Algebra of Normal Function Tables

Martin von Mohrenschildt
Electrical and Computer Engineering
McMaster University, Hamilton, ON, Canada
mohrens@mcmaster.ca

May 29, 1997

Abstract

In contrast to classical algebra and analysis the functions encountered in computer science are usually piecewise continuous functions or functions whose evaluation rules change dramatically depending on a subset of the input values. Because of the pervasiveness of the if-then-else construction in programming, we have to extend classical mathematical methods to handle this kind of object. We approach this class of functions using tabular expressions. In this paper we define a function algebra, over a many sorted algebra, which is closed under composition. Then, this function algebra is extended to tables. This enables us to define the composition of tables. We show that this algebra of tables is closed under composition. We give two algorithms to compute the composition of tables.

1 Introduction

In literature we find several different approaches to define easily readable but mathematically precise notations for piecewise functions and functions changing definition depending on parameters [Mohrenschildt96], [Parnas94.1], [Bojanowski]. We are interested in a complete mathematical formal theory for functions of this type, clearly motivated by the development and needs of modern software engineering.
Tabular expressions or tables proved to be a precise instrument used in the documentation and specification of software. Tables offer an easy to read notation, whose structure is clean and can be used in many different contexts [Parnas95], [Heniger80]. In [Parnas95], [Janicki95.2] and [Janicki95.1] a precise mathematical definition of tables is given. We take a little different view point and consider tables as algebraic objects.

This paper is part of the project we are conducting at the Software Research Group at McMaster University to automate the verification of table based specifications. To be able to perform these verifications we need to verify properness of tables, compute the composition of tables, simplify tables, perform transformations of tables, e.g. change the dimension, and to decide the semantical equality of tables by defining a canonical from.
To get the idea, a normal function table is a table consisting of *headers* and a n-dimensional *grid*. The table

	$x_1 > 0 \wedge x_1 \leq 1$	$x_1 \leq 0$	$x_1 > 1$
$x_2 < x_1$	$x_1 - x_2$	$x_1^2 - x_2^2$	$-x_1 - x_2$
$x_2 \geq x_2$	$x_2 - x_1$	$x_2^2 - x_1^2$	$-x_1 + x_2$

defines a function of two variables x_1 and x_2. The headers are expressions of a logic L and the grid entries are expressions of an algebra. The headers describe conditions, which evaluated at a point, define an index to a grid element, which is then evaluated to a value.
Based on a many sorted algebra we define our function ring $\mathcal{FA}$. $\mathcal{FA}$ is an algebra of functions of several typed variables closed under composition. The composition in $\mathcal{FA}$ can be compared with the sequential composition in computer programs, executing two pieces of software one after another. Our definition of composition allows us to show associativity of this composition. This notion of composition, sometimes called symbolic composition [Colby], can nicely be modeled using our algebra $\mathcal{FA}$.
Next the algebra $\mathcal{FA}$ is extended to tables. We will prove that this algebra of normal tables is closed in the sense that composition of normal function tables results in a function that can effectively, using an algorithm, be represented as a single normal function table again. The algorithm depends only on the associative of the composition of functions.
The paper is outlined as follows: In the first section, we define the algebra, functions and evaluation. The second section discusses the concept of a logic needed to define conditions. Then we are ready to define the notion of tables. Tables are constructed using the defined algebra and the logic. These tables can be interpreted as functions by giving the semantics of evaluation. We then extend the algebra A to the algebra $\tilde{A}$. We define the extended algebraic operations and compositions. Finally, we present the main theorem, stating that any algebraic operation or composition performed on tables can be represented as a single normal table. Two algorithms are given to compute the composition of tables.

2 The Algebras

We define a many sorted algebra A_Σ by first defining a many sorted signature Σ and then giving an interpretation of the terms [Zucker]. This many sorted

algebra A_Σ is then used to define an algebra of functions $\mathcal{FA}$ with composition.

2.1 The many-sorted signature Σ

A *many-sorted signature* $\Sigma = (\mathbf{Sort}_\Sigma, \mathbf{Func}_\Sigma)$ where $\mathbf{Sort}_\Sigma$ is a finite set of *sorts* of Σ, and $\mathbf{Func}_\Sigma$ is a finite set of Σ-function symbols, each with its *type* $s_1 \times s_2 \times \cdots \times s_n \rightarrow s$, *arity* n, *domain sorts* $s_1, \cdots, s_n$, and *range sort* s. A function of arity 0 is a *constant*.
Examples of sorts with there signatures are:

$$\begin{array}{rrl} \texttt{bool}: & \vee, \wedge: & \texttt{bool} \times \texttt{bool} \rightarrow \texttt{bool} \\ & \neg: & \texttt{bool} \rightarrow \texttt{bool} \\ & \mathbf{true}, \mathbf{false}: & \rightarrow \texttt{bool} \end{array}$$

