Tabular Expressions and Their Relational Semantics

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

Abstract

Tabular expressions (Parnas et al. [17, 23, 25, 26]) are means to represent the complex relations that are used to specify or document software systems. A formal model and a semantics for tabular expressions are presented. The model covers all known types of tables used in Software Engineering.

1 Introduction

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 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 (see [18]). 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. The *multi-dimensional tabular notation* makes it easier to consider every case separately while writing or reading a design document. It turns out that using tables helps to make mathematics more practical for computing systems applications [17].

The key assumptions behind the idea of tabular expressions are:

*Partially supported by NSERC of Canada Grant

- 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_i, $i \in I$, where I is a set of indices, each R_i can be specified rather easily. In most cases R_i 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_i, the others R_i itself.
- the tabular expression that describes R is a structured collection of cells containing definitions of R_i's. The structure of a tabular expression informs how the relation R can be composed of all the R_i's.

In principle, tabular expressions are generalization of plain two dimentional tables that are known from the beginnings of civilization.

The key ideas of a *tabular notation*, one of the cornerstones of the relational model for documenting the intended behaviour of programs [17, 23, 25, 26], were first developed in work for the U.S. Navy and applied to the A-7E aircraft [11, 10, 4, 29]. 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 [3, 21, 22].

The tabular notation is currently used among others by the Software Engineering Research Group (SERG) at McMaster University, Hamilton, Ontario, Canada [30], Ontario Hydro [20], Naval Research Laboratory [9, 6, 27], ORA Inc., [12], and University of California at Irvine [8, 19].

The model presented here seems to cover all the known types of tables used in Software Engineering (compare [1]).

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.

All examples of tables used in this paper are very simple on purpose. For more realistic examples (as loop invariants, program specifications) the reader is referenced to [1, 30, 26].

Each example (see Figures 6,7 and 8) consists of two parts, a function/relation is first described in classical mathematical notation, and is then followed by an equivalent tabular expression.

The first rough approximation of the table concept is given in Chapter 2. The crucial concept of *Cell Connection Graph* and more precise approximations of the table concept are discussed in Chapters 3, 4, 5 and 6. The formal definition of a *tabular expression* on syntactic level is given in Chapter 7, and its *semantics* is

discussed in Chapter 8. Chapter 9 contains some comments on table classification, and final, comments are in Chapter 10.

The paper [24] provided a major motivation for this work. The early results have been presented in [14]. The paper is a revised version of [15].

We assume that the reader is familiar with such concepts as function, relation, Cartesian product, etc. [7, 28]. The standard mathematical notation is used throughout the paper.

2 Raw Table Skeleton

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 (raw or medium) *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.

We are now able to define the first approximation of table skeleton.

- A *raw table skeleton* is a tuple

$$T = (H_1, ..., H_n, G)$$

  where $H_1, ..., H_n$ are headers and G is the grid indexed by the headers $H_1, ..., H_n$. The elements of the set $Components(T) = \{H_1, ..., H_n, G\}$ are called *table components.*

Figure 1 illustrates the above definitions.

3 Cell Connection Graph and Medium Table Skeleton

The first step in expressing the semantic difference between the various types of tables is to define the *Cell Connection Graph*, which characterizes information flow (*"where do I start reading the table and where do I get my result?"*). Intuitively a Cell Connection Graph is a relation that could be interpreted as an *acyclic directed*

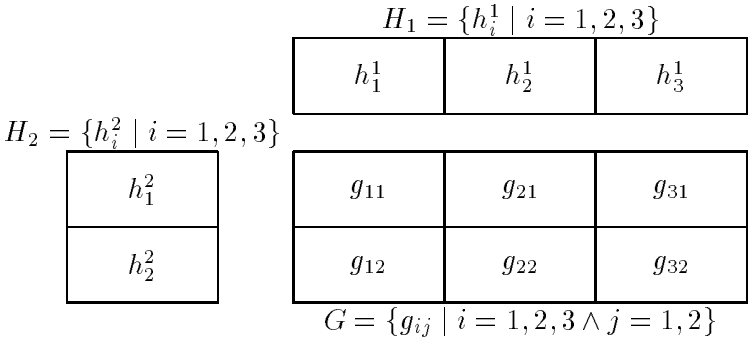


Figure 1: An example of a raw table skeleton $T = (H_1, H_2, G)$.

graph with the grid and all headers as the nodes, plus the decomposition of nodes into two distinct classes called *guard components* and *value components*. The only requirement for the relation is that each arc *must either start from or end at the grid* G.

Let $T = (H_1, ..., H_n, G)$ be a raw table skeleton, i.e. $Components(T) = \{H_1, ..., H_n, G\}$. A *Cell Connection Graph* is an *asymmetric* relation

$$\longmapsto \subseteq Components(T) \times Components(T)$$

satisfying:

$$\forall A, B \in Components(T) \;\; A \longmapsto B \;\Rightarrow\; ((A = G \vee B = G) \wedge A \neq B), \qquad (1)$$

plus a decomposition of $Components(T)$ into $Guards(T)$ and $Values(T)$.

