On a Formal Semantics of Tabular Expressions

Ryszard Janicki*
Ridha Khedri
Department of Computing and Software
McMaster University
Hamilton, Ontario, Canada L8S 4K1
{janicki,khedri}@mcmaster.ca

Abstract

In [24, 35, 38, 39] Parnas et al. advocate the use of relational model for documenting the intended behaviour of programs. In this method, *tabular expressions* (or *tables*) are used to improve readability so that *formal* documentation can replace conventional documentation. Parnas [36] describes several classes of tables and provides their *formal* syntax and semantics. In this paper, an alternative, more general and more homogeneous semantics is proposed. The model covers all known types of tables used in Software Engineering.

1 Introduction

Software has become critically important, not only in the software industry, computer industry, and information industries, but in all areas of modern technology. In all software applications, the documentation is important in both the initial development and the maintenance period that follows. Documentation is used in design reviews, to guide the programmers, to guide the users and to save cost when the software has to be extended or modified. One may observe that the inability of computer systems developers to provide precise and systematic documentation is major cause of expense and unreliability. Even small computer systems can be very complex. In other engineering fields, mathematical formulas are used to document the properties of products and their components.

However, in the classical engineering fields, as well as in mathematics, the formulas are seldom longer than a dozen and so lines. In software engineering, the formulas

*Partially supported by NSERC of Canada Grant

are often much longer. For example, an invariant of a concurrent algorithm can occupy more than one page, and the specification of a real system can be a formula dozens or more pages long.

Standard mathematical notation works well for short formulas, but not for long ones. One way to deal with long formulas is to use some form of module structure and hierarchical structuring. The paper [26] is an excellent example of this approach. However hierarchical structuring and modularity alone *are not sufficient*. The problem is that the standard mathematical notation is, in principle, *linear*. This makes it poorly readable when many cases have to be considered, when functions have many irregular discontinuities, or when the domain and range of functions are built from the elements of different types. It turns out that using tables helps to make mathematics more practical for computing systems applications [24].

Tabular notation for computer programs and modules made their appearence in the late 1950s. The General Electric Company [7], and the U.S. Air Force at Norton Air Force Base apperently played a large role in the inauduration of their use [29, 31]. The concept of using tables for software first appeared in the literature near the start of 1960s (see [6, 11, 21, 28, 32]). The form and names given to the tables also varied a lot. The designation that soon prevailed was *decision tables*. These tables are two-dimentional tables. In this paper we are considering others alternative kind of table which could be multi-dimensional. The most of (but not all) decision tables [19, 20, 36] are special case of one of these type of table (input-vector type). The *multi-dimensional tabular notation* makes it easier to consider every case separately while writing or reading a design document.

The key ideas of a *tabular notation*, one of the cornerstones of the relational model for documenting the intended behaviour of programs [24, 35, 38, 39], were first developed in work for the U.S. Navy and applied to the A-7E aircraft [9, 15, 16, 42]. The ideas were picked up by Grumman, the U.S. Air Force, Bell Laboratories and many others. Recently the tabular notations have been applied by Ontario Hydro in Darlington Nuclear Plant [4, 33, 34].

The industrial applications mentioned above were conducted on, more or less, an *ad hoc* basis, i.e. without formal syntax and semantics (new types of tables were invented according to the needs, the semantics was intuitive one, in particular the inverted tables were 'discovered' almost by mistake [37]).

The papers [38, 39] show in a formal way how the documentation required for the design and use of computing systems can consist of descriptions of a set of relations. Those relations are represented by *multi-dimensional* expressions called *tables*. Parnas [36] describes several different classes of tables and provides their *formal* syntax and semantics. All classes considered in [36] were invented for some specific practical applications. Formal relationship between some important classes of tables has been analyzed in [45]. The overall methodology and recent results of the tabular approach are presented in [24].

	(%Power% = on) ∧ (%%Shutdown%% = $operate$) ∧ (%%Watchdog%% = $operate$) ∧ (@T(%Reset% = $released$))	(%Power% = on) ∧ (%%Shutdown%% = $operate$) ∧ (%%Watchdog%% = $operate$) ∧ (@T(%Reset% = $pressed$))	(%Power% = off) ∨ (%%Shutdown%% = $shutdown$) ∨ (%%Watchdog%% = $shutdown$) ∨ X	(%Power% = off) ∨ (%%Shutdown%% = $shutdown$) ∨ (%%Watchdog%% = $shutdown$) ∨ Enter
%%PumpSwitch%% =	$closed$	$open$	$closed$	$open$

Figure 1: A part of the Software Requirements for the Water Level Monitoring System of The A-7 aircraft

The tabular notation is currently used among others by the Software Engineering Research Group (SERG) at McMaster University, Hamilton, Ontario, Canada [43], Ontario Hydro [30], Naval Research Laboratory [14], ORA Inc., [19], and University of California at Irvine [13, 27].

In this paper we propose a more general and more homogeneous approach. Instead of many different classes of tables and separate semantics in each case (as in [36]), we shall introduce only one general definition of tables, each class of [36] could be derived as a special case. The model will also indicate the other, not considered in [36], classes of tables that could be constructed in the general framework. The central concept in our approach is so-called *cell connection graph* which characterizes *information flow* (*'where do I start reading the table and where do I get my result?'*) of a given table. The model presented in this paper covers all the known types of tables used in the Software Industry (compare [1]).

All examples of tables used in this paper are very simple on purpose. In actual practice, the specifications, or the requirements for a software system are presented with simple tables. For instance, the software requirements for the water level monitoring system of A-7 aircraft is written as some small tables like the table in Figure 1. This table is borrowed from [42] the notation used in it is introduced in the A-7 document [16].

For more realistic examples (as loop invariants, program specifications) the reader is referenced to [1, 39, 43].

The key assumptions behind the idea of tabular expressions are:

- the intended behaviour of programs is modelled by a (usually complex) relation, say R.

- the relation R may itself be complex but it can be built from a collection of relations R_α, $\alpha \in I$, where I is a set of indices, each R_α can be specified rather easily. In most cases R_α can be defined by a simple linear formula that can be held in few cells of a table. Some cells define the domain of R_α, the others R_α itself.

- the tabular expression that describes R is a structured collection of cells containing definitions of R_α's. The structure of a tabular expression informs how the relation R can be composed of all the R_α's.

The paper [36] provided a major motivation for this work. The early results have been presented in [22]. The paper is a revised version of [23].

We assume that the reader is familiar with such concepts as function, relation, Cartesian product, etc. [12, 40]. The standard mathematical notation is used throughout the paper.

In Section 2, we introduce tabular expressions of relations, present six "topologically" different types of cell connection graphs, and give the definition of tabular expression (or table) as 6-tuple. In section 3, we elaborate on two components of this 6-tuple: the *table predicate rule* and the *table relation rule*. In section 4, we show how to compose the relation specified by a tabular expression from the relations described in appropriate guard and value cells. We also ahow how our approach is related to the standard Relation Algebra. Section 5 contains a final comment.

2 Tabular Expressions

Intuitively, a table is an *organized collection of sets of cells, each cell contains an appropriate expression*. Such an organized collection of *empty cells*, without expressions, will be called a *table skeleton*. We assume that a *cell* is a primitive concept which does not need to be explained.

- A *header* H is an indexed set of cells, $H = \{h_i \mid i \in I\}$, where $I = \{1, 2, ..., k\}$, some k, is a set of indexes.

- A *grid* G *indexed by headers* $H_1, ..., H_n$, with $H_j = \{h_i^j \mid i \in I^j\}$, $j = 1, ..., n$ is an indexed set of cells G, where $G = \{g_\alpha \mid \alpha \in I\}$, and $I = \prod_{i=1}^n I^i$ (or $I = I^1 \times ... \times I^n$). The set I is the *index of* G.

A collection of headers $H_1, ..., H_n$ and a grid G indexed by them can be regarded as a first approximation of table skeleton.

The elements of the set $Components = \{H_1, ..., H_n, G\}$ are called *table components*.

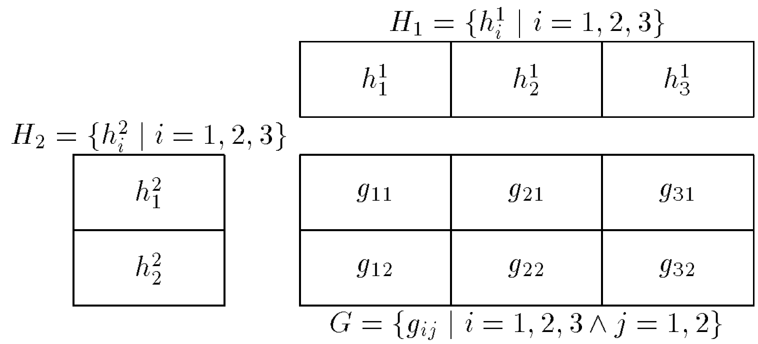


Figure 2: An example of headers H_1, H_2, $I^1 = \{1, 2, 3\}$, $I^2 = \{1, 2\}$, and grid G.

Figure 2 illustrates the above definitions.

A table is intended to represent a relation R. The relation R is composed from R_α's, $\alpha \in I$, i.e. $R = \heartsuit_{\alpha \in I} R_\alpha$. The various types of operation $\heartsuit$ will be discussed in the Section 4.

The assumption is that every R_α is fully specified by some expressions held in *one grid cell* g_α, and *header cells* $h^j_{\alpha|j} \in H_j$, where $\alpha|j$ is the *jth* coordinate of α (i.e. if $\alpha = (j_1, ..., j_n)$, then $\alpha|3 = j_3$) for $j = 1, ..., n$.

