Table Algebras: Algebraic Structures for Tabular Notation, Including Nested Header

(Preliminary Version)

Hitoshi Furusawa [1] and Wolfram Kahl [2]

[1] Research Center for Verification and Sementics (CVS), National Institute of Advanced Industrial Science and Technology (AIST)

and

[2] Software Quality Research Laboratory, McMaster University

Table Algebras: Algebraic Structures for Tabular Notation, Including Nested Headers

Hitoshi Furusawa[1⋆] and Wolfram Kahl[2⋆⋆]

[1] Center for Verification and Semantics, AIST, Amagasaki, Japan
hitoshi.furusawa@aist.go.jp
[2] Software Quality Research Laboratory, McMaster University, Hamilton, Canada
kahl@cas.mcmaster.ca

Abstract. We define a class of algebraic structures for the interpretation of tabular notations common in software requirements documentation and show how to use table algebra homomorphisms to define staged table semantics, translation between different kinds of tables, and table transformation. An important advantage of this approach is that it covers the practically important use of *nested headers* in a natural way.

1 Introduction

Parnas and his colleagues have shown that the use of two-dimensional expressions called *tables* is a very effective means to organise the presentation of complex relations appearing in rigorous software requirements specification documents [3,4,11,10,13]. To a large extent, the advantage of the use of tables is due to their graphical representation which makes complex mathematical notation much simpler and intuitively understandable.

The key to the interpretation of a table is to provide rules which determine a semantic operation that derives the meaning of a table from the meanings of its entries. Parnas was the first to supply formal table semantics by defining ten kinds of tables in [9]. Janicki later developed the currently predominant framework for table semantics based on cell connection graphs and table skeletons [5,6], and was able to define the relational semantics of the ten table kinds of Parnas as instances of this more general approach.

For the purpose of developing mechanised support tool for tables, Kahl introduced syntax and semantics of tables emphasising a compositional view of table structure [8]. Based on this compositional syntax and semantics, Kahl developed the basics of a table library in the functional programming language Haskell and a theory set in the theorem proving system Isabelle/HOL[3].

⋆ This work has been completed while H. Furusawa was visiting McMaster University supported by the Japan Society for the Promotion of Science (JSPS) Bilateral Programs for Science Exchanges.

⋆⋆ This work has been supported by the National Science and Engineering Research Council, NSERC, Canada

[3] see http://www.cas.mcmaster.ca/~kahl/Tables/.

In this paper, we present an algebraic structure, called *table algebras*, as basis for mathematical interpretation of tables.This is completely captured in the uniform framework of algebraic specifications and their models. The compositional view in [8] inspires and enables us to use the technique of free algebras, and we use our new, more general formulation to provide the first satisfactory treatment of *nested headers* that does not hide them via a pre-processing step, but fully integrates them into the syntactic and semantic framework.

We start with a short introduction of the compositional table concept of [8] in Sect. 2. We then provide a quick reference of the algebraic specification notation and nomenclature we use, and show in Sect. 4 how this can be applied to basic tables. Informed by this, we then discuss nested headers in Sect. 5, motivating our general definitions of table algebras in Sect. 6. Finally, we use free algebra machinery to obtain table specifications of a shape that allows straigh-forward implementation as data structures.

2 Compositional Table Syntax and Semantics

As a first example table we show a two-dimensional table T_f used in many references; it consists of the *header* H_1 in its *first dimension*, the header H_2 in its second dimension, and its *grid* G:

		$y = 7$	$y > 7$	$y < 7$	H_1
H_2	$x \geq 0$	0	y^2	$-y^2$	G
	$x < 0$	x	$x + y$	$x - y$	

In this paper, we always draw the first-dimension header on top of the table. Other authors frequently consider the left header of a two-dimensional table drawing as the first-dimension header.

If we want to consider T_f as result of horizontal concatenation of two subtables, then we have to equip both of these subtables with the same second-dimension header — one particular way to present T_f as horizontal concatenation of two subtables is $T_f = T_{f,\mathrm{a}} \,|||\, T_{f,\mathrm{b}}$ with subtables $T_{f,\mathrm{a}}$ and $T_{f,\mathrm{b}}$ as follows:

		$y = 7$	$y > 7$	$H_{1\mathrm{a}}$
H_2	$x \geq 0$	0	y^2	G_{a}
	$x < 0$	x	$x + y$	

		$y < 7$	$H_{1\mathrm{b}}$
H_2	$x \geq 0$	$-y^2$	G_{b}
	$x < 0$	$x - y$	

Since we want the original choice of decomposition to be arbitrary, table concatenation $|||$ has to be associative. Although demanding commutativity is often natural, it is not necessary and does not change our approach in any significant way. Therefore, we demand only associativity.[4] We also do not assume the existence of empty tables (which would be units for $|||$).

[4] This decision also has the advantage that for implementations, (non-empty) lists or concatenable arrays (for associativity alone) are usually more convenient to handle than multisets (for associativity and commutativity).

Tables like $T_{f,\mathrm{a}}$ are now considered as resulting from adding a single header cell to a one-dimensional table:

$$\begin{array}{cc} & \boxed{y=7}\ H_{1\mathrm{aa}} \\ H_2\ \begin{array}{|c|}\hline x\geq 0 \\ \hline x<0 \\ \hline \end{array} & \begin{array}{|c|}\hline 0 \\ \hline x \\ \hline \end{array}\ G_{\mathrm{aa}} \end{array} \quad = \quad (y=7)\ \triangleright \begin{array}{|c|c|}\hline x\geq 0 & x<0 \\ \hline 0 & x \\ \hline \end{array}\ \begin{array}{l} H_2 \\ G_{\mathrm{J}} \end{array}$$

In general, the infix operator $\triangleright$ adds a single header in a new first dimension, i.e., in a table $h \triangleright t$, the header h will be the single header of the *first* dimension, and the n-th dimension header of t is the $(n+1)$-th dimension header of $h \triangleright t$. Since we draw the first-dimension header on top of the grid, this explains why we "turned around" the grid in the last drawing.