The relation $\longmapsto^*$, transitive and reflexive closure[1] of $\longmapsto$, is a *partial order* [7]. A component $A \in Components(T)$ is *maximal* if $A \longmapsto^* B$ implies $B = A$ for every $B \in Components(T)$. Similarly $A \in Components(T)$ is *minimal* if $B \longmapsto^* A$ implies $B = A$ for every $B \in Components(T)$. A component $A \in Components(T)$ is *neutral* if it is neither minimal nor maximal.

The relation $\longmapsto$ 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[2] of $\longmapsto$. This means that *the components built from the cell*

[1] $A \longmapsto^* B \iff (A = B) \vee (A \longmapsto B) \vee (\exists A_1, ..., A_k.\ A \longmapsto A_1 \longmapsto A_2 \longmapsto ... \longmapsto A_k \longmapsto B)$.
[2] $A \longmapsto^+ B \iff (A \longmapsto B) \vee (\exists A_1, ..., A_k.\ A \longmapsto A_1 \longmapsto A_2 \longmapsto ... \longmapsto A_k \longmapsto B)$.

describing the domains are never maximal, while *the components built from the cells containing formulae for values are never minimal.*

Thus the partition of $Components(T)$ into $Guards(T)$ and $Values(T)$ must satisfy the following properties:

$$\begin{array}{ll} 1. & Components(T) = Guards(T) \cup Values(T), \\ 2. & Guards(T) \cap Values(T) = \emptyset, \\ 3. & A \text{ is maximal } \Rightarrow\ A \in Values(T), \\ 4. & A \text{ is minimal } \Rightarrow\ A \in Guards(T), \\ 5. & \forall A \in Guards(T). \forall B \in Values(T).\ A \longmapsto^{+} B. \end{array} \qquad (2)$$

One can also easily prove the following Lemma.

Lemma 3.1

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

■

We may now define CCG, *Cell Connection Graph*, as a triple

$$CCG = (Guards(T), Values(T), \longmapsto)$$

where $\longmapsto$ satisfies (1) and $Guards(T), Values(T)$ satisfy (2).

There are six "topologically" (but in the popular, not mathematical meaning of this word) different types of Cell Connection Graphs.

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 $Guards(T)$.

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

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

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

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 $Guards(T)$ and $Values(T)$. Figure 2 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:

- type 1 corresponds to Normal Tables of [24],
- type 2a corresponds to Inverted, Decision and Generalized Decision Tables [12, 24],
- type 2b corresponds to Vector Tables of [24].

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.

By adding the Cell Connection Graph we obtain the next approximation of the table skeleton concept.

- By a *medium table skeleton* we mean a tuple

$$T = (CCG, H_1, ..., H_n, G)$$

  where $(H_1, ..., H_n, G)$ is a raw table skeleton and CCG is a cell connection graph for $(H_1, ..., H_n, G)$.

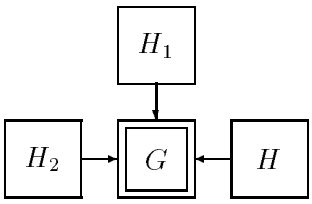
The type of Cell Connection Graph will usually be identified by a small icon resembling an appropriate graph from Figure 2. The icon is placed in left upper corner of the table. Figure 3 presents examples of medium table skeletons.

4 Raw and Medium Table Elements

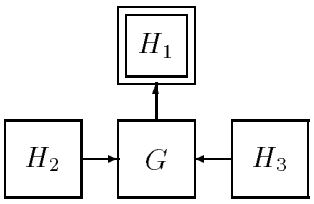
Let $T^{med} = (CCG, H_1, ..., H_n, G)$ be a medium table skeleton with the index I, and let $T^{raw} = (H_1, ..., H_n, G)$ be the raw table skeleton. Consider the element $(h^1_{i_1}, ..., h^n_{i_n}, g_\alpha) \in H_1 \times ... \times H_n \times G$. We shall say that

$$(h^1_{i_1}, ..., h^n_{i_n}, g_\alpha) \text{ is a } \textit{raw element} \iff \alpha = (i_1, ..., i_n).$$

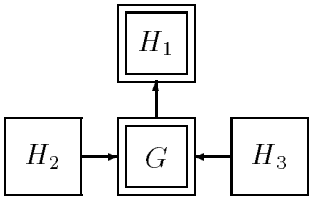
We will denote the raw element $(h^1_{i_1}, ..., h^n_{i_n}, g_\alpha)$ by $T^{raw}|_\alpha)$, since it can be interpreted as a kind of *projection* (*resrtiction*) of T^{raw} onto the index α. The *set*



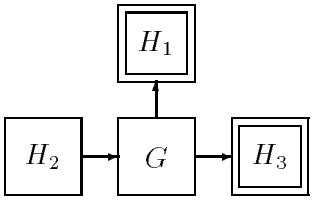
Type 1. Each element is either maximal or minimal. There is only one maximal element.



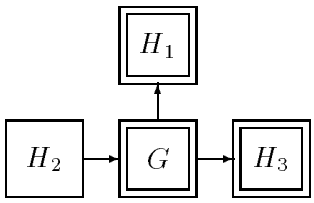
Type 2a. There is only one maximal element and a neutral element. The neutral element belongs to $Guards(T)$.