For every $\alpha \in I$ we define

$$Components_\alpha = \{h^1_{\alpha|1}, ..., h^n_{\alpha|n}, g_\alpha\}.$$

In the case of Figure 2, we have $Components_{32} = \{h_3^1, h_2^2, g_{32}\}$, $R = \heartsuit_{i=1,2,3}^{j=1,2} R_{i,j}$, R_{22} is defined by the expressions held in g_{22}, h_2^1, h_2^2, while R_{32} is defined by the expressions held in g_{32}, h_3^1, h_2^2, etc.

We assume that every relation R_α, $\alpha \in I$ is specified by an expression of the form[1]".

$$\textbf{if } P_\alpha \textbf{ then } E_\alpha$$

where P_α is the predicate that defines the domain of R_α and E_α is the predicate that defines the values of R_α. For example **if** $x_1 < 0 \wedge x_2 < 0$ **then** $y^2 = x_1^2 + x_2^2$, or **if** $-1 \leq x < 0$ **then** $y_2^2 + x = 0 \vee y_1 = 1$ (see Figures 3).

[1]The predicate **if** P_α **then** E_α can equivalently be written as $P_\alpha \wedge E_\alpha$. We shall prefer **if-then** form because it is more readable, in particular when P_α itself contains "$\wedge$" operator (see Figure 6). But clearly **if** P_α **then** $E_\alpha = P_\alpha \wedge E_\alpha$. Do not confuse "**if** P **then** E" with "$P \Rightarrow E$

$$R_{ij} = \textbf{if } x_1 < 0 \wedge x_2 < 0 \textbf{ then } y^2 = x_1^2 + x_2^2$$

$$\begin{array}{cccc} & & \boxed{x_1 < 0} & h_i^1 \\ h_j^2 & \boxed{x_2 < 0} & \boxed{\boxed{y^2 = x_1^2 + x_2^2}} & g_{ij} \end{array}$$

$$R_{ij} = \textbf{if } -1 \leq x < 0 \textbf{ then } y_2^2 + x = 0 \vee y_1 = 1$$

$$\begin{array}{cccc} & & \boxed{\boxed{y_1 = 1}} & h_i^1 \\ h_j^2 & \boxed{\boxed{y_2^2 + x = 0}} & \boxed{-1 \leq x < 0} & g_{ij} \end{array}$$

Figure 3: Two examples of placing P_α and E_α into cells. The cells containing the elements of E_α have double line borders.

The expressions P_α and E_α are built from the other expressions, all the expressions from which P_α and E_α are constructed are held in the cells $h_{i_1}^1, ..., h_{i_n}^n, g_\alpha$, where $\alpha = (i_1, ..., i_n)$ (see Figure 3).

The following two important properties are assumed:

- each cell may hold either a part of P_α or a part of E_α, but not both.
- the distribution of P_α and E_α into appropriate cells is independent of α.

In other words, each table component, a header or grid, can either hold only the elements used to define P_α's or the elements used to define E_α's.

This means we can divide $Components = \{H_1, ..., H_n, G\}$ into two sets $Guards$ and $Values$, such that

$Guards \neq \emptyset$, $Values \neq \emptyset$, $Components = Guards \cap Values$, $Guards \cap Values = \emptyset$.

We also define $Guards_\alpha = Guards \cap Components_\alpha$, and $Values_\alpha = Values \cap Components_\alpha$, $\alpha \in I$.
The $Guards_\alpha$ contains elements of P_α, $Values_\alpha$ contains elements of E_α. There is only one grid G, so it may belong to either $Values$ or $Guards$.

The definition of $Guards$ and $Values$ enables us to introduce the concept of a *cell connection graph*[2]

The *cell connection graph* characterizes information flow (*"where do I start reading the table and where do I get my result?"*). It is a relation that could be interpreted as an *acyclic directed graph* with the grid and all headers as the nodes.

Let $\longmapsto$ be a relation $\longmapsto \subseteq Components \times Components$ satisfying:

$$\forall A, B \in Components \;\; A \longmapsto B \;\Rightarrow\; ((A = G \vee B = G) \wedge A \neq B). \tag{1}$$

In other words, each arc that represents $\longmapsto$ *must either start from or end at the grid G.*

The relation $\longmapsto^*$, transitive and reflexive closure of $\longmapsto$, is a *partial order* [12], so we can talk about both maximal and minimal elements w.r.t. $\longmapsto^*$.

The relation $\longmapsto$ is a *cell connection graph* if

1. A is maximal w.r.t. $\longmapsto^* \implies A \in Values$,
2. A is minimal w.r.t. $\longmapsto^* \implies A \in Guards$,
3. $\forall A \in Guards(T).\forall B \in Values(T).\;\; A \longmapsto^+ B$.

The cell connection graph $\longmapsto$ represents *information flow* among table cells and, intuitively, if the component A is built from the cells describing the domain of a relation/function specified, and the component B is built from the cells that describe how to calculate the values of the relation/function specified, than we expect $A \longmapsto^+ B$, where $\longmapsto^+$ is the transitive closure of $\longmapsto$. This means that *the components built from the cell describing the domains are never maximal*, while *the components built from the cells containing formulae for values are never minimal.*

One can also easily prove the following Lemma.

Lemma 2.1

Only the grid G can be neutral, and there exists at most one neutral component. ∎

There are six "topologically" different types of cell connection graphs.

[2] In earlier papers [1, 22, 23], the cell connection graph was introduced first and the partition of $Components$ later.

Type 1. Each element is either maximal or minimal. There is only one maximal element.

Type 2a. There is only one maximal element and one neutral element. The neutral element belongs to $Guards$.

Type 2b. There is only one maximal element and one neutral element. The neutral element belongs to $Values$.

Type 3a. There is a neutral element and more than one maximal element. The neutral element belongs to $Guards$.

Type 3b. There is a neutral element and more than one maximal element. The neutral element belongs to $Values$.

Type 4. Each element is either maximal or minimal. There is only one minimal element.

The division into types 1, 2, 3 and 4 is based on the shape of the relation $\longmapsto$, the types a and b result from different decompositions into $Guards$ and $Values$. Figure 4 illustrate all cases for $n = 3$. When the number of headers is smaller than 3, the cases 3a and 3b disappear.

It turns out that:

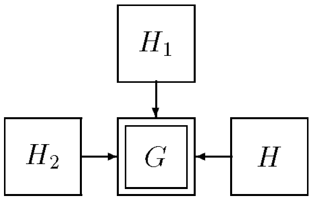
- Normal Tables of [36] are of Type1,
- Inverted, Decision and Generalized Decision Tables [19, 36] belong to Type 2a,
- type 2b Vector Tables of [36] are of Type 2b.

The types 3a, 3b and 4 have no known wide application yet. They seem to be useful when some degree of non-determinism is allowed. The types 3a and 3b might also be useful as a representation of complex vector tables. The paper [1] provides an excellent survey of all type of tables used in Software Engineering practice.

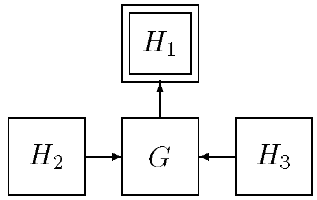
The type of Cell Connection Graph will usually be identified by a small icon resembling an appropriate graph from Figure 4. The icon is placed in left upper corner of the table. Table components belonging to $Values$ have double borders. Figure 5 illustrates the concepts discussed above.
The triple

$$TSK = (Components, Guards, Values)$$

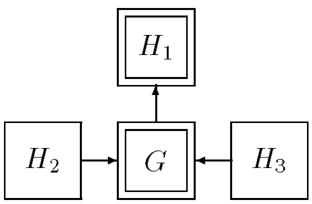
will be called a *Table Skeleton*. A table skeleton represents the structure of a tabular expression that is independent of the particular values of R_α's. To define tabular



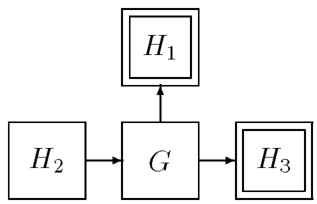
Type 1. Each element is either maximal or minimal. There is only one maximal element.



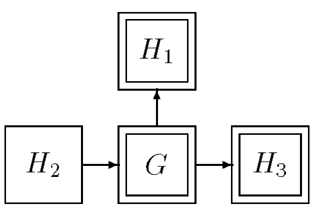
Type 2a. There is only one maximal element and a neutral element. The neutral element belongs to Guards.



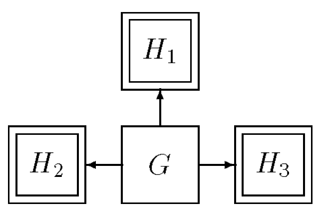
Type 2b. There is only one maximal element and a neutral element. The neutral element belongs to Values.



Type 3a. There is a neutral element and more than one maximal element. The neutral element belongs to Guards.

