On a Formal Semantics of Tabular Expressions

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

Abstract

In [15, 22, 25, 26] 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 [23] 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.

Contents

1 Introduction **2**

2 Introductory examples **4**

3 Relations **9**

3.1 Cartesian Products, Functions, Relations 9

3.2 Direct Products 10

3.3 Input-Output Relations 12

4 Raw Table Skeleton **14**

5 Cell Connection Graph and Medium Table Skeleton **15**

6 Raw and Medium Table Elements **19**

*Supported by NSERC of Canada Grant

7 Well Done Table Skeleton **20**

8 Tabular Expressions **24**

9 Semantics of Tabular Expressions **27**

10 On Table Classification **28**

11 Final Comment **29**

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 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 [17] 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. 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 [15].

The key ideas of a *tabular notation*, one of the cornerstones of the relational model for documenting the intended behaviour of programs [15, 22, 25, 26], were first

developed in work for the U.S. Navy and applied to the A-7E aircraft [10, 9, 4, 28]. 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, 20, 21].

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 [24].

The papers [25, 26] 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 [23] describes several different classes of tables and provides their *formal* syntax and semantics. All classes considered in [23] were invented for some specific practical applications. Formal relationship between some important classes of tables has been analyzed in [30]. The overall methodology and recent results of the tabular approach are presented in [15].

The tabular notation is currently used among others by the Software Engineering Research Group (SERG) at McMaster University, Hamilton, Ontario, Canada [29], Ontario Hydro [19], Naval Research Laboratory [8], ORA Inc., [11], and University of California at Irvine [7, 18].

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 [23]), we shall introduce only one general definition of tables, each class of [23] could be derived as a special case. The model will also indicate the other, not considered in [23], 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. For more realistic examples (as loop invariants, program specifications) the reader is referenced to [1, 29, 26].

The next section contains some introductory examples. Some new, table oriented, operations on sets being Cartesian products are introduced and discussed in Chapter 3. The first rough approximation of the table concept is given in Chapter 4. The crucial concept of *Cell Connection Graph* and more precise approximations of the table concept are discussed in Chapters 5, 6 and 7. The formal definition of a *tabular expression* on syntactic level is given in Chapter 8, and its *semantics* is discussed in Chapter 9. Chapter 10 contains some comments on table classification, and final, comments are in Chapter 11.

The paper is a revised and full version of [13].

2 Introductory examples

Let us consider the following definition of a function

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

This is the classical mathematical notation which allows sometimes to relax the linearity principle. Lamport [17] proposes similar relaxation rules for more complex cases. In the purely linear notation, which is unfortunately very popular in various specification techniques (just pick a case!), the definition of the function f looks like:

$$\begin{array}{l} f(x,y) = \ \textbf{if } x \geq 0 \wedge y = 10 \textbf{ then } 0 \\ \textbf{else if } x < 0 \wedge y > 10 \textbf{ then } x \\ \textbf{else if } x \geq 0 \wedge y > 10 \textbf{ then } y^2 \\ \textbf{else if } x \geq 0 \wedge y < 10 \textbf{ then } -y^2 \\ \textbf{else if } x < 0 \wedge y < 10 \textbf{ then } x + y \\ \textbf{else if } x < 0 \wedge y > 10 \textbf{ then } x - y \end{array}$$

and is less readable than the classical mathematical notation. Nevertheless the *most* readable definition is that represented in Figure 1, where the concept of a *table* is used. Consider now the function g defined as

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

Again, this not very readable description becomes very clear and obvious when the concept of an (inverted) *table* is used (see Figure 2).

The table in Figure 3 defines the following *relation* $G \subseteq \mathbf{IN} \times \mathbf{OUT}$, where $\mathbf{IN} = \mathit{Reals} \times \mathit{Reals}$, $\mathbf{OUT} = \mathit{Reals} \times \mathit{Reals} \times \mathit{Reals}$, x_1, x_2 are the variables over $\mathbf{IN}$, y_1, y_2, y_3 are variables over $\mathbf{OUT}$, and

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

$$\left\{ \begin{array}{l} y_1 = x_1 - x_2 \ \wedge \ x_1 + x_2 + x_2 y_2 = |y_2| \\ \wedge \ y_3 = x_1 \end{array} \right\} \text{ if } x_2 > 0$$

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

Figure 1: The function f defined by a (normal) table.

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

Figure 2: The function g defined by an (inverted) table.

while the table in Figure 4 defines the function φ : Temperature $\times$ Weather $\times$ Windy $\rightarrow$ *Activities*, where *Activities* = { go sailing, go to the beach, play bridge, garden}. The table from Figure 4 is called a *decision table*, and such tables have been used as a specification tools since fifties [11, 12].

Figure 5 contains a *generalized decision table* [1, 23]. It represents the function $h : Reals \times Reals \rightarrow Reals$ defined as

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

Although the tables from Figures 1 - 5 are of different types, they have some elements in common. All global functions specified by these tables: f, g, h, and φ are built from the simpler "atomic" functions.

The function f is a *composition* of "atomic representations", $f_{i,j}$, $i = 1, 2, 3$, $j = 1, 2$, where for example $f_{3,2} : (-\infty, 0) \times (10, +\infty) \rightarrow Reals$, and $f_{3,2}(x, y) = x - y$ for $(x, y) \in dom(f_{3,2})$.

	$x_2 \leq 0$	$x_2 > 0$
$y_1 =$	$x_1 + x_2$	$x_1 - x_2$
$y_2 \mid$	$y_2 x_1 - x_2 = y_2^2$	$x_1 + x_2 y_2 = \lvert y_2 \rvert$
$y_3 \mid$	$y_3 + x_1 x_2 = \lvert y_3 \rvert^3$	$y_3 = x_1$

Figure 3: The relation G defined by a (vector) table.