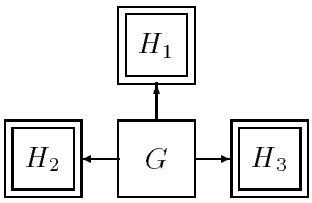
Type 2b. There is only one maximal element and a neutral element. The neutral element belongs to $Values(T)$.



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



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



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

Figure 2: Six different types of cell connection graphs ($n = 3$).

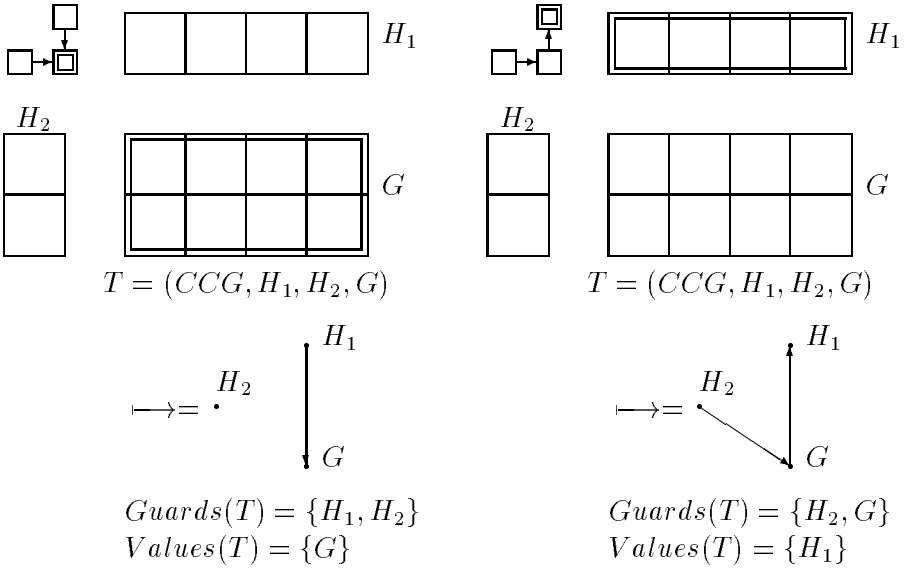


Figure 3: Medium table skeletons.

$\{h^1_{i_1}, ..., h^n_{i_n}, g_\alpha\}$ will be denoted by $Components_\alpha(T^{raw})$.

Let $\longmapsto_\alpha \subseteq Components_\alpha(T^{raw}) \times Components_\alpha(T^{raw})$ be a relation defined as

$$c_1 \longmapsto_\alpha c_2 \iff \exists A_1, A_2 \in Components(T^{raw}).\ c_1 \in A_1 \wedge c_2 \in A_2 \wedge A_1 \longmapsto A_2.$$

Since $\longmapsto_\alpha$ is isomorphic to $\longmapsto$ we will identify them and denote by the same symbol $\longmapsto$, and use the same icon to describe it. We also define $Guards_\alpha(T^{raw})$, $Values_\alpha(T^{raw})$ as appropriate projections of $Guards(T^{raw})$ and $Values(T^{raw})$ onto $(h^1_{i_1}, ..., h^n_{i_n}, g_\alpha)$. Formally

$$Guards_\alpha(T^{raw}) = \{c \mid c \in Componants_\alpha(T^{raw}) \wedge \exists A \in Guards(T^{raw}).\ c \in A\},$$
$$Values_\alpha(T^{raw}) = \{c \mid c \in Componants_\alpha(T^{raw}) \wedge \exists A \in Values(T^{raw}).\ c \in A\}.$$

The triple

$$CCG_\alpha = (Guards_\alpha, Values_\alpha, \longmapsto_\alpha)$$

will be called the *cell connection graph of* $T^{raw}|_\alpha$.

By a *medium element* of T^{med} we mean a tuple

$$T^{med}|_\alpha = (CCG_\alpha, h^1_{i_1}, ..., h^n_{i_n}, g_\alpha)$$

where $(h^1_{i_1}, ..., h^n_{i_n}, g_\alpha)$ is a raw element. Again, a medium element can be interpreted as a projection of T^{med} onto α. Figure 4 illustrates a medium element.

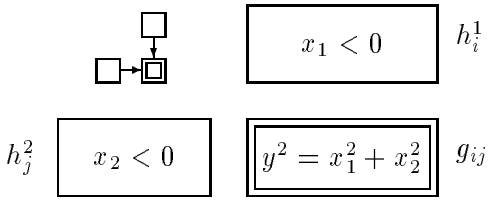


Figure 4: An example of (partially) interpreted medium element.

5 Well Done Table Skeleton

Let R be a relation that is going to be specified by a tabular expression. Let $dom(R)$ and $range(R)$ denote the *domain* and *range* of R respectively. Both $dom(R)$ and $range(R)$ could be Cartesian Products or subsets of Cartesian Products, i.e. in general $dom(R) \subseteq X_1 \times ... \times X_k$, some X_i, $range(R) \subseteq Y_1 \times ... \times Y_m$, some Y_j.