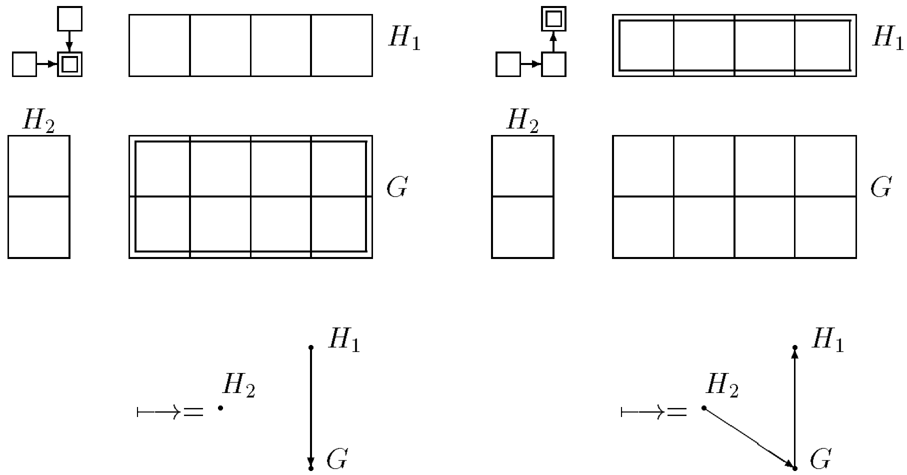


Type 3b. There is a neutral element and more than one maximal element. The neutral element belongs to Values.



Type 4. Each element is either maximal or minimal. There is only one minimal element

Figure 4: Six different types of cell connection graphs ($n = 3$).



$Guards = \{H_1, H_2\}$
$Values = \{G\}$

$Guards = \{H_2, G\}$
$Values = \{H_1\}$

Figure 5: Two examples of $Guards$, $Values$ and $\longmapsto$

expressions completely we have to precisely describe how particular cells are filled, how P_α and E_α should be constructed from the contents of appropriate cells.
Recall the idea we were using is the following:

- the expressions defining the *relational expression's* E_α's are held in value cells $Values$.
- the expressions defining the *predicate expression's* P_α's are held in guard cells $Guards$.

However, the partition of cells into value and guard types is not sufficient. Let us consider the examples in Figure 3. The top one is intended to corresponded to the expression **if** $x_1 < 0 \wedge x_2 < 0$ **then** $y^2 = x_1^2 + x_2^2$. But *why* we write $x_1 < 0 \wedge x_2 < 0$? Why not for example: $x_1 < 0 \vee x_2 < 0$, or $\neg(x_1 < 0) \wedge x_2 < 0$ etc.? The bottom one is intended to correspond to the expression **if** $-1 \leq x < 0$ **then** $y_1 = 1 \vee y_2^2 + x = 0$, or, using slightly different notation, $R_{ij} = Q_{ij} \cup S_{ij}$, where $Q_{ij} =$ **if** $-1 \leq x < 0$ **then** $y_1 = 1$ and $S_{ij} =$ **if** $-1 \leq x < 0$ **then** $y_2^2 + x = 0$. Again, why we write $y_1 = 1 \vee y_2^2 + x = 0$, or why we use $R_{ij} = Q_{ij} \cup S_{ij}$?

A table skeleton does not provide any information on how the domain and values of the relation (function) specified are determined; such information must be added.

Let $TSK = (Components, Guards, Values)$ be a table skeleton. Assume that $Guards = \{B_1, ..., B_r\}$, $Values = \{A_1, ..., A_s\}$.

- A predicate expression $PR(B_1, ..., B_r)$, where $B_1, ..., B_r$ are variables, is called a *table predicate rule*.

- A relation expression $RR(A_1, ..., A_s)$, where $A_1, ..., A_s$ are variables, is called *table relation rule.*

The predicate P_α, $\alpha \in I$ can now we derived from $PR(B_1, ..., B_s)$ by replacing each variable B_i by the content of the cell that belongs to $\{B_i\} \cap Guards_\alpha$. Similarly, the relation expression E_α can now we derived from $RR(A_1, ..., A_s)$ by replacing each variable A_i by the content of the cell that belongs to $\{A_i\} \cap Values_\alpha$.

More detailed forms of table predicate rules and table relation rules are discussed in the Section 4. The table predicate and relation rules are sufficient to understand how the expressions **if** P_α **then** E_α can be built from the contents of appropriate cells. We still do not know how the relation R should be built from all R_α's.

- A relation expression CR of the form $R = \heartsuit_{\alpha \in I} R_\alpha$ is called a *table composition rule.*

In general, $\heartsuit_{\alpha \in I} R_\alpha$ is a relational expression builts from the expressions defining R_α's, and various relational operators. We shall discuss it in detail in Section 4.

We can now define formally the concept of a tabular expression:

- A *tabular expression* (or *table*) is a tuple

  $$T = (TSK, PR, RR, CR, \mathbf{IN}, \mathbf{OUT})$$

  where TSK is a table skeleton, PR, RR, CR are respectively table predicate rule, table relation rule, and table composition rule, Ψ is a mapping which assigns a predicate expression, or part of it, to each guard cell, and a relation expression, or part of it, to each value cell. The predicate expressions have variables over $\mathbf{IN}$, the relation expression have variables over $\mathbf{IN} \times \mathbf{OUT}$, where $\mathbf{IN}$ is the set (usually heterogenous Cartesion product) of inputs, and $\mathbf{OUT}$ is the set (usually heterogenous Cartesia product) of outputs.

For every tabular expression T, we define the *signature* of T as:

$$Sign_T = (PR, RR, CR, \longmapsto).$$

The signature describes all the global and structural information about the table. We may say that a tabular expression is a triple: *signature*, *skeleton* - which describes the number of elements in headers and indexing of the grid, and the mapping Ψ - which describes the content of all cells.

Examples of tables are presented in Figures 6, 7 and 8. The signatures enriched by information about variables are presented separately in special two column tables. The above definitions describes, more or less, the *syntax* of tables. However the word 'syntax' here has the meaning closer to that used in Linguistics than in Mathematics and Computer Science. In general Ψ may assign *predicate expression*, or part of it, to guard cells, and *relation expression*, or part of it, to value cells. We do not assume much about Ψ.

Let I be the index of T, let

$$\begin{array}{l} P_\alpha = PR[\Psi(c_1)/B_1, ..., \Psi(c_s)/B_s] \\ E_\alpha = RR[\Psi(d_1)/A_1, ..., \Psi(d_r)/A_r] \end{array} \tag{2}$$

where $c_i = B_i \cap Guards_\alpha$, $i = 1, ..., s$, and $d_i = A_i \cap Values_\alpha$, $i = 1, ..., r$, $PR[\Psi(c_1)/B_1, ..., \Psi(c_s)/B_s]$ is obtained from PR by replacing each B_i by $\Psi(c_i)$, and simiraly for $RR[...]$.

Both PR and RR must satisfy the following *consistency* rule

- for every $\alpha \in I$, PP_α is a syntactically correct predicate expression.
- for every $\alpha \in I$, RR_α is a syntactically correct relation expression.

The relation composition expression CR is built from the relation/function names, indexes, and relational operators (see Section 4).

- The *semantics of a tabular expression* T can now be defined as a *relation*

$$R_T = CR(R_\alpha),$$

where $R_\alpha =$ **if** P_α **then** E_α.

Figures 6, 7 and 8 illustrate the above definitions.

3 Table Predicate Rules and Table Relation Rules

The predicate expression PR is built from table component names (variables) $B_1, ..., B_r$, where $Guards = \{B_1, ..., B_r\}$, logical operators "$\wedge$", "$\vee$", "$\neg$" (however "$\neg$" is at present disallowed for implementation reasons in the SERG tool package [1, 43]), the replacement operator, some constant and relation symbols. The *replacement operator* is of the form $E[E_1/x]$, where E, E_1 are expressions, x is a variable or constant, and $E[E_1/x]$ represents a new expression derived from E by replacing every

$$f(x,y) = \begin{cases} 0 & \text{if } x \geq 0 \wedge y = 10 \\ x & \text{if } x < 0 \wedge y = 10 \\ y^2 & \text{if } x \geq 0 \wedge y > 10 \\ -y^2 & \text{if } x \geq 0 \wedge y < 10 \\ x+y & \text{if } x < 0 \wedge y > 10 \\ x-y & \text{if } x < 0 \wedge y < 10 \end{cases}$$

input variables	$x, y : Reals$
output variables	$f : Reals$
$\longmapsto$	
PR	$H_1 \wedge H_2$
RR	G
Function name	f
CR	$\bigcup_{i=1}^{3} \bigcup_{j=1}^{2} f_{i,j}$

H_2 \ H_1	$y = 10$	$y > 10$	$y < 10$
$x \geq 0$	0	y^2	$-y^2$
$x < 0$	x	$x + y$	$x - y$

G

$$g(x,y) = \begin{cases} x+y & \text{if } (x < 0 \wedge y \geq 0) \vee (x < y \wedge y < 0) \\ x-y & \text{if } (0 \leq x < y \wedge y \geq 0) \\ & \quad \vee (y \leq x < 0 \wedge y < 0) \\ y-x & \text{if } (x \geq 0 \wedge y \geq 0) \vee (x \geq 0 \wedge y < 0) \end{cases}$$

input variables	$x, y : Reals$
output variables	$g : Reals$
$\longmapsto$	
PR	$H_2 \wedge G$
RR	H_1
Function name	g
CR	$\bigcup_{i=1}^{3} \bigcup_{j=1}^{2} g_{i,j}$

H_2 \ H_1	$x + y$	$x - y$	$y - x$
$y \geq 0$	$x < 0$	$0 \leq x < y$	$x \geq y$
$y < 0$	$x < y$	$y \leq x < 0$	$x \geq o$

G

Figure 6: Two examples of tabular expressions - normal (above) and inverted (below)

$$(x_1, x_2)R(y_1, y_2, y_3) \iff \begin{cases} y_1 = x_1 + x_2 \ \wedge \ y_2 x_1 - x_2 = y_2^2 \\ \wedge \ y_3 + x_1 x_2 = |y_3|^3 \end{cases} \text{if } x_2 \leq 0 \quad \begin{cases} y_1 = x_1 - x_2 \ \wedge \ x_1 + x_2 + x_2 y_2 = |y_2| \\ \wedge \ y_3 = x_1 \end{cases} \text{if } x_2 > 0$$

input variables	$x_1, x_2 : Reals$
output variables	$y_1, y_2, y_3 : Reals$
$\longmapsto$	
PR	H_1
RR	$H_2 \circ G$
Relation name	R
CR	$\bigotimes_{j=1}^{3} \bigcup_{i=1}^{2} R_{i,j}$

H_2 (left column), H_1 (top row), G (grid):

H_2	$x_2 \leq 0$	$x_2 > 0$
$y_1 =$	$x_1 + x_2$	$x_1 - x_2$
$y_2 \vert$	$y_2 x_1 - x_2 = y_2^2$	$x_1 + x_2 + x_2 y_2 = \vert y_2 \vert$
$y_3 \vert$	$y_3 + x_1 x_2 = \vert y_3 \vert^3$	$y_3 = x_1$

The symbol "=" after y_1 in H_2 indicates that the relations $R_{i,1}$, $i = 1, 2$, are functions. The symbol "|" after y_2 and y_3 in H_2 indicates that $R_{i,2}$ and $R_{i,3}$, $i = 1, 2$ are relations with y_2 and y_3 as a respective output variables.