A finite set of n elements equipped with equality is called `fin`.

$$\begin{array}{rrl} \texttt{fin}: & =: & \texttt{fin} \times \texttt{fin} \rightarrow \texttt{bool} \\ & a_1, a_2, \cdots a_n: & \rightarrow \texttt{fin} \end{array}$$

$$\begin{array}{rrl} \texttt{real}: & +, -, *, /: & \texttt{real} \times \texttt{real} \rightarrow \texttt{real} \\ & 0, 1 & \rightarrow \texttt{real} \\ & =, <, >, \leq, \geq: & \texttt{real} \times \texttt{real} \rightarrow \texttt{bool} \end{array}$$

Other Σ-functions symbols and sorts can be added if needed.
Given is also a finite set of variables $x, y, x_1, \cdots$. The variables are typed, $x :: s$ declare that x is of type s.

Definition 2.1 (Term of Σ) *Given a set of typed variables $x :: s_{i_1}, y :: s_{i_2}, \cdots, x_n :: s_{i_n}$, terms of type s of the many-sorted signature Σ are*

- *a constant of Σ is a term of type s,*
- *a variable $x_i :: s$ is a term of type s,*
- *if f is a Σ-function symbol of type $s_{i_1} \times s_{i_2} \times \cdots \times s_{i_n} \rightarrow s$ and t_k are terms of type s_{i_k} then $f(t_1, t_2, \cdots, t_n)$ is a term of type s..*

2.2 The many-sorted Algebra A_Σ

Given a signature Σ. The many sorted algebra A_Σ has for each sort s a *carrier set* A_s and each Σ-function symbol f of type $s_1 \times s_2 \times \cdots \times s_m \rightarrow s$ a function $f^A : A_{s_1} \times A_{s_2} \times \cdots \times A_{s_m} \rightarrow A_s$, the *interpretation* of f in A.
Our algebra A_Σ contains now a interpretation for the Σ-function symbols and constants.

Definition 2.2 (Substitution, Constant Substitution)
A substitution is a set of expression of the form $x_i \mapsto p_i$ where x_i is a variable of type s_j and p_i is a term of the same type s_j of the algebra A_Σ.
A constant substitution is a set of expressions of the form $x_i \mapsto a$ where x_i is a variable of type s_j and $a \in A_{s_j}$.

Note, a constant substitution is not a special case of a substitution. Constant substitutions, some times called states, give a semantical meaning to the terms.

- For terms t $\mathbf{var}(t)$ is set of variables occurring in t.
- For a substitution V $\mathbf{var}(V)$ is the set of variables occurring in the terms being the right hand sides of the substitutions.

$$\mathbf{var}(V) := \{x_i | x_i \in \mathbf{var}(t_j), (x_j \mapsto t_j) \in V\}$$

- With $\mathbf{rangevar}(V)$ we denote the set of variables being the left hand side of the substitution V.

$$\mathbf{rangevar}(V) := \{x_i | (x_i \mapsto t_i) \in V\}$$

Next we define the mapping *val* used for substitution and constant substitutions:

Definition 2.3 *The mapping val takes two arguments, a term of the algebra A_Σ and a substitution (constant substitution) V and returns a term of A_Σ (a constant of A_s). $val(t, V)$ is recursively defined by*

- *If $var(t) = \{\}$ then $val(t, V) = t$.*
- *If t is a variable, $t = x_i$, then $val(t, V)$ is the right hand side of the substitution $x_i \mapsto p_i$ of V.*
- *If t is composed by op_i, $t = op_i(t_1, t_2, \cdots, t_{s_i})$ then $val(t, V) = op_i(val(t_1, V), val(t_2, V), \cdots, val(t_{s_i}, V))$.*

Note, *val* represents simultaneous substitution, the right hand side of any expression in the substitution is not affected by any other substitution in the set of substitutions V. For example $val(x_1, \{x_1 \mapsto x_2, x_2 \mapsto x_3\})$ is x_2 and not x_3. We write $\mathbf{var}(\alpha, \beta, \cdots)$ for $\mathbf{var}(\alpha) \cup \mathbf{var}(\beta) \cup \cdots$.

Definition 2.4 (Evaluation of a Term) *A term t of type s is evaluated under a constant substitution V with $\mathbf{var}(t) \subseteq \mathbf{rangevar}(V)$ to $val(t, V)$ an element of the set A_s.*

The evaluation of a term under a constant substitution is often called the *state* of a term.
We extend the function *val* to substitutions. Let F and G be substitutions then

$$val(F, G) := \{x_i \mapsto val(t_i, G) | (x_i \mapsto t_i) \in F\}.$$

We close this section by the remark that *val* is not associative.