The relation R can be composed of R_α's, $\alpha \in I$, where I is a finite set of indices, and the set $\mathcal{F} = \{R_\alpha \mid \alpha \in I\}$ will be called a *representation* of R.

The very basic idea behind using tables to specify the relation R is that in practice we can frequently use a *medium element* $\Gamma_\alpha = \pi_\alpha(T)$ to specify R_α, $\alpha \in I$. The entire relation R could be very complex, but each R_α is relatively simple. The relation R is equal to $R = Expr(\mathcal{F})$, and the table structure is supposed to make the understanding of $Expr(\mathcal{F})$ natural and simple.

Let $\mathbf{x}$ be a (possible vector) variable over $dom(R)$, $\mathbf{y}$ be a (possible vector) variable over $range(R)$, and let $P(\mathbf{x})$ be a predicate defining the domain of R_α, i.e.

$$\mathbf{x} \in dom(R_\alpha) \subseteq dom(R) \iff P(\mathbf{x}) = true.$$

Let $E_\alpha(\mathbf{x}, \mathbf{y})$ be a relational expression that defines (in a readable way) a superset E_α of the relation R_α, i.e.

$$R_\alpha \subseteq E_\alpha \text{ where } (\mathbf{x}, \mathbf{y}) \in E_\alpha \iff E_\alpha(\mathbf{x}, \mathbf{y}).$$

The relation R_α is a restriction of E_α to $dom(R_\alpha)$, i.e. $R_\alpha = E_\alpha|_{dom(\alpha)}$, and is entirely described by the following predicate expression[3]

$$\textbf{if } P_\alpha(\mathbf{x}) \textbf{ then } E_\alpha(\mathbf{x}, \mathbf{y}).$$

[3]The predicate **if** $P_\alpha(\mathbf{x})$ **then** $E_\alpha(\mathbf{x}, \mathbf{y})$ can equivalently be written as $P_\alpha(\mathbf{x}) \wedge E_\alpha(\mathbf{x}, \mathbf{y})$. We shall prefer **if-then** form because it is more readable, in particular when $P_\alpha(\mathbf{x})$ itself contains "$\wedge$" operator (see Figure 6). But clearly **if** $P_\alpha(\mathbf{x})$ **then** $E_\alpha(\mathbf{x}, \mathbf{y}) = P_\alpha(\mathbf{x}) \wedge E_\alpha(\mathbf{x}, \mathbf{y})$.

We have to now fit the *predicate* expression **if** $P_\alpha(\mathbf{x})$ **then** $E_\alpha(\mathbf{x},\mathbf{y})$ into the *medium element* $\Gamma_\alpha = T|_\alpha$. Figure 4 shows how it can be done for the expression **if** $x_1 < 0 \wedge x_2 < 0$ **then** $y^2 = x_1^2 + x_2^2$,

The idea we will be using is the following:

- the expressions defining the relational expression $E_\alpha(\mathbf{x},\mathbf{y})$ are held in *value cells* $(Values(T))$.
- the expressions defining the predicate expression $P_\alpha(\mathbf{x})$ are held in *guard cells* $(Guards(T)$.

However, the partition of cells into value and guard types is not sufficient. Let us consider the cell connection graph from Figure 4. We said it corresponded to the expression **if** $x_1 < 0 \wedge x_2 < 0$ **then** $y^2 = x_1^2 + x_2^2$. But *why* $x_1 < 0 \wedge x_2 < 0$? Why not for example: $x_1 < 0 \vee x_2 < 0$, or $\neg(x_1 < 0) \wedge x_2 < 0$ etc.?

There is no explicit information in the table that indicates conjunction, or any other operation. A medium 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.

The similar situation we have for the expression $E_\alpha(\mathbf{x},\mathbf{y})$. For Types 2b, 3a, 3b and 4, $E_\alpha(x,y)$ must somehow be composed of two or more components, each component is described by an expression in held in one cell.

To say precisely how the *medium element* can be used to specify the expression **if** $P_\alpha(\mathbf{x})$ **then** $E_\alpha(\mathbf{x},\mathbf{y})$, we need not only to divide cells into value and guard types, but also to decide how $P_\alpha(\mathbf{x})$ can be built from the expressions held in the guard cells and $E_\alpha(\mathbf{x},\mathbf{y})$ from the expressions held in the value cells.

Let $T = (CCG, H_1, ..., H_n, G)$ be a medium table skeleton. Assume that $Guards(T) = \{B_1, ..., B_r\}$, $Values(T) = \{A_1, ..., A_s\}$.

- A predicate expression $P_T(B_1, ..., B_r)$, where $B_1, ..., B_r$ are variables, is called a *table predicate rule.*
- A relation expression $r_T(A_1, ..., A_s)$, where $A_1, ..., A_s$ are variables, is called *table relation rule.*

The predicate $P_\alpha(\mathbf{x})$ can now we derived from $P_T(B_1, ..., B_s)$ by replacing each variable B_i by the content of the cell that belongs to both the medium element Γ_α and

the component B_i. Similarly, the relation expression $E_\alpha(\mathbf{x}, \mathbf{y})$ can now we derived from $r_T(A_1, ..., A_s)$ by replacing each variable A_i by the content of the cell that belongs to both the medium element Γ_α and the component A_i.