The one-dimensional table seen above is again the result of a horizontal concatenation:

$$\begin{array}{|c|c|}\hline x\geq 0 & x<0 \\ \hline 0 & x \\ \hline \end{array} \quad = \quad \begin{array}{|c|}\hline x\geq 0 \\ \hline 0 \\ \hline \end{array} \ \Vert\!\Vert\!\Vert \ \begin{array}{|c|}\hline x<0 \\ \hline x \\ \hline \end{array}$$

The induvidual components are the results of adding a single header to a *zero-dimensional table*, or *cell*; we write $[\,e\,]$ for the cell containing e.

$$\begin{array}{|c|}\hline x\geq 0 \\ \hline 0 \\ \hline \end{array} \quad = \quad (x\geq 0)\ \triangleright\ \boxed{0} \quad = \quad (x\geq 0)\ \triangleright\ [\,0\,]$$

Using the operators $\Vert\!\Vert\!\Vert$ for horizontal concatenation, $\triangleright$ for adding headers, and $[\_]$ for constructing cells, the table T_f can then be written as expression in the following way (assuming that $\triangleright$ binds stronger than $\Vert\!\Vert\!\Vert$):

$$\begin{array}{r} (y=7)\triangleright\big((x\geq 0)\triangleright\ [\ \ 0\ \ ]\ \Vert\!\Vert\!\Vert\ (x<0)\triangleright\ [\ \ x\ \ ]\ \big) \\ \Vert\!\Vert\!\Vert\ (y>7)\triangleright\big((x\geq 0)\triangleright\ [\ y^2\ ]\ \Vert\!\Vert\!\Vert\ (x<0)\triangleright\ [x+y]\ \big) \\ \Vert\!\Vert\!\Vert\ (y<7)\triangleright\big((x\geq 0)\triangleright\ [(-y^2)]\ \Vert\!\Vert\!\Vert\ (x<0)\triangleright\ [x-y]\ \big) \end{array} \tag{1}$$

This table is *regular*, in the sense that in every subterm (up to associativity of $\Vert\!\Vert\!\Vert$) of the shape $(g_1 \triangleright (h_1 \triangleright t_1 \Vert\!\Vert\!\Vert \cdots \Vert\!\Vert\!\Vert\ h_m \triangleright t_m)) \Vert\!\Vert\!\Vert (g_2 \triangleright (k_1 \triangleright u_1 \Vert\!\Vert\!\Vert \cdots \Vert\!\Vert\!\Vert\ k_n \triangleright u_n))$, we have $\langle h_1, \ldots, h_m\rangle = \langle k_1, \ldots, k_n\rangle$.

Two-dimensional regular tables can be drawn in presentations like the first drawing of T_f; the previous literature only considers regular tables [6,2,7,14]. Following [8], we allow non-regular tables, also called *ragged tables*[5], and do not restrict ourselves to regular tables unless explicitly mentioned.

By adding a simple type discipline, [8] arrived at the following inductive definition of tables:

Definition 2.1 For each type α, the type $\mathbb{T}\ \langle\,\rangle\ \alpha$ of *zero-dimensional tables*, or *cells, with content type* α is the isomorphic image of α via the cell constructor

$$[\_] : \alpha \to \mathbb{T}\ \langle\,\rangle\ \alpha\ .$$

[5] name proposed by Dave Parnas

3 Signatures, Specifications, and Many-sorted Algebras

To make this paper reasonably self-contained, we now give a short summary of basic concepts of many-sorted algebra, using the language of algebraic specifications.

Definition 3.1 A *signature* $\Sigma = (\mathcal{S}, \mathcal{O})$ consists of a finite set $\mathcal{S}$ of *sorts* (symbols serving as "type names") and a countable set $\mathcal{O}$ of *operator symbols* or *function symbols*, where each function symbol $f : \mathcal{O}$ is equipped with its "functionality" in the shape of a non-empty sequence $\langle s_1, \ldots, s_n, t\rangle$ of finitely many source sorts $s_1, \ldots, s_n$ and a single target source t, written $f : s_1 \times \cdots \times s_n \to t$.

If k is a function symbol with target sort t and empty sequence of source sorts, then we consequently write $k :\to t$; such function symbols are also called *constant symbols*. □

A signature $\Sigma_1 = (\mathcal{S}_1, \mathcal{O}_1)$ is a *sub-signature* of $\Sigma_2 = (\mathcal{S}_2, \mathcal{O}_2)$ iff $\mathcal{S}_1 \subseteq \mathcal{S}_2$ and $\mathcal{O}_1 \subseteq \mathcal{O}_2$.

A signature can be considered as a hypergraph with sorts as nodes and function symbols as directed hyperedges.

A sub-signature *induced by* a set of sorts is the sub-hypergraph induced by this set of sort nodes in the usual graph-theoretic way.