2.3 The Function algebra $\mathcal{FA}$

Now we define the notion of a function of the algebra $\mathcal{FA}$ over the algebra A_Σ. The algebra $\mathcal{FA}$ is very closely related the the assignments of expressions to variables commonly used programming languages.

Definition 2.5 (Function, Evaluation of Function) *A function of $\mathcal{FA}$ is a substitution of A_Σ. A constant function is a constant substitution of A_Σ.*
A function F is is evaluated under the constant substitution V with $\mathbf{var}(f) \subseteq \mathbf{rangevar}(V)$ *to $val(F, V)$ resulting in a constant function.*

We use capital letters F, G, H for $\mathcal{FA}$ functions and t, g, f for terms of A_Σ.

Definition 2.6 (Composition of Functions) *Let F and G be $\mathcal{FA}$ function. We define the composition of F and G $F \circ G$ by*

$$F \circ G := val(F, G) \cup \{(x_i \mapsto t_i) \in G | x_i \notin \mathbf{rangevar}(F)\}$$

For example the two functions $F := \{x_1 \mapsto x_1 + x_2\}$ and $G := \{x_2 \mapsto x_2 + 1\}$ are composed to $F \circ G = \{x_1 \mapsto x_1 + x_2 + 1, x_2 \mapsto x_2 + 1\}$.

Theorem 2.7 *The function algebra $\mathcal{FA}$ with composition $\circ$ is a monoid.*

Proof We have to verify that there exists a neutral element of composition and that composition is associative.
The neutral element of composition is $I := \{\}$

$$I \circ F = \{val(\{\}, F) \cup \{(x_i \mapsto t_i) \in F | x_i \notin \mathbf{rangevar}(\{\})\} = F$$

and

$$F \circ I = val(F, \{\}) \cup \{(x_i \mapsto t_i) \in \{\} | x_i \notin \mathbf{rangevar}(F)\} = F$$

The composition is associative. We have verify that $(F \circ G) \circ H = F \circ (G \circ H)$. We show $(F \circ G) \circ H \subseteq F \circ (G \circ H)$ and $F \circ (G \circ H) \subseteq (F \circ G) \circ H$.

Let $(x_i \mapsto t_i) \in F \circ (G \circ H)$ then

$x_i \notin \mathbf{rangevar}(F)$ then $x_i \in \mathbf{rangevar}(G \circ H)$

$x_i \notin \mathbf{rangevar}(G)$ then $(x_i \mapsto t_i) \in H$

$$\Rightarrow (x_i \mapsto t_i) \in (F \circ G) \circ H$$

$x_i \in \mathbf{rangevar}(G)$ then $(x_i \mapsto g) \in G$ and $(x_i \mapsto val(g, H)) \in H$ with $val(g, H) = t_i$.

$$\Rightarrow (x_i \mapsto t_i) \in (F \circ G) \circ H$$

$x_i \in \mathbf{rangevar}(F)$ then $(x_i \mapsto f) \in F$ and $t_i \mapsto val(f, G \circ H)$.

then $x_i \mapsto val(f, val(G, H) \cup \{(x_l \mapsto t_l) \in H | x_l \notin rangevarG\})$

hence $x_i \mapsto val(val(f, H), G)$.

$$\Rightarrow (x_i \mapsto t_i) \in (F \circ G) \circ H$$

The other direction $(F \circ G) \circ H \subseteq F \circ (G \circ H)$ is similar and not presented here.

∎

This monoid property enables us to give a meaning to F^n by $F^0 := \{\}$ and $F^n = \underbrace{F \circ F \circ F \cdots \circ F}_{n}$. This enables us to define a loop construct as part of our structure. Together with tables the structure will be computational complete. (Any computable function can be expressed.)

Definition 2.8 (Extensional Equality of Functions) *The two functions F and G of $\mathcal{FA}$ are extensionally equal $F \simeq G$ if for all constant substitutions P with $\mathbf{var}(F, G) \subseteq \mathbf{rangevar}(P)$ $F \circ P = G \circ P$, (equality of the sets)*

The two functions $F := \{x_1 \mapsto x_1\}$ and $G := \{x_2 \mapsto x_2\}$ are extensional equal since for all $P := \{x_1 \mapsto a, x_2 \mapsto b\}$ with $a \in A_s$ and $b \in A_r$ $F \circ P = G \circ P = \{x_1 \mapsto a, x_2 \mapsto b\}$.

