$. Here $\Sigma = (S, \alpha\colon \Omega \to S^* \times S, (def(\omega)\colon \omega \in \Omega), \mathcal{A})$ is a signature, where S is a set of sort names and Ω a set of operator symbols. Each operator $\omega \in \Omega$ $(\omega\colon s_1\, s_2 \ldots s_n \to s)$ has an input (a sequence of sorts), an output sort and defining conditions $def(\omega)$. Operations may be total (defined for each element of the Cartesian Product of input carrier sets); then the defining conditions consist of an empty set of equations. These total operations have a degree of partiality of zero. If we use only operations of degree zero in the equation system $def(\omega)$, then the corresponding operation ω has partiality degree 1. More general, we require that each operation has a certain natural number as degree of partiality. The **degree of partiality** of ω is n, if in $def(\omega)$ contains only operations of degree smaller than n, but at least one with degree $n\text{-}1$.
The last component of our signature Σ is a set of axioms $\mathcal{A}$. To guarantee that for each Signature $\Sigma = (S, \alpha\colon \Omega \to S^* \times S, (def(\omega)\colon \omega \in \Omega), \mathcal{A})$ an initial algebra exists we allow only implications, where the left hand side and the right hand side are equations.

An algebra $A = ((A_s\colon s \in S), (\omega_A\colon \omega \in \Omega))$ is a **Σ-algebra** $(\Sigma = (S, \alpha\colon \Omega \to S^* \times S, (def(\omega)\colon \omega \in \Omega), \mathcal{A}))$, iff we have for each sort $s \in S$ a carrier set A_s, for each operation symbol $\omega \in \Omega\, (\omega\colon s_1\, s_2 \ldots s_n \to s)$ a partial function $\omega_A\colon A_{s_1} \times A_{s_2} \times \ldots \times A_{s_n} \to A_s$, such that $A_\omega(a_1,a_2,\ldots,a_n)$ is defined if and only if $(a_1,a_2,\ldots,a_n)$ is a solution of the defining condition $def(\omega)$. Further, it is required that each implication of $\mathcal{A}$ holds in the algebra.
A **term $\omega(t_1, t_2, \ldots, t_n)$ is defined** in an algebra A for $(a_1,a_2,\ldots a_m)$, if $t_1, t_2,\ldots$ and t_n are defined for $(a_1,a_2,\ldots, a_m)$, with results $b_1, b_2, \ldots, b_n$, respectively and ω_A is defined for $(b_1, b_2, \ldots, b_n)$. $\omega_A(b_1, b_2, \ldots, b_n)$ is the result of the application of $\omega(t_1, t_2, \ldots, t_n)$ on $(a_1,a_2,\ldots, a_m)$.
$(a_1,a_2,\ldots, a_m)$ is a **solution of an equation** $t_1 = t_2$, if t_1 and t_2 are defined for $(a_1,a_2,\ldots, a_m)$ and the both applications of t_1 and t_2 on $(a_1,a_2,\ldots, a_m)$ result in the same value.
An **implication** <u>*if*</u> *g* <u>*then*</u> *h* **holds** in an algebra A, if each solution of g is also a solution of h.
A **homomorphism** $f\colon A \to B$, from Σ-algebra A to Σ-algebra B is a family of mappings $(f_s\colon A_s \to B_s\colon s \in S)$, which is compatible with the operations of Σ that means for each $\omega \in \Omega$ $(\omega\colon s_1\, s_2 \ldots s_n \to s)$ and each solution $(a_1,a_2,\ldots a_n)$ of the defining condition $def(\omega)$ in A is $(f_{s_1}(a_1), f_{s_2}(a_2),\ldots, f_{s_n}(a_n))$ solution of $def(\omega)$ in B and the following equation holds:
$f_s(\omega_A(a_1,a_2,\ldots a_n)) = \omega_B(f_{s_1}(a_1), f_{s_2}(a_2),\ldots, f_{s_n}(a_n))$.
By the equation is expressed that the following diagram commutes:

$$\begin{tikzcd}
A_{s_1} \times A_{s_2} \times \ldots \times A_{s_n} \arrow[r, "f_{s_1 s_2 \ldots s_n}"] \arrow[d, "A_\omega"] & B_{s_1} \times B_{s_2} \times \ldots \times B_{s_n} \arrow[d, "B_\omega"] \\
A_s \arrow[r, "f_s"] & B_s
\end{tikzcd}$$

A Σ-algebra I is **initial**, if it exists to each other Σ-algebra A exactly one homomorphism. Inital algebras play an important role in specification languages. They have two important properties:

1) no junk (they contain only elements, which can be represented by ground terms (terms without variables))
2) no confusion (two terms are equal only, if it is forced by the axioms)

Let us consider the simplest data type of Boolean values:

BOOL =
<u>sorts</u> Bool
<u>opers</u> true, false $\to$ Bool // 0-ary operations (constants)
not (Bool) $\to$ Bool // unary operation
or, and (Bool, Bool) $\to$ Bool // binary operations
<u>axioms</u> b: Bool
not(true) = false, not(false) = true
or(b, true) = or(true, b) = true, or(false, false) = false
and(b, false) = and(false, b) = false, and(true, true) = true
<u>end</u>