φ : Temperature $\times$ Weather $\times$ Windy $\rightarrow$ *Activities*, where
Activities = { go sailing, go to the beach, play bridge, garden}.

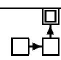
input variables	Temperature: *hot*, *cold*
	Weather: *sunny*, *cloudy*, *rain*
	Windy: *true*, *false*
output variables	φ :go sailing, go to the beach play bridge, garden
$\longmapsto$	
PR	$H_2 = G$
RR	H_1
Function name	φ
CR	$\bigcup_{i=1}^{5} \bigotimes_{j=1}^{3} \varphi_{i,j}$
notation	$* = don't\ care$

H_2 (left column), H_1 (top row), G (grid):

H_2	go sailing	go to the beach	play bridge	garden	
Temperature $\in \{hot, cool\}$	$*$	$*$	*hot*	$*$	*cool*
Weather $\in \{sunny, cloudy, rain\}$	$sunny \vee cloudy$	*sunny*	*cloudy*	*rain*	*cloudy*
Windy $\in \{true, false\}$	*true*	*false*	*false*	$*$	*false*

Figure 7: Next two examples of tabular expressions - vector table (above) and decision table [20] (below)

$$h(x_1, x_2) = \begin{cases} x_1 + x_2 & \text{if } x_1 x_2 < 20 \wedge x_1 \div x_2 > 30 \\ x_1 - x_2 & \text{if } x_1 x_2 \geq 20 \wedge x_1 \div x_2 < 30 \\ x_1 x_2 & \text{if } x_1 \div x_2 = 30 \end{cases}$$

input variables	$x_1, x_2 : Reals$
output variables	$h : Reals$
$\longmapsto$	
PR	$G[H_2/\#]$
RR	H_1
Function name	h
CR	$\bigcup_{i=1}^{3} \bigotimes_{j=1}^{2} h_{i,j}$

H_2

$x_1 x_2$
$x_1 \div x_2$

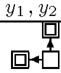
H_1

$x_1 + x_2$	$x_1 - x_2$	$x_1 x_2$

G

$\# < 20$	$\# \geq 20$	$true$
$\# < 30$	$\# > 30$	$\# = 30$

$$x \Upsilon (y_1, y_2) \iff \begin{cases} \ (x = 0 & \wedge \ \ y_1 = 0 \wedge y_2^2 + x = 1) \\ \vee \ (x < -1 & \wedge \ \ y_1 = 0 \wedge y_2^2 + x = 0) \\ \vee \ (x = 1 & \wedge \ \ y_1 = 0 \wedge y_2 = x^2) \\ \vee \ (0 < x < 1 & \wedge \ \ y_1 = 1 \wedge y_2^2 + x = 1) \\ \vee \ (-1 \leq x < 0 & \wedge \ \ y_1 = 1 \wedge y_2^2 + x = 0) \\ \vee \ (x > 1 & \wedge \ \ y_1 = 1 \wedge y_2 = x^2) \end{cases}$$

input variables	$x : Reals$
output variables	$y_1, y_2 : Reals$
$\longmapsto$	
PR	G
RR	$H_1 \otimes H_2$
Relation name	Υ
CR	$\bigcup_{j=1}^{3} \bigcup_{i=1}^{2} \Upsilon_{i,j}$

H_2

$y_2^2 + x = 1$
$y_2^2 + x = 0$
$y_2 = x^2$

H_1

$y_1 = 0$	$y_1 = 1$

G

$x = 0$	$0 < x < 1$
$x < -1$	$-1 \leq x < 0$
$x = 1$	$x > 1$

Figure 8: Another two examples of tabular expressions - generalized decision (above) and type 4 (below).

occurrence of x in E by E_1. The constants and relation symbols depend on the type of input domain of the relation specified, $dom(R)$. The relation symbol "$=$" can always be used. If the elements of $dom(R)$ are ordered, the relation symbols "$<$", "$>$" can be used[3].

The relation expression RR is built from table component names $A_1, ..., A_r$ (variables), where $Values(T) = \{A_1, ..., A_r\}$, set operators "$\cup$", "$\cap$", etc., relation operators "$=$", "$<$", "$>$", etc., the operator of *concatenation*' "$\circ$"[4].

4 Composing R from R_α

One of the fundamental assumptions behind the concept of tabular expressions is that the relation R specified by a tabular expression can be composed from the relations R_α, $\alpha \in I$, where all R_α's are described in appropriate guard and value cells i.e. $\heartsuit_{\alpha \in I}$. In this section we study some operations that can be regarded as $\heartsuit_\alpha$. We start with introducing some basic concept of the algebra of relation. The next subsection contain the classical definitions and results ([40]). The new concepts and results start from subsection 4.2, where the concept of "being a part of" and some new operations are discussed.

4.1 Basic Elements of Relation Algebra

By a (*heterogenous*) *relation* R from X to Y we mean any subset of the Cartesian product $X \times Y$, i.e. $R \subseteq X \times Y$. In this case we say that the relation R has the type $X \leftrightarrow Y$, which we write as $R : X \leftrightarrow Y$ (and we read : R has the type $X \leftrightarrow Y$). By convention, when we write $R_{X \leftrightarrow Y}$ we mean that this relation has the type $X \leftrightarrow Y$. We say that the relation $R_{X \leftrightarrow Y}$ is *homogeneous* if $X = Y$. When the context allows to identify the type or when the type is of no importance, we simply write R.

For every relation $R : X \leftrightarrow Y$, we define:

$$\begin{array}{lcl} dom(R) & = & \{x \mid \exists y \in Y.\ (x, y) \in R\}, \\ range(R) & = & \{y \mid \exists x \in X.\ (x, y) \in R\}, \end{array}$$

In this paper we assume the relations are heterogeneous in general. We shall now recall the basic components and operations of heterogenous relation algebras.

[3] The survey [1] indicates that "$\wedge$", "$\vee$", "$=$" and "$E[E_1/x]$" suffice in most cases. They are the only operators used in [1] here the most of known types of tables were analyzed and converted in the extension of the earlier version [22] of the approach presented here.

[4] For example for Figure 6 we have $((y_1 =) \circ (x_1 + x_2)) = (y_1 = x_1 + x_2)$, $((y_3|) \circ (y_3 + x_1x_2 = |y_3|^3) = (y_3 \mid y_3 + x_1x_2 = |y_3|^3)$, where $(y_3 \mid y_3 + x_1x_2 = |y_3|^3)$ means that y_3 is the (only) output variable in the expression $y_3 + x_1x_2 = |y_3|^3$.

We have five basic relational operations: supremum (union), infimum (intersection), complement, inverse and composition.

Let $P_{A \leftrightarrow B}$ and $Q_{B \leftrightarrow C}$ be two relations.

- The *supremum* (*union*) of $P_{A \leftrightarrow B}$ and $Q_{B \leftrightarrow C}$, denoted by $P_{A \leftrightarrow B} \cup Q_{B \leftrightarrow C}$, is defined as:

$$P_{A \leftrightarrow B} \cup Q_{B \leftrightarrow C} = \begin{cases} \{(x,y) \mid (x,y) \in P_{A \leftrightarrow B} \ \vee \ (x,y) \in Q_{B \leftrightarrow C}\} & \text{if } A = C \\ & \text{and } B = D; \\ \text{undefined} & \text{otherwise.} \end{cases}$$

- The *infimum* (*intersection*) of $P_{A \leftrightarrow B}$ and $Q_{B \leftrightarrow C}$, denoted by $P_{A \leftrightarrow B} \cap Q_{B \leftrightarrow C}$, is defined as:

$$P_{A \leftrightarrow B} \cap Q_{B \leftrightarrow C} = \begin{cases} \{(x,y) \mid (x,y) \in P_{A \leftrightarrow B} \ \wedge \ (x,y) \in Q_{B \leftrightarrow C}\} & \text{if } A = C \\ & \text{and } B = D; \\ \text{undefined} & \text{otherwise.} \end{cases}$$

- The *complement* of $P_{A \leftrightarrow B}$, denoted by $\overline{P}_{A \leftrightarrow B}$, is defined as:

$$\overline{P}_{A \leftrightarrow B} = \{(x,y) \mid x \in A \ \wedge \ y \in B \ \wedge \ (x,y) \notin P_{A \leftrightarrow B}\}$$

- The *converse* of relation $P_{A \leftrightarrow B}$, $P^{\smile}_{A \leftrightarrow B}$, is defined as:

$$P^{\smile}_{A \leftrightarrow B} = \{(x,z) \mid (z,x) \in P_{A \leftrightarrow B}\}.$$

- The *relational composition* of $P_{A \leftrightarrow B}$ and $Q_{C \leftrightarrow D}$, denoted by $P_{A \leftrightarrow B} ; Q_{C \leftrightarrow D}$, is defined as:

$$P_{A \leftrightarrow B} ; Q_{C \leftrightarrow D} = \begin{cases} \{(x,y) \mid \exists z \in B \cdot ((x,z) \in P_{A \leftrightarrow B} \wedge (z,y) \in Q_{C \leftrightarrow D})\} & \\ & \text{if } B = C; \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The unary operations $\overline{}$ and $\smile$ are total, while the binary operations $\cup, \cap$, and $;$ are partial. The operations supremum and infimum are just set theoretical union and intersection but restricted to the relations of the same type. We shall use the same symbol for both relational and set theoretical operations, however for relations these operations are no longer total.