Example 2.9 *Let $x_1 ::$ `real`, $x_2 ::$ `real` and $x_3 ::$ `bool`*
The term $x_1 + x_2$ is evaluated under the constant substitution $\{x_1 \mapsto 1, x_2 \mapsto 2\}$ to $val(x_1 + x_2, \{x_1 \mapsto 1, x_2 \mapsto 2\}) = 1 + 2 = 3$.
The function $F = \{x_1 \mapsto x_1 + x_2, x_2 \mapsto x_2 + 2\}$ evaluated under the constant substitution $\{x_1 \mapsto 1; x_2 \mapsto 2\}$ is $val(F, \{x_1 \mapsto 1, x_2 \mapsto 2\} = \{x_1 \mapsto 3, x_2 \mapsto 4;\}$
The composition of $F := \{x_2 \mapsto x_2 + x_2, x_3 \mapsto x_1 > 0\}$ and $G := \{x_1 \mapsto x_1 + 1, x_2 \mapsto x_2 + x_1\}$ is $F \circ G = \{x_1 \mapsto x + 1 + 1, x_2 \mapsto 2x_2 + 2x_1, x_3 \mapsto x_1 + 1 > 0, \}$.

Example 2.10 *We can use $\mathcal{FA}$ to decide the equality of "assignments":*

```
y:=x-3;          y:=x-3;          x:=x+2;
x:=y+5;          x:=x+2;          y:=x-5;
```

The $\mathcal{FA}$ representation of all these three assignments is:

$$\{x \mapsto x + 2, y \mapsto x - 3\}$$

In [Colby] problems of this kind are examined using a different approach.

Remark $\mathcal{FA}$ provides a method to decompose a program for parallel execution. For example the instruction sequence $x_1 \mapsto a; x_2 \mapsto x_1 + b; x_1 \mapsto x_2 + x_2$ written as $\mathcal{FA}$ functions $\{x_1 \mapsto a\} \circ \{x + 2 \mapsto x_1 + b\} \circ \{x_1 \mapsto x_2 + x_1;\}$ resulting in $\{x_1 \mapsto a + a + b, x_2 \mapsto a + b\}$. The elements of a $\mathcal{FA}$ functions can allays be computed in parallel since there right-hand sides are independent.

3 The Logic

From now on we assume that `bool` is one of the sorts in our many sorted algebra A_Σ. A function symbol with range-type `bool` is a predicate. Note that our algebra is closed under boolean combinations of predicates.

Definition 3.1 *A condition c of the algebra A_Σ is a term with range sort* `bool`.

A condition c evaluates under the constant substitution V with $\mathbf{var}(c) \subseteq \mathbf{var}(V)$ resulting in **true** or **false**.

Example 3.2 *With $x_1 ::$ `real` and $x_2 ::$ `real` and $m ::$ `fin`$\{A, B, C\}$ $x_1 > x_2$ or $x_1 + x_2 \leq 0$ or $m = A \wedge x_1 < 0$ are conditions.*

The composition of a condition c and a function F is defined by

$$c \circ F := val(c, F)$$

Note that the conditions of the algebra A_Σ are closed under composition with functions.

Definition 3.3 *A condition c is always true or false iff for all constant substitutions V with $\mathbf{var}(c) \subseteq \mathbf{rangevar}(V)$ $val(c, V) =$ **true**; $val(c, V) =$ **false**. We write $c =$ **true** ; $c =$ **false**.*

Consequently we write $c_1 \wedge c_2 =$ **true** iff $val(c_1, V) \wedge val(c_2, V) =$ **true**.

Definition 3.4

- *A set of conditions $c_1, c_2, \cdots, c_n$ is disjoint, iff $c_1 \wedge c_2 \wedge \cdots \wedge c_n =$ **false**.*
- *A set of conditions $c_1, c_2, \cdots, c_n$ is universal if $c_1 \vee c_2 \vee \cdots \vee c_n =$ **true**.*

4 Normal Function Tables over an Algebra

Let us define the notion of tables [Parnas95] [Zucker93], [Janicki95.1]. For the concepts studied in this paper, we restrict ourself to the so called proper normal function tables defined in [Parnas95], [Zucker93], [Zucker94]. A general definition of tables can be found in "Towards a Formal Semantic of Tables" [Janicki95.2].
We start with an example:

$x < 1$	$x \geq -1 \wedge x \leq 1$	$x > 1$
$\{x \mapsto -x\}$	$\{x \mapsto x^2\}$	$\{x \mapsto x\}$

This normal function table T with variable $x ::$ `real` defines a function. There is one *header* H with three conditions $x < 1$, $x \geq -1 \wedge x \leq 1$ and $x > 1$ and a *grid* containing functions. Evaluating this table under the constant substitution $\{x \mapsto -2\}$ results in $\{x \mapsto 2\}$ and under $\{x \mapsto 1/2\}$ we get $\{x \mapsto 1/4\}$.