The algebra I with the two truth values $I_{Bool} = \{T, F\}$ and the well known truth-operation is the initial algebra for this signature BOOL, because for every other BOOL-algebra A exactly one homomorphism $h: I \to A$ exists.
Consider for example an algebra A with $A_{Bool} = \{a\}$. Because all operations are total the image of each application on $A_{Bool}{}^0$, A_{Bool}, $A_{Bool}{}^2$ is a. Therefore
$f: I_{Bool} \to A_{Bool}$ with $f(T) = f(F) = a$ is a homomorphism and the only homomorphism from I to A.
Let us consider an algebra B with $B_{Bool} = \{T, F, \perp\}$ and
$B_{true} = T$, $B_{false} = F$,
$B_{not}(T) = F$, $B_{not}(F) = T$, $B_{not}(\perp) = \perp$

or	T	F	⊥
T	T	T	T
F	T	F	⊥
⊥	T	⊥	⊥

and	T	F	⊥
T	T	F	⊥
F	F	F	F
⊥	⊥	F	⊥

Evidently $i: I_{Bool} \to B_{Bool}$ with $i(T) = T$ and $i(F) = F$ is the only homomorphism from I to B.
On the other hand A is not the initial algebra, because no homomorphism exists from A to I for example. To guarantee compatibility with *true* a has to be mapped to T and to guarantee compatibility with *false* a has to be mapped to F.
Further, B is not the initial algebra because there exists no homomorphism from B to I.
Assume g is such a homomorphism $g: B \to I$
Let $g(\perp) = T \in \{T, F\}$
Because of compatibility with *not* holds:
$g(not_B(\perp)) = not_I(g(\perp))$, but the left hand side is equal to T and the right hand side to F. The same inequality results, if we choose $g(\perp) = F$.
In general, it is known that from the term algebra the initial algebra can be constructed. We have to say only in which cases we consider two terms to represent the same element. This is the case, if the equality of the two terms is forced by the given axioms. In the above specification the quotient algebra consists of two classes only:
T = {true, and(true, true), or(true, true), not(false), and(true, not(false)),...}
F = {false, and(false, false), or(false, false), not(true), and(true, false),.....}.
We see we can represent both equivalence classes by the ground terms *true* and *false*.
In the same way by n constants an arbitrary finite carrier set can be specified.
The next interesting data type are the natural numbers.

sorts Nat
opers zero: $\rightarrow$ Nat
succ: Nat $\rightarrow$ Nat
end

Because we have no axioms no ground terms are equalized, such that
{zero, succ(zero), succ(succ(zero)), succ(succ(succ(zero))),...}
is the carrier set of the initial algebra.
If we want to specify the integers, then we need the predecessor operation as an additional generating operation:

INT = BOOL +
sorts Int
opers 0 $\rightarrow$ Int
succ, pred (Int) $\rightarrow$ Int
+, -, * (Int, Int) $\rightarrow$ Int
pos (Int) $\rightarrow$ Bool // positive
equal-i (Int, Int) $\rightarrow$ Bool
div (Int, y: Int iff equal-i(y, 0) = false) $\rightarrow$ Int // degree of partiality 1
axioms x, y: Int
succ(pred(x)) = x
pred(succ(x)) = x

x + 0 = x
x + (succ(y)) = succ(x + y) (*)
x + (pred(y)) = pred(x + y)

x – 0 = x
x – succ(y) = pred(x – y)
x – pred(y) = succ(x – y)

x * 0 = 0
x * (succ(y)) = (x*y) + x
x * pred(y) = x*y -x

pos(0) = false
pos(succ(0)) = true
if pos(x) = true then pos(succ(x)) = true
if pos(x) = false then pos(pred(x)) = false

equal-i(x, x) = true
if pos(x) then equal-i(0, x) = false & equal-i(x, 0) = false
if equal-i(x, y) = false then equal-i(succ(x), succ(y)) = false
if equal-i(x, y) = false then equal-i(pred(x), pred(y)) = false

if pos(x) = pos(y) = pos(y - x) = true then div(x, y) = 0
if equal-i(x, 0) = false then div(x, x) = succ(0)
if pos(x – y) = pos(y) = true then div(x, y) = succ(div(x – y, y))
if pos(0 – x) = pos(0 - y) = true then div(x, y) = div(0 – x, 0 - y)
if pos(x) = pos(0 – y) = true then div(x, y) = 0 – div(x, 0 - y)
if pos(0 – x) = pos(y) = true then div(x, y) = 0 – div(0 – x, y)
end