Definition 3.2 Given a signature $\Sigma = (\mathcal{S}, \mathcal{O})$, a *$\Sigma$-algebra* $\mathcal{A}$ consists of the following items:

- a sort-indexed family $(s^{\mathcal{A}})_{s:\mathcal{S}}$ of *carrier sets*, i.e., for every sort $s : \mathcal{S}$, a set $s^{\mathcal{A}}$, and
- for every function symbol $f : \mathcal{O}$ with $f : s_1 \times \cdots \times s_n \to t$ a total function $f^{\mathcal{A}}$ from the Cartesian product $s_1^{\mathcal{A}} \times \cdots \times s_n^{\mathcal{A}}$ of the source sort carriers to $t^{\mathcal{A}}$, the target sort carrier. □

Definition 3.3 Given a sub-signature $\Sigma_1 = (\mathcal{S}_1, \mathcal{O}_1)$ of $\Sigma_2 = (\mathcal{S}_2, \mathcal{O}_2)$, the *$\Sigma_2$-reduct* of a Σ_1-algebra $\mathcal{A}$ is the Σ_2-algebra $\mathcal{A}\,|_{\Sigma_2}$ consisting only of interpretations (taken from $\mathcal{A}$) for the items in Σ_2. □

Definition 3.4 Let a signature $\Sigma = (\mathcal{S}, \mathcal{O})$ and two Σ-algebras $\mathcal{A}$ and $\mathcal{B}$ be given.

A *Σ-compatible family of functions from $\mathcal{A}$ to $\mathcal{B}$* is a $\mathcal{S}$-indexed family of functions $\Phi = (\Phi_s)_{s:\mathcal{S}}$ such that $\Phi_s : s^{\mathcal{A}} \to s^{\mathcal{B}}$ for every sort $s : \mathcal{S}$.

A *Σ-algebra homomorphism from $\mathcal{A}$ to $\mathcal{B}$* is a Σ-compatible family of total functions from $\mathcal{A}$ to $\mathcal{B}$ such that for every function symbol $f \in \mathcal{O}$ with $f : s_1 \times \cdots \times s_n \to t$, the following equation holds:

$$\forall\, x_1 : s_1^{\mathcal{A}};\ \ldots;\ s_n^{\mathcal{A}} \bullet f^{\mathcal{B}}(\Phi_{s_1}(x_1), \ldots, \Phi_{s_n}(x_n)) = \Phi_t(f^{\mathcal{A}}(x_1, \ldots, x_n))\ .$$ □

Assuming standard definitions of terms, term languages, equations, and validity, we just show the definitions for equational specifications and their models:

Definition 3.5 A *Σ-specification* is a triple $(\Sigma, \mathcal{X}, \mathcal{E})$ consisting of a signature $\Sigma = (\mathcal{S}, \mathcal{O})$, a sort-indext family $\mathcal{X} = (\mathcal{X}_s)_{s \in \mathcal{S}}$ of finite sets of *sorted variables*, and a set $\mathcal{E}$ of *equations*, where each equation $l = r$ is a pair of terms l and r from the term language $\mathcal{T}_{\Sigma,\mathcal{X}}$ of well-formed terms with function symbols from Σ and variables from $\mathcal{X}$.

We call a specification *sensible* if no equation in $\mathcal{E}$ is of the shape "$x = e$" or "$e = x$" for a variable x and an arbitrary expression e. □

It is easy to see that in a sensible specification $(\Sigma, \mathcal{X}, \mathcal{E})$, no equation of the forbidden shape (except $x = x$) can be derived from the equations in $\mathcal{E}$.

Definition 3.6 A spec-*model* of a specification $\mathsf{spec} = (\Sigma, \mathcal{X}, \mathcal{E})$ is a Σ-algebra that satisfies all equations in $\mathcal{E}$. □

Note that if $x \in \mathcal{X}_s$ and $x = e$ is an equation in spec, which is therefore not sensible, then in every spec-model M, the carrier set s^M has only one element.

4 Basic Table Algebra Specifications

The first step of formalising tables and their semantics in a conventional algebraic way is to fix the signature of table algebras, and there, we have to start with identifying the sorts.

For treating n-dimensional tables, there are two kinds of objects we have to deal with, and, accordingly, two groups of sorts:

- The basic table constituents are the elements that go into grid cells and header cells. With respect to the typing system used in Def. 2.1, we now change the numbering of the header sorts to go "inside-out", so we will have sorts $\mathbf{H}_0$ for grid cell contents and $\mathbf{H}_i$ for the headers from the $(n+1-i)$-th dimension, for $i \in \{1, \ldots, n\}$.
- For the values corresponding to k-dimensional subtables we will have sort $\mathbf{S}_k$ for $k \in \{0, \ldots, n\}$.

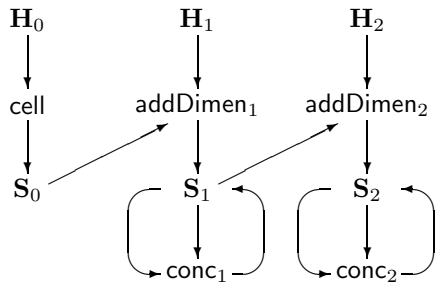
The operations (function symbols) all have subtable sorts as target:

- $\mathsf{cell} : \mathbf{H}_0 \to \mathbf{S}_0$ maps cell contents to cells, or zero-dimensional subtables (the function f in TESs).
- $\mathsf{addDimen}_i : \mathbf{H}_i \times \mathbf{S}_{i-1} \to \mathbf{S}_i$ for $i \in \{1, \ldots, n\}$ corresponds to $\triangleright$ at dimension $n - i + 1$ (in the general TES pattern, this is $\otimes_{n-i+1}$).
- $\mathsf{conc}_i : \mathbf{S}_i \times \mathbf{S}_i \to \mathbf{S}_i$ for $i \in \{1, \ldots, n\}$ corresponds to concatenation $|||$ at dimension $n - i + 1$ (in the general TES pattern, this is $\oplus_{n-i+1}$).

The material so far defines the *basic n-dimensional table signature* $\mathsf{sigTable}_n$.

(In comparison with the TESs of [8] mentioned in Sect. 2, we omit the wrapper w — it would require a separate result sort and a unary operation from $\mathbf{S}_n$ to that result sort.)