Definition 4.1 (Normal Function Table) *Given the algebra $\mathcal{FA}$ we define a normal function table by:*

1. *A header H^i is a set $H^i = \{h^i_1, h^i_2, \cdots h^i_{l_i}\}$, where the h^i_k are conditions.*
2. *A grid G is the indexed set of functions of $\mathcal{FA}$, such that integers $i_1, i_2, \cdots, i_n$ with $0 \leq i_k \leq l_k$, give a unique index to G identifying the unique function $G_{i_1,i_2,\cdots,i_n}$.*
3. *A Normal Function table $T = (G, H^1, H^2, \cdots, H^l)$ consists of headers $H^i = (h^i_1, h^i_2, \cdots, h^i_{l_i})$ and a grid $G_{i_1,i_2,\cdots,i_l}$. A sequence of conditions $h^1_{i_1} \wedge h^2_{i_2} \wedge \cdots \wedge h^l_{i_l}$ gives an index to the grid entry $G_{i_1,i_2,\cdots,i_l}$.*

An example of a normal function table with two headers H^1, H^2.

	h^1_1	h^1_2	h^1_3	h^1_4
h^2_1	$G_{1,1}$	$G_{1,2}$	$G_{1,3}$	$G_{1,4}$
h^2_2	$G_{2,1}$	$G_{2,2}$	$G_{2,3}$	$G_{2,4}$
h^2_3	$G_{3,1}$	$G_{3,2}$	$G_{3,3}$	$G_{3,4}$

Because of obvious typesetting reasons most function tables are one or two dimensional tables. Following the definitions given by D. Parnas [Parnas95] we define the semantics of a proper normal function table.
For a normal table T $\mathbf{var}(T) = \mathbf{var}(H^1, H^2, \cdots, H^n, G)$ and $\mathbf{rangevar}(T) := \mathbf{rangevar}(G) = \bigcup_{1 \leq i_k \leq l_k} \mathbf{var}(G_{i_1,\cdots,i_n})$.

Definition 4.2 (Proper Normal Function Table)
Let $T = (G, H^1, H^2, \cdots, H^l)$ be a function table,

1. *A header $H^i = (h^i_1, h^i_2, \cdots, h^i_{l_i})$ is called proper if*
   (a) *Universal: $h^i_1 \vee h^i_2 \vee \cdots \vee h^i_l = \mathbf{true}$*
   (b) *Disjoint: the h^i_k are disjoint, $h^i_k \wedge h^i_l = \mathbf{false}$, $k \neq l$*
2. *A function table is called proper if all headers are proper.*

We extend the mapping *val* to tables T. For the table $T = (G, H^1, H^2, \cdots, H^l)$ $\mathbf{var}(T, S)$ is $\tilde{T} = (\tilde{G}, \tilde{H^1}, \tilde{H^2}, \cdots, \tilde{H^l})$ with $\tilde{G}_{i_1,i_2,\cdots,i_n} = val(G_{i_1,i_2,\cdots,i_n}, S)$ and $\tilde{H^k} = (val(h^k_1, S), \cdots, val(h^i_{k_i}, S))$.
Note that for a proper normal function table T_1 for each header H^i evaluated under a constant substitution V with $\mathbf{var}(H^i) \subseteq \mathbf{rangevar}(V)$ only one of the h^k_i can be true. This enables us to define the semantics of a proper normal function table. Composing a normal function table with a constant substitution is called evaluation of the table under a constant substitution.

Definition 4.3 (Evaluation of a Proper Normal Function Table) *A proper normal function table T evaluated under the constant substitutions V, $var(T) \subseteq$ **rangevar**(V) $val(T,V)$ to the constant function $val(G^k_{i_1,\cdots,i_l}, V\}$, where*

$$val(h^1_{i_1} \wedge h^2_{i_2} \wedge \cdots \wedge h^l_{i_l}, V) = \textbf{true}.$$

This definition justifies the name normal *function* table. The composition of tables and function is examined in the next section.

Example 4.4 *We give a table with two headers. The headers contain variables $x_1 ::$ `real` and $x_2 ::$ `real`*

	$x_1 \geq 0$	$x_1 < 0$
$x_2 \geq 0$	$\{x_1 \mapsto x_1, x_2 \mapsto x_2\}$	$\{x_1 \mapsto -x_1, x_2 \mapsto x_2\}$
$x_2 < 0$	$\{x_1 \mapsto x_1, x_2 \mapsto -x_2\}$	$\{x_1 \mapsto -x_1, x_2 \mapsto -x_2\}$

This table is proper since each header is disjoint and universal.

For more examples of tables and their applications please refer to [Parnas94.1], [Parnas94.2], [Heniger80].

5 Algebra of Normal Function Tables

We extend the algebra $\mathcal{FA}$ of the functions to an algebra of normal tables by defining the composition of tables with functions and tables. We are able to define the composition of table independent of the notion of "type" or "return types" of a table. The presented algorithms are also general in the sense that they only depend on the associative of the composition of the functions the tables are build with.