The predicate expression P_T is built from table component names (variables) $B_1, ..., B_r$, where $Guards(T) = \{B_1, ..., B_r\}$, logical operators "$\wedge$", "$\vee$", "$\neg$" (however "$\neg$" is at present disallowed for implementation reasons in the SERG tool package [1, 30]), 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 occurrence of x in E by E_1. The constants and relation symbols depend on the type of input domain $dom(R)$. The relation symbol "=" can always be used. If the elements of $dom(R)$ are ordered, the relation symbols "<", ">" can be used[4].

The relation expression r_T 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" "$\circ$"[5].

The table predicate and relation rules are sufficient to understand how the expressions **if** $P_\alpha(\mathbf{x})$ **then** $E_\alpha(\mathbf{x}, \mathbf{y})$ 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 that create a *representation* $\mathcal{F}$ of R. There is nothing in the middle table skeleton to say how all those R_α's should be composed, which leads as to the following concept.

- A relation expression C_T of the form $R = Expr(\mathcal{F})$ is called a *table composition rule.*

In general, $Expr(\mathcal{F})$ is a relational expression builts from the expressions defining R_α's, and various relational operators. We shall discuss it in detail in the next section.

The final approximation of a table skeleton is the following.

- A *well done table skeleton* is a tuple

$$T = (P_T, r_T, C_T, CCG, H_1, ..., H_n, G),$$

  where $(CCG, H_1, ..., H_n, G)$ is a medium table skeleton, P_T is a table predicate rule, r_T is a table relation rule, and C_T is a table composition rule.

[4] 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 [14] of the approach presented here.

[5] 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 $y_3 + x_1x_2 = |y_3|^3$.

In principle, well done table skeleton defines all the structure of a tabular expression except filling out all the cells with proper expressions that define all R_α's.

The definition is illustrated in Figure 5.

6 Composing R from R_α

Let $P \subseteq X \times Y$, $Q \subseteq X' \times Y'$. If $X = X'$ and $Y = Y'$, then $P \cup Q$, $P \cap Q$, $P \setminus Q$ are standardly defined (see [28]).

We shall all a reprsentation $\mathcal{F} = \{R_\alpha \mid \alpha \in I\}$ of R *plain* if for all α, $dom(R_\alpha) \subseteq dom(R)$ and $R = \bigcup_{\alpha \in I} R_\alpha$.

The model presented in [14] deals with plain represenations only.

In many cases R_α's are heterogenous relations defined on the domains that can intuitively be interpreted as subdomains of $dom(R)$ but they are *not* subsets of $dom(R)$. For example $dom(R) = D_1 \times D_2 \times D_3$, $dom(R_1) = D_1 \times D_3$, $dom(R_2) = D_2 \times D_3$, etc.

Let T be a set of indices, $\{D_t \mid t \in T\}$ be a family of sets (domains), J, K, L, M be subsets of T, and let P, Q be the following relations

$$P \subseteq \prod_{t \in J} D_t \times \prod_{t \in K} D_t,\ Q \subseteq \prod_{t \in L} D_t \times \prod_{t \in M} D_t.$$

$\prod_{t \in J} D_t$ denotes the direct product of D_t, for all $t \in T$. For example if $J = \{1, 2, 4\}$ then $\prod_{t \in J} D_t$ is equivalent to $D_1 \times D_2 \times D_4$.

If $J \subseteq L$, $x \in \prod_{t \in L} D_t$, then $x|_J \in \prod_{t \in J} D_t$ is a *restriction* (*projection*) of x to J. For instance if $x = (x_1, x_2, x_4) \in D_1 \times D_2 \times D_4$, $J = \{1, 4\}$, then $x|_J = (x_1, x_4)$.

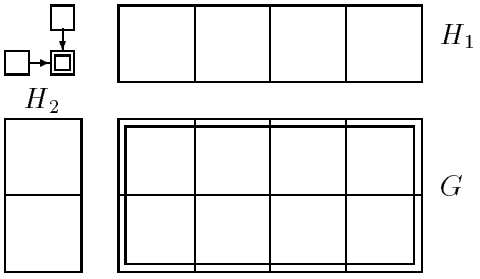
We now define the oparations $\oplus$, $\otimes$, $\ominus$ as follows

$$\begin{aligned}
P \oplus Q &= \{(x,y) \mid x \in \textstyle\prod_{t \in J \cup L} D_t \wedge y \in \prod_{t \in K \cap M} D_t \wedge ((x|_J, y|_K) \in P \vee (x|_L, y|_M) \in Q)\},\\
P \otimes Q &= \{(x,y) \mid x \in \textstyle\prod_{t \in J \cup L} D_t \wedge y \in \prod_{t \in K \cap M} D_t \wedge ((x|_J, y|_K) \in P \wedge (x|_L, y|_M) \in Q)\},\\
P \ominus Q &= \{(x,y) \mid x \in \textstyle\prod_{t \in J \cup L} D_t \wedge y \in \prod_{t \in K \cap M} D_t \wedge ((x|_J, y|_K) \in P \wedge (x|_L, y|_M) \notin Q)\},
\end{aligned}$$