We have three special kind of relations: identity, universal relation, and the empty relation.

- For every set A, the relation $\mathbb{I}_{A \leftrightarrow A} = \{(x,x) \mid x \in A\}$ is called the *identity* on A (of type $A \leftrightarrow A$).
- For every two sets A, B, the relation $\mathbb{T}_{A \leftrightarrow B} = A \times B$ is called the *universal relation* (of type $A \leftrightarrow B$).
- For every two sets A, B, the relation $\mathbb{L}_{A \leftrightarrow B} = \{(x,y) \mid \mathsf{false}\}$ is called the *empty relation* (of type $A \leftrightarrow B$).

For the usual rules of the calculus of relations see [5, 8, 40, 41].

There are many different types of relations, however in this paper we shall use only two: total relations, and functions (univalent relations).

- The relation $R_{A \leftrightarrow B}$ is *total* if $dom(R_{A \leftrightarrow B}) = A$.
- The relation $R_{A \leftrightarrow B}$ is a *function* (univalent relation) if

$$\forall x \in A. \forall y, z \in B.\ (x,y) \in R \wedge (x,z) \in R \ \Rightarrow \ y = z.$$

If f is a function we shall rather write $f : A \longrightarrow B$ instead of $f_{A \leftrightarrow B}$.

Corollary 4.1 *For each relation R :*

1. R is total $\iff \mathbb{T} = R;\mathbb{T}$;
2. R is function $\iff R^{\smile};R \subseteq \mathbb{I} \iff R;\overline{\mathbb{I}} \subseteq \overline{R}$. ■

We shall now define the concept of a restriction of a relation.

- For every set $A \subseteq X$ and every relation $R_{X \leftrightarrow Y}$ we define a relation $R|_A \subseteq X \times Y$, *restriction of* R to A, as:

$$\forall x \in X. \forall y \in Y.\ \ (x,y) \in R|_A \iff x \in A \wedge (x,y) \in R.$$

In other words, if $P_A(x)$ is a predicate that describes the set A, i.e. $x \in A \iff P_A(x)$, and $R(x,y)$ is a relational expression that defines the relation R, i.e. $(x,y) \in R \iff R(x,y)$, then the relation $R|_A$ is described by the expression $P_A(x) \wedge R(x,y)$ or **if** $P_A(x)$ **then** $R(x,y)$ (see Footnote 1) If R has a type $X \leftrightarrow Y$ then $R|_A$ has a type $A \leftrightarrow Y$. The same notation will be used for functions.

For the rest of the paper, we assume T to be an universal set of indexes and $\{D_i \mid i \in T\}$ to be an appropriate set of domains.
We shall also use the following notation:

- For every $I \subseteq T$, let $D_I = \prod_{i \in I} D_i$, where $\prod_{i \in I}$ is a *direct product* over I.

The set D_I can be seen as the set of all functions $f : I \longrightarrow \bigcup_{i \in I} D_i$ such that $\forall i \in I.\, f(i) \in D_i$.

A reader without experience in this kind of notation is referred to the Appendix which contains illustrative examples.

We will always write explicitly $f|_K$ to denote the restriction of the function f to K.

We shall now recall the concept of a *projection*.

- Let $J \subseteq I \subseteq T$. The *projection* from D_I onto D_J, denoted by ${}_I\Pi_J$, is defined as:

$$ {}_I\Pi_J = \{(f, g) \mid f \in D_I \ \wedge \ g = f|_J\} $$

Note that ${}_I\Pi_J$ is a *relation*, ${}_I\Pi_J : D_I \leftrightarrow D_J$. From this definition, it follows immediately that, ${}_T\Pi_T = \mathbb{I}_{D_T \leftrightarrow D_T}$, and ${}_I\Pi_J^{\smile} ; {}_I\Pi_J = \mathbb{I}_{D_J \leftrightarrow D_J}$ and ${}_I\Pi_J$ is total.

4.2 The Relation *Part of*

The fundamental idea behind the concept of tabular expressions is that it allows to specify, in an intuitive and relatively easy way, a complex relation or function from *parts*. It is assumed that the parts may be defined rather easily, but the whole may not. When software engineers discuss a specification using tabular expression, the statements like "this is *a part of* a bigger relation" can be heard very often.

Unfortunately, the only meaning of "being a part of", can so far be only an intuitive one, since the standard algebra of relations lacks the formal concept of being a *part of* concept[5]. The concept of subset is not enough, for instance if $A \subseteq B$ and $D = B \times C$, then A is not a subset of of D, but is obviously a *part of* D.

Intuitively, in most cases, R_α is a *part of* R.

In this subsection we give an initial attempt to define the relation "a part of" for the algebra of relations. We start with the concept of "part of" for direct prducts.

- Let I, J be subsets of T such that $I \subseteq J$, and let $A \subseteq D_I, B \subseteq D_J$. We define the relation $\sqsubseteq$ as:

$$ A \sqsubseteq B \iff \forall f \in A.\exists g \in B.f = g|_I $$

[5]The relation *part of* is the basic notion of *Lesniewski's Mereology* [44], which is a version of set theory proposed as an antinomy-free counterpart of naive Cantor set theory. Lesniewski's systems are different than the standard set theory based on Zermello-Freankel axioms. Unfortunately the formal translation of Lesniewski's ideas into the standard set theory framework is not obvious, although possible [44], and certainly beyond the scope of this paper. The relation $\sqsubseteq$ introduced in this paper roughly (and intuitively) can be seen as a special case of Lesniewski's *part of*. The relation *part of* was also a partial motivation for introduction the cylindric algebras ("*a circle is a part of a cylinder*", see [17]), but this concept never become a formal part of cylindric algebras.

We shall say that A is *part of* B is $A \sqsubseteq B$.

Clearly if $I = J$ then $A \sqsubseteq B \iff A \subseteq B$. For example, if $B \subseteq X_1 \times X_2 \times X_3 \times X_4$ and $A \subseteq X_2 \times X_4$, then $A \sqsubseteq B \iff \forall(x_2, x_4) \in A. \exists x_1 \in X_1, x_3 \in X_3.\ (x_1, x_2, x_3, x_4) \in B$.

We shall now extend the concept of *part of* to the relations.

- Let I, J, K and L be subsets of T such that $K \subseteq I$ and $L \subseteq J$. Let $P : D_K \leftrightarrow D_L$ and $Q : D_I \leftrightarrow D_J$ be two relations. We define $\sqsubseteq$ as:

$$P \sqsubseteq Q \iff \forall(f_1, f_2) \in P. \exists(g_1, g_2) \in Q. f_1 = g_1|_K \ \wedge \ f_2 = g_2|_L$$

If $P \sqsubseteq Q$ we say that P is *part of* Q. Consider the following example. We take $I = \{2, 3\}$ and $J = \{2\}$, $T = \{1, 2, 3\}$. Let $P : D_I \leftrightarrow D_J$ and let $Q : D_T \leftrightarrow D_T$ be relations such that

$$P = \Big\{\Big((\alpha, m), \beta\Big), \Big((\beta, n), \alpha\Big)\Big\}$$

$$Q = \Big\{\Big((1, \alpha, m), (2, \beta, m)\Big), \Big((2, \beta, n), (2, \alpha, n)\Big), \Big((2, \alpha, m), (2, \beta, n)\Big)\Big\}$$

We have $P \sqsubseteq Q$ since $(\alpha, m) = (1, \alpha, m)|_{\{2,3\}}, \beta = (2, \beta, m)|_{\{2\}}$, and $(\beta, n) = (2, \beta, n)|_{\{2,3\}}, \alpha = (2, \alpha, n)|_{\{2\}}$. A question one may ask is "can $\sqsubseteq$ be expressed in terms of standard relational algebra operations?". To answer this question we start with the concept of a cylindrification relation.

- Let ${}_I\delta_J$, called a *cylindrification relation*, be the following relation

$${}_I\delta_J = {}_I\Pi_J \,;\, {}_I\Pi_J^{\smile}$$

The relation ${}_I\delta_J$ expresses the fact that projecting from D_I onto D_J followed by the inverse operation comes down to preserve uniquely the components given by the family of indexes J. The effect of this relation is similar to what is expressed in cylindric algebras [3, 17, 18] by some unary operators called *cylindrification operators*. The relation ${}_I\delta_J$ has been introduced and analyzed in [25].

Clearly we have ${}_I\delta_J = {}_I\delta_J^{\smile}$, and if $K \subseteq J$ then ${}_I\delta_J \,;\, {}_I\delta_K = {}_I\delta_K$.

The relation ${}_I\delta_J$ can also be defined element-wise.