Here is a drawing of the signature hypergraph of $\mathsf{sigTable}_2$, drawn as bipartite graph; for each binary operator the two incoming edges are ordered:



To finish the specification, we have only one group of axioms:

– For each $i \in \{1, \ldots, n\}$, we demand associativity of conc_i:

$$\forall\, x, y, z : \mathbf{S}_i \bullet \mathsf{conc}_i(x, \mathsf{conc}_i(y, z)) = \mathsf{conc}_i(\mathsf{conc}_i(x, y), z)$$

Adding these axioms to $\mathsf{sigTable}_n$ yields the *basic n-dimensional table algebra specification* $\mathsf{specTable}_n$.

If T is a model of $\mathsf{sigTable}_n$, then we call T a *basic n-dimensional table algebra*.

The set $\mathbb{T}\ \langle \beta_1, \ldots, \beta_n \rangle\ \alpha$ of n-dimensional tables as defined in Def. 2.1 is a basic n-dimensional table algebra where each element records the history of its construction (up to associativity of concatenation).

5 Nested Headers

Parnas described "abbreviated grids" [9, Sect. 5] as a means to further structure the information in table headers (we are not aware of any application of "abbreviated grids" to the main table grid), essentially turning the list structure of headers into a tree structure, frequently used as a decision tree. The resulting *nested headers* seem to be an extremely popular table feature; for example, Wassyng and Lawford reported that tables epuipped with such nested headers were used very frequently in their project [13].

However, it appears that abbreviated grids have not yet been discussed in any of the other approaches to table semantics. Parnas treats them somewhat like preprocessor macros that can be considered as already flattened out when the semantics is calculated. The fact that the same reasoning could also be used for the table structure itself should be a convincing argument for treating nested headers, which are established as a useful feature of tabular notation, on an equal footing with the "standard" table structure.

The approach we present here to achieve this can be seen as a generalisation and unification of the two different approaches to coping with nested headers explored in [8].

The structure of a table header constructed using abbreviated grids is that of a node-labelled ordered forest, i.e., a sequence of node-labelled trees where

outgoing edges are linearly ordered. We shall use the name *nested headers* for table headers that are potentially constructed from "abbreviated grids".

The most frequent use of nested headers is for factoring out common parts of conjunctions in condition headers, as in the following example ("%" stands for the modulus operation):

	$x \,\%\, 2 = 0$	$x \,\%\, 2 \neq 0$	
		$x \,\%\, 3 = 0$	$x \,\%\, 3 \neq 0$
$y \geq 0$	0	y^2	$-y^2$
$y < 0$	x	$x + y$	$x - y$

(In practice, the header with the deeper nesting is usually drawn to the left of the table — in fact, as can be seen in [13], one-dimensional tables with nested headers are quite common.)

A different use of nested headers is to factor out arbitrary parameterisation, using what Parnas called "substitution grids", as in the following exmaple:

T_{nh1}

		x $\%\, 3$ = _		
		0	1	2
$y \,\%\, 2$ = _	0	0	y^2	$-y^2$
	1	x	$x + y$	$x - y$

This can be seen as an abbreviation of the following conventional table:

	$x \,\%\, 3 = 0$	$x \,\%\, 3 = 1$	$x \,\%\, 3 = 2$
$y \,\%\, 2 = 0$	0	y^2	$-y^2$
$y \,\%\, 2 = 1$	x	$x + y$	$x - y$

The crucial step in defining an algebraic structure for nested headers that does not make it necessary to consider them as an abbreviation for an operation joining them with their goverend sub-headers is to recognise that a nested-header table like T_{nh1} should *not* be considered as constructed from the header "$x \,\%\, 3$ = _", but rather should be considered as the result of a horizontal concatenation, for example of the following two tables:

		x $\%\, 3$ = _
		0
$y \,\%\, 2$ = _	0	0
	1	x

$\|\|\|$

x $\%\, 3$ = _	
1	2
y^2	$-y^2$
$x + y$	$x - y$

This shows that a governing header element of a nested header can be considered as *shared structure*, shared among the horizontally concatenated sub-tables just like the lower-dimension headers.

A direct consequence is that the table T_{nh1} must be considered as equivalent to the following, and that the primary subdivisions in this fully expanded header grid are vertical, not horizontal:

	$x \,\%\, 3 \;=\; \_$	$x \,\%\, 3 \;=\; \_$	$x \,\%\, 3 \;=\; \_$
	0	1	2
$y \,\%\, 2 \;=\; \_$ 0	0	y^2	$-y^2$
$y \,\%\, 2 \;=\; \_$ 1	x	$x + y$	$x - y$

We continue to ignore the nested header also in the next decomposition step for the left subtable in from above, which produces an application of $\triangleright$:

x %3 = _
0

$\triangleright$

y %2 = _	
0	1

0	x

Only now, the nested header is decomposed, using an operation that will be interpreted by a substitution operation.

This analysis of nested headers coincides with the conventional view that nested headers are essentially abbreviations allowing graphical sharing of header components. The difference is that we view the operation of header nesting at the same level as the operations of header adding and table concatenation, since they belong to the "table part" of the notation, and we wish to provide foundations for semantics and tool support for the whole table notation.

From Parnas' explanations it follows, although not very obviously, that different abbreviation mechanisms appear to be allowed within a single header. For this, it is necessary to equip each nested-header-separation-line with information about the abbreviation mechanisms used. In our context, this means that we will not necessarily be restricted to a single function symbol representing header nesting, but may have to provide several such symbols even within a single header.

To accomodate nested headers, we therefore extend the table signature hypergraph by operations targetted at the header sources. This may then also induce introduction of additional sorts — while decision tree headers need only a single operation with signature $\mathbf{H}_i \times \mathbf{H}_i \to \mathbf{H}_i$ that will be interpreted as conjunction of formulae, substitution headers can usefully be considered as composed from elements of two different sorts, and none of these two sorts necessarily has to be a header sort.