For example if $dom(P) \subseteq D_1 \times D_3$, $range(P) \subseteq D_5$, $dom(Q) \subseteq D_1 \times D_2$, $range(Q) \subseteq D_4$, where all $D_1, ..., D_5$ are reals, and

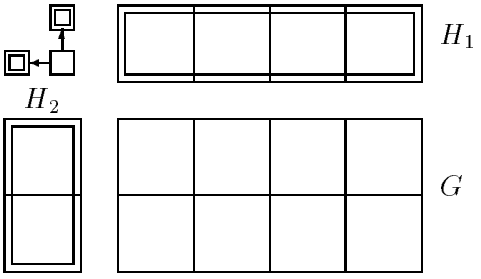
$$P = \{((x_1, x_2), x_5) \mid x_5 = x_1 + x_3\}\ Q = \{((x_1, x_2), x_4) \mid x_4 = x_1 * x_2\}$$

Normal



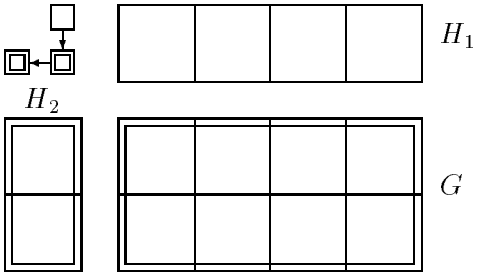
$P_T(H_1, H_2) = H_1 \wedge H_2$
$r_T(G) = G$
Relation name : R
$C_T : (R = \bigcup_{i=1}^{4} \bigcup_{j=1}^{2} R_{i,j})$

Type 4



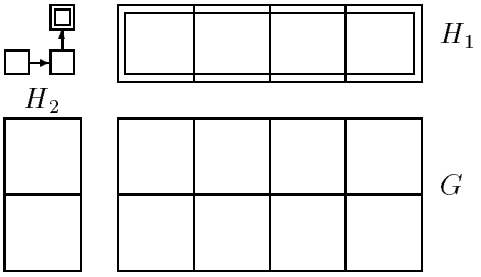
$P_T(G) = G$
$r_T(H_1, H_2) = H_1 \cup H_2$
Relation name : R
$C_T : (R = \bigoplus_{i=1}^{4} \bigotimes_{j=1}^{2} R_{i,j})$

Vector



$P_T(H_1) = H_1$
$r_T(H_2, G) = H_2 \circ G$
Relation name : R
$C_T : (R : \bigotimes_{i=1}^{4} \bigcup_{j=1}^{2} R_{i,j})$

Generalized Decision



$P_T(H_2, G) = G[H_2/\#G]$
$r_T(H_1) = H_1$
Relation name : R
$C_T : (R = \bigcup_{i=1}^{4} \bigcap_{j=1}^{2} R_{i,j})$

Figure 5: Four examples of well done table skeletons.

then we have

$$\begin{aligned} P \oplus Q &= \{((x_1,x_2,x_3),(x_4,x_5)) \mid ((x_1,x_3),x_5) \in P \vee ((x_1,x_2),x_4) \in Q\}, \\ P \otimes Q &= \{((x_1,x_2,x_3),(x_4,x_5)) \mid ((x_1,x_3),x_5) \in P \wedge ((x_1,x_2),x_4) \in Q\}, \\ P \ominus Q &= \{((x_1,x_2,x_3),(x_4,x_5)) \mid ((x_1,x_3),x_5) \in P \wedge ((x_1,x_2),x_4) \notin Q\}, \end{aligned}$$

If $J = L$ and $K = M$ then $\oplus$, $\otimes$, $\ominus$ are just $\cup$, $\cap$, and $\setminus$. The operator $\otimes$ can also be regarded as a generalization of a *natural join* operator used in relational data bases [2].

We think that in general the problem of composing R from R_α is an open research problem. The definitions of operations $\oplus$, $\otimes$, $\ominus$ were application driven. The general problem can be formulated as "how to build the whole, i.e. R, from the parts, i.e. R_α's" in terms of algebra of relations[6]

7 Tabular Expressions

We are now able to define formally the concept of table expression.

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

$$T = (P_T, r_T, C_T, CCG, H_1, ..., H_n, G; \Psi, \mathbf{IN}, \mathbf{OUT})$$

  where $(P_T, r_T, C_T, CCG, H_1, ..., H_n, G)$ is a well done table skeleton, and 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 **IN**, the relation expression have variables over $\mathbf{IN} \times \mathbf{OUT}$, where **IN** is the set (usually heterogenous product) of inputs, and **OUT** is the set (usually heterogenous product) of outputs..

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

$$Sign_T = (P_T, r_T, C_T, CCG).$$

The signature describes all the global and structural information about the table. We may say that a tabular expression is a triple: *signature*, *raw 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.

[6] We are aware of the fact that the relation *part of* is the basic notion of *Lesniewski's mereology* [31], which is a version of a set theory prposed as an antinomy-free counterpart of naive Cantor set theory. Lesniewki's systems are different than the standard Zermello-Freankel set theory. The research on how Lesniewski's approach can help has just started.