Definition 5.1 (Composition) *Let T_1 and T_2 be proper normal function tables. With $T_1 \circ T_2$ we denote the composition of T_1 and T_2 a function such that:*

$$(T_1 \circ T_2) \circ V = T_1 \circ (T_2 \circ V)$$

for all constant substitutions V with **var**$(T_1, T_2) \subseteq$ **rangevar**(V).

The composition of a normal function table $T = (H^1, H^2, \cdots, H^n, G)$ with a function F $T \circ F$ results in the table $\bar{T} = (\bar{H}^1, \cdots, \bar{H}^n, \bar{G})$ where $\bar{H}^i = (h^i_1 \circ F, \cdots, h^i_{l_i} \circ F)$ and $\bar{G}_{i_1,i_2,\cdots,i_n} = G_{i_1,i_2,\cdots,i_n} \circ F$.
The composition of a function F with a normal table $T = (H^1, H^2, \cdots, H^n, G)$ results in the table $\bar{T} = (H^1, H^2, \cdots, H^n, \bar{G})$ where $\bar{G}_{i_1,i_2,\cdots,i_n} = F \circ G_{i_1,i_2,\cdots,i_n}$.

Example 5.2

$$F = \{y \mapsto 2\ x\}, G = \begin{array}{|c|c|}\hline y < x & y \geq x \\ \hline \{x \mapsto x - y\} & \{x \mapsto y - x\} \\ \hline \end{array}$$

$$G \circ F = \begin{array}{|c|c|}\hline 2\ x < x & 2\ x \geq x \\ \hline \{x \mapsto x - 2\ x, y \mapsto 2\ x\} & \{x \mapsto 2\ x - x, y \mapsto 2\ x\} \\ \hline \end{array}$$

this can be simplified to:

$x < 0$	$x \geq 0$
$\{x \mapsto -x, y \mapsto 2\ x\}$	$\{x \mapsto x, y \mapsto 2\ x\}$

Theorem 5.3 (Closed Algebra) *The extended algebra of the proper normal function tables over an algebra $\mathcal{FA}$ is closed under composition. There exist algorithms to compute the normal function table resulting from the composition of two normal function tables.*

We will give an algorithm to transform the composition of normal function tables to a single normal function table. The next examples illustrate the ideas used for the construction of the composed tables.

Example 5.4 *We define the tables T_1 and T_2 with $x :: \texttt{real}$ and the finite set variables $m :: \texttt{fin}\{A, B\}$.*

$$T_1 = \begin{array}{|c|c|}\hline x < 0 & x \geq 0 \\ \hline \{x \mapsto -x\} & \{x \mapsto 2x\} \\ \hline \end{array} \quad T_2 := \begin{array}{|c|c|}\hline m = A & m = B \\ \hline \{x \mapsto x + 1\} & \{x \mapsto x + 2\} \\ \hline \end{array}$$

We construct the tables $T_1 \circ T_2$ and $T_2 \circ T_1$. Note that $\mathbf{var}(H_2) \cap \mathbf{rangevar}(T_1) = \{\}$*, meaning the headers of T_2 do not depend on the "output" of T_1 for this case we give a special construction for the table $T_2 \circ T_1$.*

$$T_1 \circ T_2 = \begin{array}{|c|c|}\hline m = A \wedge x + 1 < 0 & \{x \mapsto -x - 1\} \\ \hline m = A \wedge x + 1 \geq 0 & \{x \mapsto 2x + 2\} \\ \hline m = B \wedge x + 2 < 0 & \{x \mapsto -x - 2\} \\ \hline m = B \wedge x + 2 \geq 0 & \{x \mapsto 2x + 4\} \\ \hline \end{array}$$

$$T_2 \circ T_1 = \begin{array}{|c||c|c|}\hline & x < 0 & x \geq 0 \\ \hline\hline m = A & \{x \mapsto -x + 1\} & \{x \mapsto 2x + 1\} \\ \hline m = B & \{x \mapsto -x + 2\} & \{x \mapsto 2x + 2\} \\ \hline \end{array}$$

Proof We construct the composition of the two proper normal tables $T = (G, H^1, \cdots H^k)$ and $\bar{T} = (\bar{G}, \bar{H}^1, \cdots \bar{H}^r)$. We distinguish two cases depending if $var(H^1, H^2, \cdots, H^n) \cap \mathbf{rangevar}(\bar{T}) = \{\}$ or not.

Disjoint Composition $var(H^1, H^2, \cdots, H^n) \cap \mathbf{rangevar}(\bar{T}) = \{\}$

We show by construction that there exists a proper normal function table $\tilde{T} = (\tilde{G}, \tilde{H}^1, \cdots \tilde{H}^{k+r})$ such that

$$\tilde{T} \simeq T \circ \bar{T}.$$

The headers of $\tilde{T}$ are the union of the headers of T and $\bar{T}$.

$$\tilde{T} = (\tilde{G}, H^1, H^2, \cdots, H^k, \bar{H}^1, \cdots \bar{H}^r)$$

Clearly the headers of $\tilde{T}$ are proper since the headers of T and $\bar{T}$ are proper.