We formally define the resulting generalised concepts of table signature and table algebra in the next section.

6 General Table Algebra Specifications

As we have seen in the last section, we understand nested headers, or abbreviated grids in general, as new ways to provide header (or grid) cell contents, but not

as new ways to construct tables. Rather, we consider the graphical presentation of nested headers as another way to graphically share common components of different subtables, but the sharing itself is not reflected in the semantic treatment.

From the discussion in the previous section, we see that a nested-header table algebra with different kinds of nesting in the two dimensions can be based on a signature graph like the following:

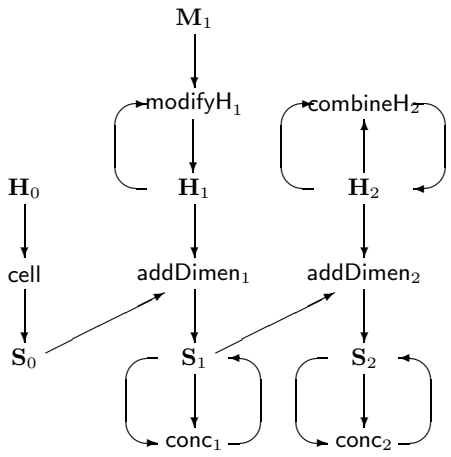


Fig. 1. A generalised table signature for two kinds of nested headers

To make this a flexible concept, we define generalised table signature only with very few restrictions:

Definition 6.1 An *n-dimensional table signature* is a signature Σ such that
- $\mathsf{sigTable}_n$ is a sub-signature of Σ
- in Σ considered as a graph, all hyperedges (i.e., operations) incident with any subtable sort $\mathbf{S}_i$ are hyperedges from $\mathsf{sigTable}_n$,
- $\mathbf{S}_n$ is reachable from each sort. □

The second condition guarantees that it is still discernible which dimension such a table has, since restricts the top-level structure of expressions over a general n-dimensional table signature to be precisely the term language of $\mathsf{sigTable}_n$.

Our restrictions on the laws enforced upon all generalised table algebras are motivated by the desire to guarantee at least a basic degree of "graphical representability" of free table algebras.

Definition 6.2 An *n-dimensional table algebra specification* is a specification $(\mathsf{sigE}, \mathcal{X}, \mathcal{E})$ such that

– sigE is an n-dimensional table signature,
– $\mathsf{specTable}_n$ is a sub-specification of $(\mathsf{sigE}, \mathcal{X}, \mathcal{E})$, and
– besides the laws of $\mathsf{specTable}_n$, $\mathcal{E}$ only contains associativity laws for binary homogeneous operators. □

In the nested header example from above, on would naturally impose an associativity law for $\mathsf{combineH}_2$.

To see how our restrictions on the laws interact with the restrictions of graphical presentation consider, for example, a homogeneous binary operator. If this operator is not restricted by any laws, it gives rise to a groupoid, the elements of which can be drawn as binary trees with the operator labelling the node and elements of the groupoid's carrier set that lie outside the range of the operator labelling the leaves.

If associativity is imposed, the trees collapse into non-empty sequences — this is the most usual situation.

However, if any kind of permutation law, as for example commutativity, is imposed, then the drawings themselves as syntactic tables do not immediately satisfy these laws. One might still make the point that two different drawings related by such a permutation are intended to "denote the same table", our machinery from below carries over to the case of permutation laws. Therefore, our decision against permutation laws can be seen as a matter of taste, and could easily be reverted.

A similar argument might still be made for unit laws.

Other kinds of laws, however, would break away from what could reasonably be understood as *syntactic* equivalence, so we have to exclude such laws since our interest here is in providing a semantics to tables that are considered as essentially a syntactic device.

The framework for table semantics is now immediately generated by the model concept of Def. 3.6:

Definition 6.3 If $\mathsf{specT} = (\mathsf{sigE}, \mathcal{X}, \mathcal{E})$ is an n-dimensional table algebra specification, then *table algebra over* specT is just a specT-model. □

As we will see below, free table algebras exist, and are therefore the natural formalisation of tables as syntactic objects.

These free table algebras are initial objects in appropriate categories of table algebras, and this fact directly turns every table algebra T into a semantics for the repective kind of tables — the semantics mapping is the unique arrow from the algebra of syntactic tables into T.

In the next section, we show how syntactic table algebras can be transformed into a canonic shape that is directly accessible to implementation as data structure due to the restrictions we imposed on table specifications.

7 Well-Based Table Syntax

We concentrate on properties of free table algebras, i.e., on tables as syntactic objects, and use some fairly standard material concerning free algebras to be

able to transform general table specifications into a canonic form where free algebras have useful additional properties.

One frequently finds formulations like "the free monoids over set S". The implicit signature sigMon underlying these monoids has only one sort, which is target of both the unit and the composition operator; as a result, the set S is injectively embedded in the carrier of "the free monoid over set S".

The analogous statement for many-sorted equational algebraic specifications is the following:

Theorem 7.1 Let a *sensible* specification $\mathsf{spec} = (\Sigma, \mathcal{X}, \mathcal{E})$ with $\Sigma = (\mathcal{S}, \mathcal{O})$ be given.

For each indexed family of sets $\mathcal{U} = (\mathcal{U}_s)_{s:\mathcal{S}}$ there is then a free spec model $\mathcal{F}_\mathcal{U}$ such that $\mathcal{U}_s$ is *injectively* embedded in $s^{\mathcal{F}_\mathcal{U}}$ for every sort $s : \mathcal{S}$.

Proof. This follows from the fact that for a many sorted signature Σ, the forgetful functor U mapping Σ-algebras to sort-indexed families of sets has a left adjoint, since spec contains only equations. See [1, Sect. 7] for more details. □

In this context, the sources in a signature graph deserve special attention, so we define:

Definition 7.2 A sort $s : \mathcal{S}$ is called a *source sort* of $\Sigma = (\mathcal{S}, \mathcal{O})$ if $\mathcal{O}$ contains no function symbol with target s. □

Note that a source sort therefore also cannot have any constants.