Because of the first two equations in each ground term in *succ* and *pred* each occurrence of *succ(pred(* and *pred(succ(* can be eliminated. That means that each term in *0*, *succ*, and *pred* can be reduced to either $succ^n(0)$, or $pred^n(0)$ (n>0) or *0*. By these terms our integers can be represented. Now, we have to ensure that each application of one of the remaining operations to the above terms can be reduced by finite applications of the axioms to the above terms. Further, we have to guarantee that by the axioms no two of the above terms $succ^n(0)$, $pred^n(0)$, or *0* are forced to be identified.
If we consider for example the addition, then $pred^n(0) + succ^m(0)$ can be reduced by (*) to $succ(pred^n(0)+succ^{m-1}(0))$. If we apply the rule once more (m-1)-times, then $succ^m(pred^n(0)+0)$ results. By the preceding axiom results $succ^m(pred^n(0))$. By the second axiom this term can be reduced to one of the ground terms described above.
In the same way we can choose all other combinations of ground terms and operations. Especially, we should look that *pos* does not create new truth values. Because of the partiality of *div*, we don't have to reduce terms of type *div(t, 0)*. We are not forced to introduce error-values to make *div* artificially total.
From methodological point of view the above specification can be improved. It is better to introduce at first only the operations, which are necessary to generate all elements of a new sort and add further operation step by step. For the above example this means that we have to introduce the sort *Int* only with the operations *0*, *succ*, and *pred* and with the first two axioms. The remaining operations can then be introduced by so called initial extensions.
If $\Sigma_1 \subseteq \Sigma_2$ (that means each sort, each operator symbol, and each axiom of Σ_1 is contained in Σ_2), then the Σ_2-algebra *I* is called **initial extension** of a Σ_1-algebra *A*, if $I\downarrow\Sigma_1 = A$ (The Σ_1-Redukt $I\downarrow\Sigma_1$ consists of all carrier sets and operations of *I* from Σ_1) and for each Σ_2-algebra *B* and each Σ_1-homomorphism $h: A \to B\downarrow\Sigma_1$ exactly one Σ_2-homomorphism $f: I \to B$ exists with $f\downarrow\Sigma_1 = h$.

$$\begin{tikzcd} A \arrow[rr, "id"] \arrow[dr, "h"'] & & I \arrow[dl, "f"] \\ & B & \end{tikzcd}$$

If we choose for Σ_1 the empty signature then the notion of initial extension coincides with the notion of initial algebra.
Now, we can introduce a very general notion of theory by the definition of two kinds of theory extensions:

1. The empty signature (∅) is a theory.
2. If *T* is a theory based on signature Σ_1, then ***T <u>def</u>*** $\boldsymbol{\Sigma_2}$ is a theory, if $\Sigma_1 \subseteq \Sigma_2$ and if each T-model has a Σ_2-initial extension. The initial extensions are the models of *T <u>def</u>* Σ_2.
3. If T is a theory, based on signature Σ_1, then ***T <u>req</u>*** $\boldsymbol{\Sigma_2}$ is a theory, if $\Sigma_1 \subseteq \Sigma_2$. All Σ_2-algebras *A*, for which $A\downarrow\Sigma_1$ is a T-model are the *T <u>req</u>* Σ_2 models.
4. Theories are constructed only by rules 1, 2, 3.

By req-extensions in general parameter theories are specified. Here, is only required that the axioms are satisfied. By def-extensions always the initial algebras are specified.
For example, the only model of ∅ <u>def</u> BOOL is the initial algebra but each BOOL-algebra is a model of ∅ <u>req</u> BOOL.
Although we shall use in general def-extensions, it should be remarked that we can specify for example the equality relation in a very simple way by <u>req</u>-extensions:
DATA =
∅ <u>def</u> BOOL
<u>req</u>

sorts Data
opers equal-d (Data, Data) →Bool
axioms x, y: Data
equal-d(x, x) = true
if equal-d(x, y) = true then x = y
end

Although we specified the operation equal-d unique up to isomorphism the specification does not contain an algorithm, how to compute the corresponding truth value. If a def-extension is correct, then the specified operations are computable with respect to the given theory. It is proved that the function, which can be specified by def_extensions of natural numbers are exactly the partial recursive functions.

Specification of a stack

1. Stack with partial Operations

STACK1 =
def
sorts Data, Bool, Stack
opers d1, d2, …,dn → Data
true, false → Bool
empty → Stack
push (Data Stack) → Stack
def
opers is_empty Stack → Bool
pop (s: Stack iff is_empty(s) = false) → Stack
top (s: Stack iff is_empty(s) = false) → Data
axioms d: Data, s: Stack
is_empty(empty) = true
is_empty(push(d, s)) = false
pop(push(d, s)) = s
top(push(d, s)) = d
end

2. Stack with total Operations, but an Data-Error-Value (Error Recovery)

STACK2
def
sorts Data, Stack
opers d1, d2,…, dn, error: → Data
empty → Stack
push (Data, Stack) → Stack
top (Stack) → Data
pop (Stack) → Stack
axioms d: Data, s: Stack
pop(push(d,s)) = s
top(push(d,s)) = d
top(empty) = error
pop(empty) = empty
end

STACK2-Model:
A_{Data} = {d1, d2,…, dn, e}

$A_{Stack} = A_{Data}^{*}$
$push_A(x, w) = xw$
$pop_A(w) =$ if $w = xw'$ then w' else λ
$top_A(w) =$ if $w = xw'$ then x else e