Lemma 4.2

$$(f, g) \in {}_I\delta_J \iff f \in D_I \ \wedge \ g \in D_I \ \wedge \ f|_J = g|_J.$$

Proof. Directly from the definiton of projection and composition ■

Corollary 4.3 *Let I, J, K, and L be subsets of T such that $K \subseteq I$ and $L \subseteq J$. Let $P : D_K \leftrightarrow D_L$ and $Q : D_I \leftrightarrow D_J$ be two relations.*

1. ${}_I\Pi_K ;P; {}_J\Pi_L^{\smile} = \{(f,g) \mid f \in D_I \ \wedge\ g \in D_J \ \wedge\ (f|_K, g|_L) \in P\}$
2. ${}_I\delta_K ;Q; {}_J\delta_L = \{(f,g) \mid f \in D_I \ \wedge\ g \in D_J$
   $\wedge\ \exists (h,t) \in Q \cdot (h|_K = f|_K \ \wedge\ t|_L = g|_L)\}$

■

We can now define $\sqsubseteq$ in terms of projections and cylindrifications.

Theorem 4.4 *Let I, J, K, and L be subsets of T, such that $K \subseteq I$ and $L \subseteq J$. Let $P : D_K \leftrightarrow D_L$ and $Q : D_I \leftrightarrow D_J$ be two relations. Then*

$$P \sqsubseteq Q \iff {}_I\Pi_K ;P; {}_J\Pi_L^{\smile} \ \subseteq\ {}_I\delta_K ;Q; {}_J\delta_L .$$

Proof.

$$\begin{array}{ll}
& {}_I\Pi_K ;P; {}_J\Pi_L^{\smile} \ \subseteq\ {}_I\delta_K ;Q; {}_J\delta_L \\
\iff & \qquad \langle \text{ corollary 4.3 } \rangle \\
& \{(f,g) \mid f \in D_I \ \wedge\ g \in D_J \ \wedge\ (f|_K, g|_L) \in P\} \\
& \subseteq \quad \{(f,g) \mid f \in D_I \ \wedge\ g \in D_J \ \wedge\ \exists (g_1, g_2) \in Q \cdot (g_1|_K = f|_K \ \wedge\ g_2|_L = g|_L)\} \\
\iff & \qquad \langle \text{ relabelling } f|_K \text{ and } g|_L \text{ as } f_1 \text{ and } f_2 \text{, respectively } \rangle \\
& \forall (f_1, f_2) \in P \cdot \Big(\exists (g_1, g_2) \in Q \cdot (g_1|_K = f_1 \ \wedge\ g_2|_L = f_2)\Big) \\
\iff & \qquad \langle \text{ definition of } \sqsubseteq \ \rangle \\
& P \sqsubseteq Q
\end{array}$$

■

4.3 Operations $\oplus$, $\otimes$ and $\ominus$

The survey of known tables used in Software Engineering [1] has shown that in all cases we have either $R = \bigoplus \bigotimes R_\alpha$ or $R = \bigotimes \bigoplus R_\alpha$ (or some special case of the above two) where $\oplus$ and $\otimes$ are some generalizations of $\cup$ and $\cap$. The opration $\ominus$ is a generalization of $\setminus$. Note that the operations $\oplus$, $\otimes$ are *total* ($\cup$, $\cap$ are partial).

Let P and Q be two relations such that: $P : D_I \leftrightarrow D_J$ and $Q : D_K \leftrightarrow D_L$. W define

- $P \otimes Q = \{(f,g) \in D_{I \cup K} \times D_{J \cup L} \ \mid\ (f|_I, g|_J) \in P \ \wedge\ (f|_K, g|_L) \in Q\}$
- $P \oplus Q = \{(f,g) \in D_{I \cup K} \times D_{J \cup L} \ \mid\ (f|_I, g|_J) \in P \ \vee\ (f|_K, g|_L) \in Q\}$
- $P \otimes Q = \{(f,g) \in D_{I \cup K} \times D_{J \cup L} \ \mid\ (f|_I, g|_J) \in P \ \wedge\ (f|_K, g|_L) \notin Q\}$

Let $P : X_1 \times X_3 \leftrightarrow X_5$ and $Q : X_1 \times X_2 \leftrightarrow X_4$ where $X_1 = X_2 = X_3 = X_4 = X_5 = Reals$. Suppose that

$$P = \{((x_1, x_3), x_5) \mid x_5 = x_1 + x_3\}$$

and

$$Q = \{((x_1, x_2), x_4) \mid x_4 = x_1 * x_2.\}$$

Then we have

$$P \oplus Q = \{((x_1, x_2, x_3), (x_4, x_5)) \mid ((x_1, x_3), x_5) \in P \ \vee \ ((x_1, x_2), x_4) \in Q\},$$

$$P \otimes Q = \{((x_1, x_2, x_3), (x_4, x_5)) \mid ((x_1, x_3), x_5) \in P \ \wedge \ ((x_1, x_2), x_4) \in Q\},$$

and

$$P \ominus Q = \{((x_1, x_2, x_3), (x_4, x_5)) \mid ((x_1, x_3), x_5) \in P \ \wedge \ ((x_1, x_2), x_4) \notin Q\}.$$

Lemma 4.5

If $P : D_I \leftrightarrow D_J$ and $Q : D_K \leftrightarrow D_L$. then

1. $P \oplus Q = {}_{(I \cup K)}\Pi_I ; P ; {}_{(J \cup L)}\Pi_J^{\smile} \ \cup \ {}_{(I \cup K)}\Pi_K ; Q ; {}_{(J \cup L)}\Pi_L^{\smile}$
2. $P \otimes Q = {}_{(I \cup K)}\Pi_I ; P ; {}_{(J \cup L)}\Pi_J^{\smile} \ \cap \ {}_{(I \cup K)}\Pi_K ; Q ; {}_{(J \cup L)}\Pi_L^{\smile}$
3. $P \ominus Q = {}_{(I \cup K)}\Pi_I ; P ; {}_{(J \cup L)}\Pi_J^{\smile} \ \cap \ {}_{(I \cup K)}\Pi_K ; \overline{Q} ; {}_{(J \cup L)}\Pi_L^{\smile}$

Proof.

1. $P \oplus Q = {}_{(I \cup K)}\Pi_I ; P ; {}_{(J \cup L)}\Pi_J^{\smile} \ \cup \ {}_{(I \cup K)}\Pi_K ; Q ; {}_{(J \cup L)}\Pi_L^{\smile}$
$= \quad \langle$ corollary 4.3(1) $\ \&\ $ $\{x|p\} \cup \{x|q\} = \{x|p \ \vee \ q\}$ $\rangle$
$\{(f, g) \in D_{I \cup K} \times D_{J \cap L} \mid (f|_I, g|_J) \in P \ \vee \ (f|_K, g|_L) \in Q\}$

The proofs of 2 and 3 are similar. ■

One may observe that if $I = K$ and $J = L$ then $P \otimes Q = P \cap Q, P \oplus Q = P \cup Q$, and $P \ominus Q = P \setminus Q$. The operator $\otimes$ can also be regarded as a generalization of a *natural join* operator used in relational data bases [2].
It turns out we can express $\ominus$ and $\oplus$ using $\otimes$, $\cup$, and $\overline{}$.

Lemma 4.6

Let $P, Q,$ and R be relations. Then

$$P \ominus Q = P \otimes \overline{Q} \text{ and also } P \oplus Q = P \otimes Q \cup P \otimes \overline{Q} \cup \overline{P} \otimes Q.$$

∎

The proof follows from the fact that ${}_I\Pi_J$ is total and univalent.

We will now show that $\oplus$ and $\otimes$ obey distributivity laws similar to those for $\cup$ and $\cap$.

Lemma 4.7

Let $P, Q,$ and R be relations.

1. $P \otimes (Q \oplus R) = (P \otimes Q) \oplus (P \otimes R)$
2. $P \oplus (Q \otimes R) = (P \oplus Q) \otimes (P \oplus Q)$

Proof. Let $P : D_I \leftrightarrow D_J$, $Q : D_K \leftrightarrow D_L$ and $R : D_M \leftrightarrow D_N$.