In the following, we will frequently work with families of sets indexed over only some subset $\mathcal{Q}$ of $\mathcal{S}$; if $\mathcal{U} = (\mathcal{U}_s)_{s:\mathcal{Q}}$ is such a family, then we extend this to another family $\mathcal{U}^\mathcal{S}$ indexed over the whole of $\mathcal{S}$ by assigning the empty set to the remaining sorts: we set $\mathcal{U}^\mathcal{S}_s = \mathcal{U}_s$ for $s \in \mathcal{Q}$, and $\mathcal{U}^\mathcal{S}_s = \{\}$ for $s \in \mathcal{S} \setminus \mathcal{Q}$.

Using this notation in the proof, we see that the choice of interpretation of the source sorts is preserved by the construction of the free algebra:

Theorem 7.3 Let a specification $\mathsf{spec} = (\Sigma, \mathcal{X}, \mathcal{E})$ with $\Sigma = (\mathcal{S}, \mathcal{O})$, and a sort-indexed family $\mathcal{U} = (\mathcal{U}_s)_{s:\mathcal{S}}$ be given.

The source-sort embeddings for the free spec algebra $\mathcal{F}_{\mathcal{U}^\mathcal{S}}$ are isomorphisms.

Proof. Let $\mathcal{G}$ be the set of source sorts of Σ.

The forgetful functor from $\mathcal{M}_\mathsf{spec}$ to $Set^\mathcal{G}$ which is determined by the mapping $\mathcal{A} \mapsto (s^\mathcal{A})_{s:\mathcal{G}}$ then has a left adjoint, because Theorem 7.1 holds and there exists an adjunction between $Set^\mathcal{G}$ and $Set^\mathcal{S}$, where the forgetful functer is determined by the mapping $(\mathcal{U}_s)_{s:\mathcal{S}} \mapsto (\mathcal{U}_s)_{s:\mathcal{G}}$ and the left adjoint is determined by the mapping $(\mathcal{V}_s)_{s:\mathcal{G}} \mapsto (\mathcal{V}^\mathcal{S}_s)_{s:\mathcal{S}}$.

Also, each component of the unit with respect to the left adjoint is an isomorphism in $Set^\mathcal{G}$, because for each $s \in \mathcal{G}$, the carrier $s^{\mathcal{F}_\mathcal{U}}$ in the free algebra does not contain any elements resulting from application of operations. □

This property is useful for modularity of reasoning about and implementation of many-sorted algebras, since it turns the more implicit embedding of Theorem 7.1 into explicit function symbols of the signature, and the source interpretations into a proper sub-algebra of the free algebra.

This theorem suggests that for applications of free algebras "over" some families of sets one should work with signatures in which only the source sorts need to be instantiated. This is not always the case, so we first introduce some auxiliary concepts.

Definition 7.4 For a signature $\Sigma = (\mathcal{S}, \mathcal{O})$, a sort set $\mathcal{S}_0 \subseteq \mathcal{S}$ is called a *base* of Σ if adding a constant symbol to each sort in $\mathcal{S}_0$ is sufficient to let every sort have ground terms.

A signature is called *well-based* if the set of its source sorts is a base. □

Obviously, if there is a base consisting only of source sorts, then this cannot be further reduced. On the other hand, if there is no base consisting only of source sorts, then there may be several minimal bases, for example in a signature with two sorts s_1, s_2 and two function symbols $f : s_1 \to s_2$ and $g : s_2 \to s_1$, each singleton sort set is a base.

In table signatures, the base sorts are the sorts of "general cells", for example in the sense of Parnas [9] who refers to grid cells, header cells, and cells in nested headers (nested grids).

By the above definition, the basic n-dimensional table signature $\mathsf{sigTable}_n$ is, for every natural number n, obviously a well-based n-dimensional table signature with $\{\mathbf{H}_j \mid 0 \leq j \leq n\}$ as set of source sorts.

However, the nested-header signature of Fig. 1 is not well-based, since $\mathbf{H}_1$ and $\mathbf{H}_2$ are not source sorts, and are targets only of cyclic hyperedges, so both would have to be part of any base.

It is easy to see that our restrictions on table signatures have a useful effect here:

Proposition 7.5 In an n-dimensional table signature Σ, if $\mathcal{B}$ is a base, then, for the subtable sorts $\mathbf{S}_0, \ldots, \mathbf{S}_n$ of Σ, the set $\mathcal{B} \setminus \{\mathbf{S}_0, \ldots, \mathbf{S}_n\}$ is a base, too.

We call $\mathcal{B}$ a *table base* of Σ if $\mathcal{B} \cap \{\mathbf{S}_0, \ldots, \mathbf{S}_n\} = \varnothing$. □

Given a base, each non-well-based signature can be extended to a well-based signature by adding a sorce sort for each base sort that is not a source, and a unary embedding function symbol:

Definition 7.6 Let a signature $\Sigma = (\mathcal{S}, \mathcal{O})$ with a base $\mathcal{B} \subseteq \mathcal{S}$ be given.