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

input variables	$x, y : Reals$
output variables	$f : Reals$
CCG	
P_T	$H_1 \wedge H_2$
r_T	G
Function name	f
C_T	$\bigcup_{i=1}^{3} \bigcup_{j=1}^{2} f_{i,j}$

H_1

$y = 10$	$y > 10$	$y < 10$

H_2

$x \geq 0$
$x < 0$

0	y^2	$-y^2$
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	$g : Reals$
CCG	
P_T	$H_2 \wedge G$
r_T	H_1
Function name	g
C_T	$\bigcup_{i=1}^{3} \bigcup_{j=1}^{2} g_{i,j}$

$x + y$	$x - y$	$y - x$

H_1

H_2

$y \geq 0$
$y < 0$

$x < 0$	$0 \leq x < y$	$x \geq y$
$x < y$	$y \leq x < 0$	$x \geq o$

G

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

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. The author thinks that precise meaning of the syntax concept for *non-linear* notations (as tables) is yet to be defined. 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

$$\begin{aligned} P_{\alpha}^{T} &= P_T[\Psi(c_1)/B_1, ..., \Psi(c_s)/B_s] \\ r_{\alpha}^{T} &= r_T[\Psi(d_1)/A_1, ..., \Psi(d_r)/A_r] \end{aligned} \tag{3}$$

where $c_i = B_i \cap Guards_{\alpha}(T)$, $i = 1, ..., s$, and $d_i = A_i \cap Values_{\alpha}(T)$, $i = 1, ..., r$.

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

input variables	$x_1, x_2 : Reals$
output variables	$y_1, y_2, y_3 : Reals$
CCG	
P_T	H_1
r_T	$H_2 \circ G$
Relation name	G
C_T	$\bigotimes_{j=1}^{3} \bigcup_{i=1}^{2} G_{i,j}$

H_2

$x_2 \leq 0$	$x_2 > 0$

H_1

G

$y_1 =$	$x_1 + x_2$	$x_1 - x_2$
$y_2 \mid$	$y_2 x_1 - x_2 = y_2^2$	$x_1 + x_2 y_2 = \lvert y_2 \rvert$
$y_3 \mid$	$y_3 + x_1 x_2 = \lvert y_3 \rvert^3$	$y_3 = x_1$

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
CCG	
P_T	$H_2 = G$
r_T	H_1
Function name	φ
C_T	$\bigcup_{i=1}^{5} \bigotimes_{j=1}^{3} \varphi_{i,j}$
notation	$* = don't\ care$

H_1

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

G

Figure 7: Next two examples of tabular expressions - vector table (above) and decision table [13] (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	$h : Reals$
CCG	
P_T	$G[H_2/\#]$
r_T	H_1
Function name	h
C_T	$\bigcup_{i=1}^{3} \bigotimes_{j=1}^{2} h_{i,j}$

H_2

$x_1 x_2$
$x_1 \div x_2$

$x_1 + x_2$	$x_1 - x_2$	$x_1 x_2$	H_1
$\# < 20$	$\# \geq 20$	$true$	G
$\# < 2$	$\# > 2$	$\# = 2$	

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

input variables	$x : Reals$
output variables	$y_1, y_2 : Reals$
CCG	
P_T	G
r_T	$H_1 \otimes H_2$
Relation name	Υ
C_T	$\bigcup_{j=1}^{3} \bigcup_{i=1}^{2} \Upsilon_{i,j}$

H_2

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

$y_1 = 0$	$y_1 = 1$
$x = 0$	$0 < x < 1$
$x < -1$	$-1 \leq x < 0$
$x = 1$	$x > 1$

H_1

G

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

Both P_T and r_T must satisfy the following *consistency* rule

- for every $\alpha \in I$, P_α^T is a syntactically correct predicate expression.
- for every $\alpha \in I$, r_α^T is a syntactically correct relation expression.

The relation composition expression C_T is built from the relation/function names, indexes, and operators $\oplus$, $\otimes$, $\ominus$, $\cup$, $\cap$, $\setminus$ are special cases, etc. The survey [1] shows that the patterns $\bigotimes_j \bigcup_i R_{i,j}$, $\bigcup_\alpha R_\alpha$, and $\bigcup_i \bigotimes_j R_{i,j}$ are sufficient in the most cases.

8 Semantics of Tabular Expressions

Let $T = (P_T, r_T, C_T, CCG, H_1, ..., H_n, G; \Psi, \mathbf{IN}, \mathbf{OUT})$ be a tabular expression, with the index I, and let $\alpha \in I$. By an *interpreted medium element* we mean a tuple:

$$T|_\alpha = (P_T, r_T, CCG_\alpha; \psi|_{Components_\alpha(T)}).$$

Figure 4 plus $P_T = H_1 \wedge H_2$, $r_T = G$, represents an example of the interpreted medium element.