The grid of the table $\tilde{T}$ has the entries

$$\tilde{G}_{i_1,\cdots i_k,i_{k+1},\cdots i_{k+r}} := G_{i_1,\cdots i_k} \circ \bar{G}_{i_{k+1},\cdots i_{k+r}}.$$

The construction is completed.

It is left to show that $\tilde{T} \simeq T \circ \bar{T}$. For any constant function V with $\mathbf{var}(T, \bar{T}) \subseteq \mathbf{rangevar}(V)$

$$\tilde{T} \circ V = G_{i_1,\cdots i_k,i_{k+1},\cdots i_{k+r}} \circ V$$

where $val(h^1_{i_1} \wedge \cdots \wedge h^k_{i_k} \wedge \bar{h}^1_{i_2} \cdots \bar{h}^r_{i_2}, V) = \mathbf{true}$

$$= (G_{i_1,i_2,\cdots,i_k} \circ \bar{G}_{i_{k+1},i_{k+2},\cdots,i_{k+r}}) \circ V = G_{i_1,i_2,\cdots,i_k} \circ (\bar{G}_{i_{k+1},i_{k+2}} \circ V)$$

Composition Since now the headers of T depend on the "output" of $\bar{T}$ $var(H^1, H^2, \cdots, H^n) \cap \mathbf{rangevar}(\bar{T}) \neq \{\}$, we have to choose a different construction for the table $\tilde{T} \simeq T \circ \bar{T}$.

We construct the table $\tilde{T} = (\tilde{G}, \tilde{H})$ a table with one header $\tilde{H}$

The condition of the header $\tilde{H}$ are constructed by

$$\tilde{h}_k = \bar{h}^1_{i_1} \wedge \bar{h}^2_{i_2} \wedge \cdots \wedge \bar{h}^k_{i_k} \wedge h^1_{p_1} \circ \bar{G}^k_{i_1,i_2,\cdots,i_k} \wedge h^2_{p_2} \circ \bar{G}^k_{i_1,i_2,\cdots,i_k} \wedge \cdots \wedge h^l_{p_l} \circ \bar{G}^k_{i_1,i_2,\cdots,i_k}$$

This formula consists of all combinations of one condition of every header of table $\bar{T}$ and one conditions of every header of table T composed with the grid entry of table $\bar{T}$ indexed by the sequence $\bar{h}^1_{i_k}$.

The grid entries of $\tilde{T}$ are formed by composition of the grid entries of T and $\bar{T}$. The grid entry corresponding to the above condition h_k is

$$\tilde{G}_k = G_{p_1,\cdots,p_l} \circ \bar{G}_{i_1,\cdots,i_k}.$$

The number of grid entries is the product of the number of grid entries of the two original tables.

We are left to show that the constructed table $\tilde{T}$ represents the composition

$$\tilde{T} \simeq T \circ \bar{T}.$$

For any constant function V with $var(T, \bar{T}) \subseteq \mathbf{rangevar}(V)$

$$\tilde{T} \circ V = \tilde{G}_k \circ V$$

where

$$val(\tilde{h}_k, V) = val(\bar{h}^1_{i_1} \wedge \bar{h}^2_{i_2} \wedge \cdots \wedge \bar{h}^k_{i_k} \wedge h^1_{p_1} \circ \bar{G}^k_{i_1,i_2,\cdots,i_k} \wedge h^2_{p_2} \circ \bar{G}^k_{i_1,i_2,\cdots,i_k} \wedge \cdots \wedge h^l_{p_l} \circ \bar{G}^k_{i_1,i_2,\cdots,i_k}, V)$$

is true.

$$= (G_{i_1,i_2,\cdots,i_k} \circ \bar{G}_{i_1,i_2,\cdots,i_k}) \circ V = G_{i_1,i_2,\cdots,i_k} \circ (\bar{G}_{i_{k+1},i_{k+2}} \circ V).$$

The resulting table $\tilde{T}$ is proper since T and $\bar{T}$ where proper and the construction preserves properness.

∎

Naturally we ask the question why the composition of two normal function tables of any dimension results in one one-dimensional table in the general case. Clearly in some special cases we could find a different representation of the table representing the composition. But note that our construction produces tables with the minimum number of cells in the sense that if table T_1 has n cells in it's grid and T_2 has m cells in it's grid then the composition has $n \times m$ entries, all possible combinations. The presented algorithms produce tables with exactly this amount of cells.
Note that the proof does not depend on the definition of the composition of the $\mathcal{FA}$ functions, just on the associativity of the composition.
Finally, we can "pull" the semi-group structure of the functions $\mathcal{FA}$ to the composition of tables.