Then $\Sigma_1 = (\mathcal{S}_1, \mathcal{O}_1)$ is a *well-based extension of* Σ *via* $\mathcal{B}$ if (let $\mathcal{B}'$ be the set of all base sorts from $\mathcal{B}$ that are not source sorts):

- Σ is a sub-signature of Σ_1,
- $\mathcal{S}_1 \setminus \mathcal{S}$ is isomorphic to $\mathcal{B}'$, with embedding $\mathsf{Base} : \mathcal{B}' \rightarrowtail\!\!\!\!\!\rightarrow \mathcal{S}_1 \setminus \mathcal{S}$,
- $\mathcal{O}_1 \setminus \mathcal{O}$ is isomorphic to $\mathcal{B}'$, with embedding $\mathsf{base} : \mathcal{B}' \rightarrowtail\!\!\!\!\!\rightarrow \mathcal{O}_1 \setminus \mathcal{O}$ and for every non-source base sort $b : \mathcal{B}'$, we have: $\mathsf{base}\ b : \mathsf{Base}\ b \to b$.

A specification $(\Sigma_1, \mathcal{X}, \mathcal{E})$ is a *well-based extension of a Σ-specification* $(\Sigma, \mathcal{X}, \mathcal{E})$ *via* $\mathcal{B}$ iff Σ_1 is a well-based extension of Σ via $\mathcal{B}$. □

Now the free models factor appropriately:

Theorem 7.7 Let a sensible specification $\mathsf{spec} = (\Sigma, \mathcal{X}, \mathcal{E})$ with $\Sigma = (\mathcal{S}, \mathcal{O})$ be given, and let $\Sigma_1 = (\mathcal{S}_1, \mathcal{O}_1)$ be a well-based extension of Σ via the base $\mathcal{B} \subseteq \mathcal{S}$.

Now let $\mathcal{G}$ be the set of source sorts of Σ_1; in terms of Def. 7.6, we then have $\mathcal{B}' = \mathcal{B} \setminus \mathcal{G}$.

For a family of sets $\mathcal{U} = (\mathcal{U}_s)_{s:\mathcal{B}}$ indexed over the *base sorts* only, define $\mathcal{W}$ indexed over $\mathcal{G}$ by setting $\mathcal{W}_s = \mathcal{U}_s$ for $s \in \mathcal{G} \cap \mathcal{B}$, and $\mathcal{W}_{\mathsf{Base}\ s} = \mathcal{U}_s$ for $s \in \mathcal{B}'$.

Then $\mathcal{F}_{\mathcal{U}^S}$ is isomorphic to the Σ-reduct of $\mathcal{F}_{\mathcal{W}^{S_1}}$.

Proof. Since the only connection between the sorts in $\mathcal{B}'$ and those in $\mathcal{O}_1 \setminus \mathcal{O}$ are the unary function symbols $\mathsf{base}\ b$ for $b \in \mathcal{B}'$, and no laws have been added for these function symbols, their interpretations in free algebras are injective. The statement follows then from Theorems 7.1 and 7.3 and the arguments in their proofs. □

Thanks to this theorem, we do not have to restrict considerations to well-based signatures, but can move to well-based extensions of non-well-based signatures where it is technically useful to do so.

In particular, it would be quite natural to specify a table type with nested headers using a non-well-based signature like the one in Fig. 1, and then implement this using the well-based extension, since free algebras over source-only bases are straight-forward to represent as data structures.

8 Conclusion

Starting from the compositional approach to table syntax and semantics of [8], we made the initial algebra aspects more explicit, visualised the resulting signature hypergraphs, and then found ourselves in a good position to generalise this understanding of table syntax in such a way that we can easily and naturally cover nested headers.

As a result, we provided an algebraic structure that is, in a certain sense, the simplest possible that is rich enough to capture tables (including nested headers), which means,

- we have tables as a result of a free construction, and
- the construction fits our intuition: if we prepare a set for each source sort of a table algebra specification ST, the elements of the resulting free ST-table-algebra are recognisable as tables.

As an additional benefit, defining syntactic tables as elements of a free table algebra provides a simple and clean way to define table semantics by providing appropriate table algebras and then automatically obtaining the corresponding inductively defined semantics.

References

1. Michael Barr and Charles Wells. *Category Theory for Computing Science*. Centre de recherches mathématiques (CRM), Université de Montréal, 3rd edition edition, 1999.
2. Jules Desharnais, Ridha Khedri, and Ali Mili. Interpretation of tabular expressions using arrays of relations. In E. Orlowska and A. Szalas, editors, *Relational Methods for Computer Science Applications*, volume 65 of *Studies in Fuzziness and Soft Computing*, pages 3–14, Heidelberg, 2001. Springer-Physica Verlag.
3. K. L. Heninger, J. Kallander, David Lorge Parnas, and J.E. Shore. Software requirements for the A-7E aircraft. NRL Memorandum Report 3876, United States Naval Research Laboratory, Washington DC, November 1978.
4. S. D. Hester, D. L. Parnas, and D. F. Utter. Using documentation as a software design medium. *Bell System Tech. J.*, 60(8):1941–1977, October 1981.
5. Ryszard Janicki. Towards a formal semantics of Parnas tables. In *Proc. of the* 17^{th} *Internat. Conf. on Software Engineering, Seattle, WA*, pages 231–240, 1995.
6. Ryszard Janicki and Ridha Khedri. On a formal semantics of tabular expressions. *Science of Computer Programming*, 39:189–213, 2001.
7. Ryszard Janicki, David Lorge Parnas, and Jeffery Zucker. Tabular representations in relational documents. In Chris Brink, Wolfram Kahl, and Gunther Schmidt, editors, *Relational Methods in Computer Science*, Advances in Computing Science, chapter 12, pages 184–196. Springer, Wien, New York, 1997.
8. Wolfram Kahl. Compositional syntax and semantics of tables. SQRL Report 15, Software Quality Research Laboratory, Department of Computing and Software, McMaster University, October 2003. available from http://www.cas.mcmaster.ca/sqrl/sqrl_reports.html.
9. David Lorge Parnas. Tabular representation of relations. Technical Report CRL Report 260, McMaster Univ., Communications Research Laboratory, TRIO (Telecommunications Research Inst. of Ontario), October 1992.
10. David Lorge Parnas. Inspection of safety critical software using program-function tables. In K. Duncan and K. Krueger, editors, 13^{th} *World Computer Congress 1994, Vol. 3: Linkage and Developing Countries*, volume A-53 of *IFIP Transactions*, pages 270–277. North-Holland, August 1994. Invited paper.
11. David Lorge Parnas, G. J. K. Asmis, and Jan Madey. Assessment of safety-critical software in nuclear power plants. *Nuclear Safety*, 32(2):189–198, 1991.
12. J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science. Prentice Hall, second edition, 1992. Out of print; available at http://spivey.oriel.ox.ac.uk/~mike/zrm/.
13. Alan Wassyng and Mark Lawford. Lessons learned from a successful implementation of formal methods in an industrial project. In Keijiro Araki, Stefania Gnesi, and Dino Mandrioli, editors, *FME 2003: Formal Methods*, volume 2805 of *LNCS*, pages 133–153. Springer, 2003.
14. Jeffery I. Zucker and Hong Shen. Table transformation: Theory and tools. Technical Report CAS 98-01, McMaster University, Dept of Computing and Software, 1998. also Communications Research Laboratory (CRL) Report 363.