1.
$$\begin{array}{ll}
 & P \otimes (Q \oplus R) \\
= & \qquad \langle \text{ definition of } \oplus \rangle \\
 & P \otimes \left({}_{(K\cup M)}\Pi_K ; Q ; {}_{(L\cup N)}\Pi^{\smile}_L \;\cup\; {}_{(K\cup M)}\Pi_M ; R ; {}_{(L\cup N)}\Pi^{\smile}_N \right) \\
= & \qquad \langle \text{ definition of } \otimes \rangle \\
 & {}_{(I\cup K\cup M)}\Pi_I ; P ; {}_{(J\cup L\cup N)}\Pi^{\smile}_J \\
 & \cap\; {}_{(I\cup K\cup M)}\Pi_{K\cup M} ; \left({}_{(K\cup M)}\Pi_K ; Q ; {}_{(L\cup N)}\Pi^{\smile}_L \;\cup\; {}_{(K\cup M)}\Pi_M ; R ; {}_{(L\cup N)}\Pi^{\smile}_N \right) ; {}_{(J\cup L\cup N)}\Pi^{\smile}_{L\cup N} \\
= & \qquad \langle \cup\text{-distributivity of } ; \;\; \& \;\; \cup\text{-distributivity of } \cap \rangle \\
 & {}_{(I\cup K\cup M)}\Pi_I ; P ; {}_{(J\cup L\cup N)}\Pi^{\smile}_J \\
 & \quad \cap\; {}_{(I\cup K\cup M)}\Pi_{K\cup M} ; {}_{(K\cup M)}\Pi_K ; Q ; {}_{(L\cup N)}\Pi^{\smile}_L ; {}_{(J\cup L\cup N)}\Pi^{\smile}_{L\cup N} \\
 & \cup\; {}_{(I\cup K\cup M)}\Pi_I ; P ; {}_{(J\cup L\cup N)}\Pi^{\smile}_J \\
 & \quad \cap\; {}_{(I\cup K\cup M)}\Pi_{K\cup M} ; {}_{(K\cup M)}\Pi_M ; R ; {}_{(L\cup N)}\Pi^{\smile}_N ; {}_{(J\cup L\cup N)}\Pi^{\smile}_{L\cup N} \\
= & \qquad \langle {}_I\Pi_J ; {}_J\Pi_K = {}_I\Pi_K \\
 & \qquad\;\; \& \;\; (P;Q)^{\smile} = Q^{\smile} ; P^{\smile} \rangle \\
 & {}_{(I\cup K\cup M)}\Pi_I ; P ; {}_{(J\cup L\cup N)}\Pi^{\smile}_J \;\cap\; {}_{(I\cup K\cup M)}\Pi_K ; Q ; {}_{(J\cup L\cup N)}\Pi^{\smile}_L \\
 & \cup\; {}_{(I\cup K\cup M)}\Pi_I ; P ; {}_{(J\cup L\cup N)}\Pi^{\smile}_J \;\cap\; {}_{(I\cup K\cup M)}\Pi_M ; R ; {}_{(J\cup L\cup N)}\Pi^{\smile}_N \\
= & \qquad \langle \text{ for } K \subseteq J \subseteq I \text{ we have } {}_I\Pi_J ; {}_J\Pi_K = {}_I\Pi_K \\
 & \qquad\;\; \& \;\; (P;Q)^{\smile} = Q^{\smile} ; P^{\smile} \\
 & \qquad\;\; \& \;\; \cup\text{-distributivity of } ; \rangle
\end{array}$$

$$
\begin{aligned}
& {}_{(I\cup K\cup M)}\Pi_{I\cup K} ; \left({}_{(I\cup K)}\Pi_{I} ; P ; {}_{(J\cup L)}\Pi^{\smile}_{J} \;\cap\; {}_{(I\cup K)}\Pi_{K} ; Q ; {}_{(J\cup L)}\Pi^{\smile}_{L} \right) ; {}_{(J\cup L\cup N)}\Pi^{\smile}_{J\cup L} \\
& \cup {}_{(I\cup K\cup M)}\Pi_{I\cup M} ; \left({}_{(I\cup M)}\Pi_{I} ; P ; {}_{(J\cup N)}\Pi^{\smile}_{J} \;\cap\; {}_{(I\cup M)}\Pi_{M} ; R ; {}_{(J\cup N)}\Pi^{\smile}_{N} \right) ; {}_{(J\cup L\cup N)}\Pi^{\smile}_{J\cup N} \\
= \quad & \qquad \langle \text{ definition of } \otimes \rangle \\
& {}_{(I\cup K\cup M)}\Pi_{I\cup K} ; (P \otimes Q) ; {}_{(J\cup L\cup N)}\Pi^{\smile}_{J\cup L} \\
& \cup {}_{(I\cup K\cup M)}\Pi_{I\cup M} ; (P \otimes R) ; {}_{(J\cup L\cup N)}\Pi^{\smile}_{J\cup N} \\
= \quad & \qquad \langle \text{ definition of } \oplus \rangle \\
& (P \otimes Q) \oplus (P \otimes R)
\end{aligned}
$$

2. The proof is similar to the previous, except that we use the following facts: ${}_{I}\Pi_{J}$ is univalent, and if P is univalent then we have $P;(Q\cap R);P^{\smile} = P;Q;P^{\smile} \cap P;R;P^{\smile}$.

■

4.4 Classification on the Basis of Table Composition Rule

Let R be a relation specified by a tabular expression. The survey [1] shows that the patterns $\bigoplus_j \bigotimes_i R_{i,j}$, $\bigotimes_j \bigoplus_i R_{i,j}$, and their special cases as $\bigotimes_j \bigcup_i R_{i,j}$, $\bigcup_\alpha R_\alpha$, and $\bigcup_i \bigotimes_j R_{i,j}$ are sufficient in all the cases. This gives us some bases for the following classification.

- The table is called *plain* if $R = \bigcup_{\alpha\in I} R_\alpha$.
- The table is called *output-vector* if $R = \bigotimes_j \bigoplus_i R_{i,j}$.
- The table is called *input-vector* if $R = \bigoplus_i \bigotimes_j R_{i,j}$.

All tables modeled in [22] are plain. The vector tables of [36] are of output-vector type, the most of (but not all) decision tables [19, 20, 36] are of input-vector type.

5 Final Comment

In the paper a formal semantics for tabular expressions is proposed. The tables introduced here are generalizations of those from [22, 36] and [1]. As opposed to [36], one model covers all cases. An introduction of cell connection graph, table predicate rules, table relation rules and table composition rules gives us a tool to define various types of tables, some of them could really be useful. The cell connection graph and the table composition rule are major sources of the classification (on the syntactic level, without taking Ψ into account). In this paper the tabular expressions have been divided into six different classes according to cell connection graph, and three major types have been distinguished according to the table connection rule. This paper is an extension and continuation of [1, 22, 23]. In [22] only plain tables were considered, [1] gives some initial models for non-plain tables. The model covers all types of tables currently used in Software Engineering. It also allows us to define precisely new types tables.

An alternative semantics in terms of arrays of relations has been proposed in [10]. The operations $\oplus$, $\otimes$ and $\ominus$ were application driven. We think that in general the problem of composing R from R_α's, $\alpha \in I$ is an open reserach problem, that can be formulated as "how to build the whole, i.e. R, from the parts, i.e.R_α's" in terms of the algebra of relations.

Acknowledgments

Dave Parnas provided both inspiration and initial attempt to formalize the concept of tables. Ruth Abraham is thanked for many discussions and for showing how the theory can be applied in practice. John van Schouwen is thanked for detailed comments and correcting many errors.

References

[1] R. Abraham, Evaluating Generalized Tabular Expressions in Software Documentation, M. Eng. Thesis, Dept. of Electrical and Computer Engineering, McMaster University 1997, also CRL Report 346, McMaster University, Hamilton, Ontario, Canada, 1997. Available at http://www.crl.mcmaster.ca/SERG/serg.publications.html

[2] A. V. Aho, J. D. Ullman, *Foundations of Computer Science*, Computer Science Press 1992.

[3] H. Andréka, I. Németi. On cylindric-relativized set algebras. In A. Dold, B. Eckmann (eds.), *Cylindric Set Algebras*, *Lecture Notes in Mathematics* 883, Springer-Verlag 1981, pp. 131-315.

[4] G. H. Archinoff, R. J. Hohendorf, A. Wassyng, B. Quigley, M. R. Borsch, Verification of the Shutdown System Software at the Darlington Nuclear Generating Station, *International Conference on Control and Instrumentation in Nuclear Installations*, Glasgow, U.K., 1990, No. 4.3.

[5] R. Berghammer, H. Zierer. Relational algebraic semantics of deterministic and nondeterministic programs. *Theoretical Computer Science*, 43:123–147, 1986.

[6] H.N. Cantrell, J. King, and F.E.H. King. Logic structure tables. *Communications of the ACM*, 4(6):272–275, June 1961.

[7] Ned Chapin. An introduction to decision tables. *DPMA Quarterly*, 3(3):2–23, April 1967.

[8] L. Chin, A. Tarski. Distributive and modular laws in the arithmetic of relation algebras. *University of California Publications*, 1:341–384, 1951.

[9] P. C. Clements, Function Specification for the A-7E Function Driver Module, *NRL Memorandum Report 4658*, U.S. Naval Research Lab., 1981.

[10] J. Desharnais, R. Khédri, A. Mili, Towards a Uniform Relational Semantics for Tabular Expressions, Proc. of RELMICS 98, Warsaw 1998.

[11] Burton Grad. Tabular form in decision logic. *Datamation*, 7(7):22–26, July 1961.

[12] P. R. Halmos, *Naive Set Theory*, Springer 1960.

[13] M. P. E. Heimdahl, N. G. Leveson, Completeness and Consistency Analysis of State-Based Requirements, *17th International Conference on Software Engineering* (*ICSE'95*), IEEE Computer Society, Seattle, WA, 1995, 3-14.

[14] C. Heitmeyer, A. Bull, C. Gasarch, B. Labaw, SCR*: A Toolset for Specifying and Analyzing Requirements, *Proc. 9th Annual Conf. on Computer Assurance* (*COMPASS'95*), Gaithersburg, MD, 1995.

[15] K. L. Heninger, Specifying Software Requirements for Complex Systems: New Techniques and their Applications, *IEEE Transactions on Software Engineering*, 6, 1, (1980), 2-13.

[16] K. L. Heninger, J. Kallander, D. L. Parnas, J. E. Shore, Software Requirements for the A-7E Aircraft, *NRL Memorandum Report 3876*, U.S. Naval Research Lab., 1978.

[17] L. Henkin, J. D. Monk, A. Tarski, *Cylindric Algebras, Part I*, North Holland 1971.

[18] L. Henkin, J. D. Monk, A. Tarski. Cylindric set algebras and related structures. In A. Dold, B. Eckmann (eds.), *Cylindric Set Algebras*, *Lecture Notes in Mathematics* 883, Springer-Verlag 1981, pp. 1-129.