Lemma 5.5 *The structure build with the functions of $\mathcal{FA}$ and the tables from a semi-group under composition.*

Proof We can easily verify that for the two presented algorithm the following identity holds:

$$(T_1 \circ (T_2 \circ T_2)) \circ V = ((T_1 \circ T_2) \circ T_2) \circ V$$

by using the associative of FA. The one element of this semi-group is $I := \{\}$.

∎

Example 5.6 *We given another example how to use the composition of tables* `in_open` :: `bool`, `out_open` :: `bool`, `out` :: `bool`

$$T_1 := \{\texttt{out} := \texttt{in\_open} \vee \neg\texttt{out\_open}\}$$

$T_2 :=$

$\texttt{meter} > 0$	$\texttt{meter} \leq 0$
$\{\texttt{in\_open} \mapsto \textbf{true}\}$	$\{\texttt{in\_open} \mapsto \textbf{false}\}$

$T_3 :=$

$\texttt{flow} > 0$	$\texttt{flow} \leq 0$
$\{\texttt{out\_open} \mapsto \textbf{true}\}$	$\{\texttt{out\_open} \mapsto \textbf{false}\}$

$T_1 \circ T_2 \circ T_3 =$

	$\texttt{meter} > 0$	$\texttt{meter} \leq 0$
$\texttt{flow} > 0$	$\{\texttt{out} \mapsto \textbf{true} \vee \neg\textbf{true}\}$	$\{\texttt{out} \mapsto \textbf{false} \vee \neg\textbf{true}\}$
$\texttt{flow} \leq 0$	$\{\texttt{out} \mapsto \textbf{true} \vee \neg\textbf{false}\}$	$\{\texttt{out} \mapsto \textbf{false} \vee \neg\textbf{false}\}$

$=$

	$\texttt{meter} > 0$	$\texttt{meter} \leq 0$
$\texttt{flow} > 0$	$\{\texttt{out} \mapsto \textbf{true}\}$	$\{\texttt{out} \mapsto \textbf{false}\}$
$\texttt{flow} \leq 0$	$\{\texttt{out} \mapsto \textbf{true}\}$	$\{\texttt{out} \mapsto \textbf{true}\}$

6 Future Work

- We are developing the notion of a canonical form for tables. A canonical from allows to decide semantical equality of tables.
- We have a project concerning the simplification of tables. Clearly the key word is computer algebra and normal forms. As part of the "Tools for software engineering" project at the McMaster software engineering group we developed a link of our "table holder" to the computer algebra system Maple in order to perform simplifications of tables.
- As we are able to represent the composition of two tables as a single table again it is natural to ask the same question concerning iteration and recursion.

References

[Bojanowski] .Bojanowski, M. Iglewski, J. Madey, A. Obaid, Functional approach to Protoclols Specifications Proceedings of the 14th International IFIP Symposium of Protocol Specification, PSTV'94, Vancouver, June 94, pp. 371-783

[Parnas95] D. L. Parnas Tabular Representation of Relations *CRL Report No. 260 1995*

[Parnas94.1] D. L. Parnas Mathematical Description and Specification of Software, Proceedings of IFIP World, Congress 1994, Volume1 pp.270-277.

[Parnas94.2] nspectiong safty critical software using function tables, *Proceedings of IFIP World Congress 1994, Volume I*

[Mohrenschildt96] . v. Mohrenschildt Normal Forms for Function Fields Containing Piecewise Functions, *Technical Report no. 96-14 University of Waterloo*

[Mohrenschildt94] . Engeler, M. v. Mohrenschildt The combinatory Program *Birkhaeuser 1994*

[Zucker94] J. Zucker Transformations of Normal and Inverted Function Tables to be publishes in Formal Aspects of Computing, and *CRL Report NO. 291, 1994*

[Zucker93] J. Zucker Normal and Inverted Function Tables *CRL Report NO. 265 1993*

[Heniger80] .L. Heniger Specifying software requierments for complex systems *IEEE Transactions on Software Engineering SE-6, 2-13*

[Janicki95.1] R. Janicki Towards a Formal Semantic of Parnas Tables *McMaster MC-SERG Technical Report 95-03, 1995*

[Janicki95.2] R. Janicki, D. L. Parnas, J. Zucker Tabular Representations in Rational Documents *CRL Report No. 313 1995*, to appear in

[Sperschneider] V. Sperschneider, G. Antoniou Logic, A Foundation for Computer Science *International Computer Science Series*

[Colby] C. Colby (1996) Semantics-based Program Analysis via Symbolic Composition of Transfer Relations *Carnegie Mellon University CMU-CS-96-162.*