For each type $\mathbb{T}\ bs\ \alpha$ of n-dimensional tables and each type β, the type $\mathbb{T}\ (\beta :: bs)\ \alpha$ of $(n+1)$-*dimensional tables with header types* $\beta :: bs$ *and cell type* α is the initial algebra generated by the two constructors

$$\begin{array}{l} \_ \triangleright \_ : \quad \beta \to \mathbb{T}\ bs\ \alpha \quad\quad \to \mathbb{T}\ (\beta :: bs)\ \alpha \\ \_ \,|||\, \_ : \mathbb{T}\ (\beta :: bs)\ \alpha \to \mathbb{T}\ (\beta :: bs)\ \alpha \to \mathbb{T}\ (\beta :: bs)\ \alpha \end{array}$$

and the associativity law for $|||$. □

For defining table semantics, [8] introduces *table evaluation structures (TESs)*; we will explain these here only by example: A TES for an n-dimensional table consists of two unary functions w and f, and n pairs $(\otimes_i, \oplus_i)$ of binary functions (here used as infix operators) grouped together in the following way:

$$w \,(\!|\, (\otimes_1, \oplus_1), (\otimes_2, \oplus_2) \,|\!)\, f$$

Applying this general TES to T_f from above replaces each occurrence of $[\_]$ by f, each second-dimension concatenation by $\oplus_2$, each second-dimension header adding by $\otimes_2$, analogously for the first-dimension operations, and "wraps" the resulting expression by applying w to it, resulting in the following:

$$w \begin{pmatrix} ((y=7) \otimes_1 (((x \geq 0) \otimes_2 \quad f(0)) & \oplus_2 ((x<0) \otimes_2 \quad f(x)))) \\ \oplus_1 ((y>7) \otimes_1 (((x \geq 0) \otimes_2 \quad f(y^2)) & \oplus_2 ((x<0) \otimes_2 f(x+y)))) \\ \oplus_1 ((y<7) \otimes_1 (((x \geq 0) \otimes_2 f((-y^2))) & \oplus_2 ((x<0) \otimes_2 f(x-y)))) \end{pmatrix}$$

The standard reading of T_f is to consider it as a "normal function table" [9,7]; which corresponds in the two-dimensional case to the following TES:

$$S_{\mathsf{ND}} \quad := \quad w_1 \,(\!|\, (\wedge, \vee),\ (\wedge, \vee) \,|\!)\, (z = \_)$$

with[6] $w_1(F) = \{x, y, z : \mathbb{R} \mid F \bullet (x, y) \mapsto z\}$.

We can now form the *tabular expression* consisting of the table T_f and this table evaluation structure, and use it to define a function f:

$$f := [\![\, S_{\mathsf{ND}} \,]\!]\, \ulcorner T_f \urcorner$$

By definition of tabular expressions via TES application, this is equivalent to writing:

$$f := \{x, y, z : \mathbb{R} \mid \begin{pmatrix} (y = 7 \wedge (\quad (x \geq 0 \wedge z = 0 \quad\quad\ \ ) \\ \qquad\qquad\quad \vee \ (x < 0 \wedge z = x \quad\quad\ \ ))) \\ \vee\ (y > 7 \wedge (\quad (x \geq 0 \wedge z = y^2 \quad\ \ ) \\ \qquad\qquad\quad \vee \ (x < 0 \wedge z = x + y\,))) \\ \vee\ (y < 7 \wedge (\quad (x \geq 0 \wedge z = -y^2 \ \ ) \\ \qquad\qquad\quad \vee \ (x < 0 \wedge z = x - y\,))) \end{pmatrix} \bullet (x, y) \mapsto z\}$$

Starting from this compositional understanding of tables and table semantics of [8], we now turn to a more general, algebraic way of presenting these issues.

[6] We use the conventions of the Z notation [12] for set comprehensions.

テーブル代数：テーブル表現の代数構造 (in English)
(算譜科学研究速報)
発行日：2004 年 12 月 10 日
編集・発行：独立行政法人産業技術総合研究所関西センター尼崎事業所
システム検証研究センター
同連絡先：〒661-0974 兵庫県尼崎市若王寺 3-11-46
e-mail：informatics-inquiry@m.aist.go.jp

Table Algebras: Algebraic Structures for Tabular Notation, Including Nested Headers
(Programming Science Technical Report)
December 10, 2004
Center for Verification and Semantics
AIST Kansai, Amagasaki Site
National Institute of Advanced Industrial Science and Technology (AIST)
3-11-46 Nakouji, Amagasaki, Hyogo, 661-0974, Japan
e-mail : informatics- inquiry@m.aist.go.jp