[19] D. N. Hoover, Z. Chen, Tablewise, a Decision Table Tool, *Proc. 9th Annual Conf. on Computer Assurance* (*COMPASS'95*), Gaithersburg, MD, 1995.

[20] R. B. Hurlay, *Decision Tables in Software Engineering*, Van Nostrand Reinhold Company, New York 1983.

[21] IBM Corp. Hipo-a design aid and documentation technique. (White Plains, NY: IBM corp., 1974).

[22] R. Janicki, Towards a Formal Semantics of Parnas Tables, *17th International Conference on Software Engineering* (*ICSE'95*), IEEE Computer Society, Seattle, WA, 1995, 231-240.

[23] R. Janicki, On Formal Semantics of Tabular Expressions, CRL Report 355, McMaster University, Hamilton, Ontario 1997. Available at http://www.crl.mcmaster.ca/SERG/serg.publications.html

[24] R. Janicki, D. L. Parnas, J. Zucker, Tabular Representations in Relational Documents, in C. Brink, W. Kahl, G. Schmidt (eds.): *Relational Methods in Computer Science*, Springer-Verlag 1997.

[25] R. Khédri. *Concurrence, bisimulation et équation d'interface : une approche relationnelle.* PhD thesis, Faculté des études supérieures de l'Université Laval, Québec, Canada, 1998.

[26] L. Lamport, How to Write a Long Formula, *SRC Research Report 119*, DEC System Research Centre, Palo Alto, CA, 1993.

[27] N. G. Leveson, M. P. E. Heimdahl, H. Hildreth, J. D. Reese, Requirements Specifications for Process-Control Systems, *IEEE Transaction on Software Engineering*, 20, 9, 1994.

[28] R. Maes. On the representation of program structures by decision tables: a critical assessment. *The Computer Journal*, 21(4):290–295, November 1978.

[29] Herman McDaniel. *An Introduction to Decision Logic Tables.* New York: John Wiley and Sons Inc., 1968.

[30] J. McDougall, E. Jankowski, Procedure for the Specification of Software Requirements for Safety Critical Systems, Report CE-1001-PROC, Computer Centre of Excellence, 1995.

[31] Michael Montalbano. *Decision Tables.* Palo Alto, CA: Science Research Associates, 1974.

[32] Nickerson R.C. An engineering application of logic-structure tables. *Communications of the ACM*, 4(11):516–520, November 1961.

[33] D. L. Parnas, G. J. K. Asmis, J. D. Kendall, Reviewable Development of Safety Critical Software, *International Conference on Control and Instrumentation in Nuclear Installations*, Glasgow, U.K., 1990, No. 4.3.

[34] D. L. Parnas, G. L. K. Asmis, J. Madey, Assessment of Safety-Critical Software in Nuclear Power Plants, *Nuclear Safety*, 32,2 (1991), 189-198.

[35] D. L. Parnas, A Generalized Control Structure and Its Formal Definition, *Communications of the ACM*, 26, 8 (1983), 572-581.

[36] D. L. Parnas, Tabular Representation of Relations, *CRL Report 260*, Telecommunications Research Institute of Ontario (TRIO), McMaster University, Hamilton, Ontario, Canada, 1992.

[37] D. L. Parnas, Personal communication, 1993.

[38] D. L. Parnas, J. Madey, Functional Documentation for Computer Systems Engineering, *Science of Computer Programming*, 25, 1 (1995), 41-61.

[39] D. L. Parnas, J. Madey, M. Iglewski, Precise Documentation of Well-Structured Programs, *IEEE Transactions on Software Engineering*, 20, 12 (1994), 948-976.

[40] G. Schmidt, T. Ströhlein. *Relations and Graphs.* EATCS Monographs in Computer Science, Springer-Verlag, 1993.

[41] G. Schmidt, T. Strölein. Relations algebras: Concept of points and representability. *DISCR*, 54:83–92, 1985.

[42] A. J. van Schouwen, The A-7 Requirements Model: Re-examination for Real-Time Systems and an Application to Monitoring Systems, *Technical Report 90-276*, Queen's University, CIS, TRIO, Kingston, Ontario, Canada, 1990.

[43] SERG - Software Engineering Group, Table Tool System Developer's Guide, CRL REport 339, TRIO, McMaster University, Hamilton, Ontario, Canada 1997.

[44] J. T. J. Strzednicki, V. F. Rickey (eds.), *Lesniewski's Systems*, Kluwer Academic, 1984.

[45] J. Zucker, Transformations of Normal and Inverted Function Tables, *Formal Aspects of Programming*, 8 (1996), 679-705.

Appendix

This Appendix contains some illustrative examples of direct product, restriction operation, projection and cylindrification relations. We start with the example of a direct product.

Example 5.1 *Let* $T = \{u, v, w\}$*,* $D_u = \{1, 2\}$*,* $D_v = \{\alpha, \beta\}$ *and* $D_w = \{m, n\}$*.*

$$D_T = \prod_{t \in T} D_t = D_u \times D_v \times D_w = \{(a, b, c) \mid a \in D_u \ \wedge \ b \in D_v \ \wedge \ c \in D_w \}$$

The set D_T *can be seen as the folowing relation.*

$$\Big\{ f \mid f : \{u, v, w\} \longrightarrow \{1, 2, \alpha, \beta, m, n\} \ \wedge \ f(u) \in D_u \ \wedge \ f(v) \in D_v \ \wedge \ f(w) \in D_w \Big\}.$$

The three following tables represent these two isomorphic representations of D_T*. The first table represents* D_T *as a set of functions. In the second table we have, with a permutation of the columns, the same set of functions. The representation given by the third table supposes that the elements of the first column belong to* D_u*, the elements of the second column belong to* D_v*, and the elements of the third column belong to* D_w*.*

$f(u)$	$f(v)$	$f(w)$
1	α	m
1	α	n
1	β	m
1	β	n
2	α	m
2	α	n
2	β	m
2	β	n

$\Longleftrightarrow$

$f(w)$	$f(v)$	$f(u)$
m	α	1
n	α	1
m	β	1
n	β	1
m	α	2
n	α	2
m	β	2
n	β	2

1	α	m
1	α	n
1	β	m
1	β	n
2	α	m
2	α	n
2	β	m
2	β	n

■

The next example is an example of a function restriction.

Example 5.2 *If* $I = \{1, 2, 3, 4\}$*,* $K = \{2, 4\}$*,* $D_I = X_1 \times X_2 \times X_3 \times X_4$*,* $f = (x_1, x_2, x_3, x_4) \in D_I$ *(or* $f : \{1, 2, 3, 4\} \longrightarrow D_1 \cup D_2 \cup D_3 \cup D_4$*, and* $f(i) = x_i$*,* $i = 1, 2, 3, 4$*), then* $f_K : \{2, 4\} \longrightarrow D_2 \cup D_4$*,* $f|_K(i) = x_i$*,* $i = 2, 4$*, i.e.* $f|_K = (x_2, x_4)$*.* ■

We shall now illustrate the concept of projection relation.

Example 5.3 *We continue with the terms of Example 5.1. Let us take* $I = \{v, w\}$*. The following tables illustrate the relation* ${}_T\Pi_I$*. A row from the first table represents a function* $f \in D_T$ *and the corresponding row from the second table represents a function* $g \in D_I$*. Hence, a whole row represents a member of* ${}_T\Pi_I$*, i.e. , third row,* $((1, \beta, m), (\beta, m)) \in {}_T\Pi_I$*.*

$f(u)$	$f(v)$	$f(w)$
1	α	m
1	α	n
1	β	m
1	β	n
2	α	m
2	α	n
2	β	m
2	β	n

$g(v)$	$g(w)$
α	m
α	n
β	m
β	n
α	m
α	n
β	m
β	n

In this case we can either write:

$$_T\Pi_I = \{(f,g) \mid f \in D_T \ \wedge \ g \in D_I \ \wedge \ f(v) = g(v) \ \wedge \ f(w) = g(w)\,\}.$$

or, by using tuples instead of functions,

$$_T\Pi_I = \{\big((a,b,c),(b',c')\big) \mid b' = b \ \wedge \ c' = c\,\}.$$

■

The last example is an example of cylindrification relation.

Example 5.4 *Let us take the projection $_T\Pi_I$ of Example 5.3. We have*

$$
\begin{array}{ll}
 & _T\delta_I = {_T\Pi_I} ; {_T\Pi_I^{\smile}} \\
= & \qquad \langle \text{ Example5.3 } \rangle \\
 & \{((a,b,c),(b',c')) \mid b' = b \ \wedge \ c' = c\,\} ; \{((b,c),(a',b',c')) \mid b' = b \ \wedge \ c' = c\,\} \\
= & \qquad \langle \text{ definition of ; } \rangle \\
 & \{((a,b,c),(a',b',c')) \mid b' = b \ \wedge \ c' = c\,\}
\end{array}
$$

*The below tables illustrate the relation $_T\delta_I$, where T, I, and D_T are those from Example 5.1. A row from the first table is an element of D_T. The same row from the second table corresponds to this element by $_T\delta_I$. A tuple: (*a row from first table, a row from the second table*) is an element of the relation $_T\delta_I$.*

1	α	m
1	α	m
1	α	n
1	α	n
1	β	m
1	β	m
1	β	n
1	β	n
2	α	m
2	α	m
2	α	n
2	α	n
2	β	m
2	β	m
2	β	n
2	β	n

1	α	m
2	α	m
1	α	n
2	α	n
1	β	m
2	β	m
1	β	n
2	β	n
2	α	m
1	α	m
2	α	n
1	α	n
2	β	m
1	β	m
2	β	n
1	β	n

$\blacksquare$