For every $\alpha \in I$, we define A_α, E_α, as

$$\begin{aligned} \mathbf{x} \in A_\alpha &\iff P_\alpha(\mathbf{x}) = true, \\ (\mathbf{x}, \mathbf{y}) \in E_\alpha &\iff E_\alpha(\mathbf{x}, \mathbf{y}). \end{aligned}$$

Every interpreted medium element $T|_\alpha$ describes now the relation $R_\alpha = E_\alpha | A_\alpha$, i.e.

$$(\mathbf{x}, \mathbf{y}) \in R_\alpha \iff \textbf{if } P_\alpha(\mathbf{x}) \textbf{ then } E_\alpha(\mathbf{x}, \mathbf{y}).$$

We may now define the semantics of tabular expressions in a formal way:

- The relation R_α describes the semantics of the *interpreted medium skeleton* $T|_\alpha$.
- The *semantics* of a *tabular expression* T is defined by:

$$R_T = C_T(R_\alpha).$$

Figures 6, 7 and 8 illustrate the above definitions.

9 On Table Classification

Tabular expressions can be classified according to:

- cell connection graph, CCG,
- table composition rule, C_T,
- table predicate and relation rules, P_T and r_T,
- the mapping Ψ which assigns meanings to the cells.

In most cases we do not provide a complete classification, rather some special cases are chosen and named.

The table classification according to CCG is presented in Figure 2. The type 1 is standardly called *normal*, and type 2a is called *inverted*. The relationship between normal and inverted table is analyzed in detail in [32].

The table is called *plain* if C_T defines the plain representation of R_T , i.e. if $R_T = \bigcup_{\alpha \in I} R_\alpha$. The table is called *output-vector* if $R_T = \bigotimes_j \bigoplus_i R_{i,j}$. The table is called *input-vector* $R_T = \bigoplus_i \bigotimes_j R_{i,j}$. All tables modeled in [14] are plain. The vector tables of [24] are of output-vector type, the most of (but not all) decision tables [12, 13, 24] are of input-vector type.

The classification according to P_T and r_T has not yet been proposed. Since the most popular type of P_T (see [1]) is conjunction, followed by disjunction [27], equality, and replacement $E[E'/\#]$ [1, 24], the *disjunctive tables*, *conjunctive tables*, *equality tables*, and *#-replacement tables* are natural candidates for special table types.

The classification on the basis of Ψ is a different type of classification than each of the above. It depends on what the contents of cells is, and is not the subject of this paper. The division of tables into function, relation and predicate types, as well as into proper and improper [24, 30, 32], is based on Ψ. Some popular types as vector, decision, and generalized decision require the special type of Ψ, the special type of C_T and the special type of P_t and/or r_T.

10 Final Comment

The model covers all types of tables currently used in Software Engineering. It also allows us to define precisely new types tables. The specific forms of Ψ are not the subject of this paper, so we do not make any distinction between function and relation tables and between proper and not proper tables [24]. Of course, for an efficient use of tables some classification according to Ψ is necessary. In particular the distinction between functions and relations is important from the application point of view, since the properties are different in many aspects. However it should

be done *after* the general semantics is precisely defined, so in this paper we do not touch this problem.

The approach presented here is complementary to that of [24]. The classification provided in [24] was based on several years of practical experience of using tables for specifying real computing systems. The classification provided in this paper follows from the topology of an abstract entity called 'table', and per se, is application independent. In fact, some possibilities allowed by this approach might have rather rare applications.

In principle, the approach presented in this paper is based on the following concepts

- the cell connection relation $\longmapsto$, which represents the information flow in tables,
- division of table components into $Guards(T)$ and $Values(T)$,
- P_T, the table predicate rule,
- r_T, the table relation rule,
- C_T, the table composition rule.

So far, in our approach, not much assumption about the forms of P_T and r_T is made. This is an area for further development, since not all forms of P_T and r_T make practical sense. When real examples are analyzed (see for instance [1, 26]), one may observe that in many cases the distribution of input and output variables among headers is not arbitrary, but on purpose (see top table in Figure 6 and bottom table in Figure 7). This problem is also not addressed here. The cases $n = 2$ and $n = 3$ need special attention since they will eventually be the most frequently used in practice. Finally, concurrency, non-determinism and the concept of time, are not addressed yet. We believe the approach of [16] might be useful here.

An algebra of arrays of relations has been used in [5] to present somewhat alternative semantics for tabular expression.

Acknowledgments

Dave Parnas provided both inspiration and initial attempt to formalize the concept of tables. If not [24] this paper would have never been written. 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.

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

[3] 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.

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

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

[6] GARD Research Consulting Inc., Software Requirements for AECB Project 2.314.1, 23141-DOC-4, Revision Draft 1, 1994.

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

[8] 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.

[9] 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.

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

[11] 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.

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

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

[14] 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.

[15] 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

[16] R. Janicki, M. Koutny, Structure of Concurrency, *Theoretical Computer Science*, 112 (1993), 5-52.

[17] 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.

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

[19] 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.

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

[21] 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.

[22] 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.

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

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

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

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

[27] B. Plenderleith, Care and Feeding of Living Software Documentation, Lecture at Workshop on Tools for Tabular Notation, McMaster University, Hamilton, Ontario, Canada 1996.

[28] G. Schmidt, T. Ströhlein, Relations and Graphs. Discrete Mathematics for Computer Scienctistc, Springer 1993.

[29] 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.

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

[31] J. Strzednicki, V.F. Rickey (eds.), Lesniewski's Systems, Kluwer Academic, 1984.

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