	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 $\in \{true, false\}$	$true$	$false$	$false$	$*$	$false$

$* = don't\ care$

Figure 4: The function φ defined by a (decision) table (from [11]).

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

Figure 5: The function h defined by a (generalized decision) table.

Similarly g is a composition of $g_{i,j}$, $i = 1, 2, 3$, $j = 1, 2$, and for example $dom(g_{1,2}) = \{(x, y) \mid 0 \leq x < y\}$, $g_{1,2}(x, y) = x - y$. The functions f and g are *unions* of their "atomic" representations, i.e.

$$f = \bigcup_{i \in \{1,2,3\} \wedge j \in \{1,2\}} f_{i,j} \quad \text{and} \quad g = \bigcup_{i \in \{1,2,3\} \wedge j \in \{1,2\}} g_{i,j}.$$

The relation G is a composition of "atomic" representations $G_{i,j}$, $i = 1, 2$, $j = 1, 2, 3$, and for instance $G_{1,3} \subseteq IN_{1,3} \times OUT_{1,3}$, where $IN_{1,3} = Reals \times (-\infty, 0\rangle$, $OUT_{1,3} = Reals$, and

$$(x_1, x_2) G_{1,3} y_3 \iff y_3 + x_1 x_2 = |y_3|^3.$$

The relation $G_{1,1}$ is a function $G_{1,1} : IN_{1,1} \to OUT_{1,1}$, with $IN_{1,1} = IN_{1,3}$ and $OUT_{1,1} = Reals$, and

$$(x_1, x_2) G_{1,1} y_1 \iff y_1 = G_{1,1}(x_1, x_2) = x_1 + x_2.$$

The relations $G_{i,1}$ are functions, which is indicated by the symbol "=" after variable y_1 in the left header. The symbol "|" after y_2 and y_3 indicates that $G_{i,2}$ and $G_{i,3}$ are relations with y_2 and y_3 as a respective output variables.

The function φ is a composition of $\varphi_{i,j}$, $i = 1, ..., 5$, $j = 1, 2, 3$, and for instance $\varphi_{2,2} : \{sunny\} \to \{\text{go to the beach}\}$, $\varphi_{2,2}(sunny) = \text{go to the beach}$. The domain of $\varphi_{2,2}$ is $\{sunny\}$ rather than $\{sunny, cloudy, rain\}$ since we prefer to deal with total functions.

The function h is a composition of $h_{i,j}$, $i = 1, 2, 3$, $j = 1, 2$. For instance $h_{1,2} : IN_{1,2} \to Reals$, where $IN_{1,2} = \{(x_1, x_2) \mid x_1 x_2 > 20\}$ and $h_{1,2}(x_1, x_2) = x_1 - x_2$,

while $h_{1,3} : Reals \times Reals \to Reals$ and $h_{1,3}(x_1, x_2) = x_1 x_2$.

However the functions h, φ and the relation G are *not* the unions of $h_{i,j}$'s, $\varphi_{i,j}$'s and $G_{i,j}$'s. We have here

$$h = \bigcup_{i=1}^{3} \bigcap_{j=1}^{2} h_{i,j},$$

and we will show that

$$G = \bigotimes_{j=1}^{3} \bigcup_{i=1}^{2} G_{i,j}, \quad \varphi = \bigcup_{i=1}^{5} \bigotimes_{j=1}^{3} \varphi_{i,j}, \quad \textit{and} \quad h = \bigcup_{i=1}^{3} \bigotimes_{j=1}^{2} h_{i,j},$$

where $\otimes$ is an operator, a generalization of *both* the intersection and the well-known "join" from the relational data-base theory [2]. The operator $\otimes$ is discussed in details in the next section.

Each of the tables from Figures 1 - 5 consists of two one dimensional *headers* (top row and left-hand column), and one two dimensional *grid*. Both headers and grids consist of *cells*, each cell contains an *expression*. The *atomic representations* of functions and the relation from Figures 1 - 5 are defined by parts of tables we will call *raw elements*. The raw element is just a table restricted to one cell for each header and one cell for the grid (see Figure 6).

Every *atomic representation* $R_{i,j}$ can be represented by the relation/function expression of the type

$$\mathbf{x} R_{i,j} \mathbf{y} \iff \textbf{if } P_{i,j}(\mathbf{x}) \textbf{ then } R_{i,j}(\mathbf{x}, \mathbf{y}),$$

where $\mathbf{x}$ is the (vector) *input variable*, $\mathbf{y}$ is the (vector) *output variable*, $P_{i,j}(\mathbf{x})$ is the *predicate expression* built from the *guard* expressions held in *guard cells*, and $R_{i,j}(\mathbf{x}, \mathbf{y})$ is the expression defining a *function/relation* and is built form the *value* expressions held in *value* cells. For example for $G_{1,3}$ we have $P_{1,3}(x) = x_2 \leq 0$, $R_{1,3}(\mathbf{x}, y_3)$ equals to $y_3 + x_1 x_2 = |y_3|^3$, where $\mathbf{x} = (x_1, x_2)$, for $h_{1,2}$ we have $P_{1,2}(x) = x_1 x_2 \geq 20$, $R_{1,2}(\mathbf{x}, y)$, where $\mathbf{x} = (x_1, x_2)$, equals to $y = x_1 - x_2$ (see Figure 6).

For each table and each header or grid, either all cells contain guard expressions, or all contain value expressions. If the header or grid contains value cells it has *double border lines*.

All the above observations will be used to build a homogeneous semantics for (almost) all possible types of tables.

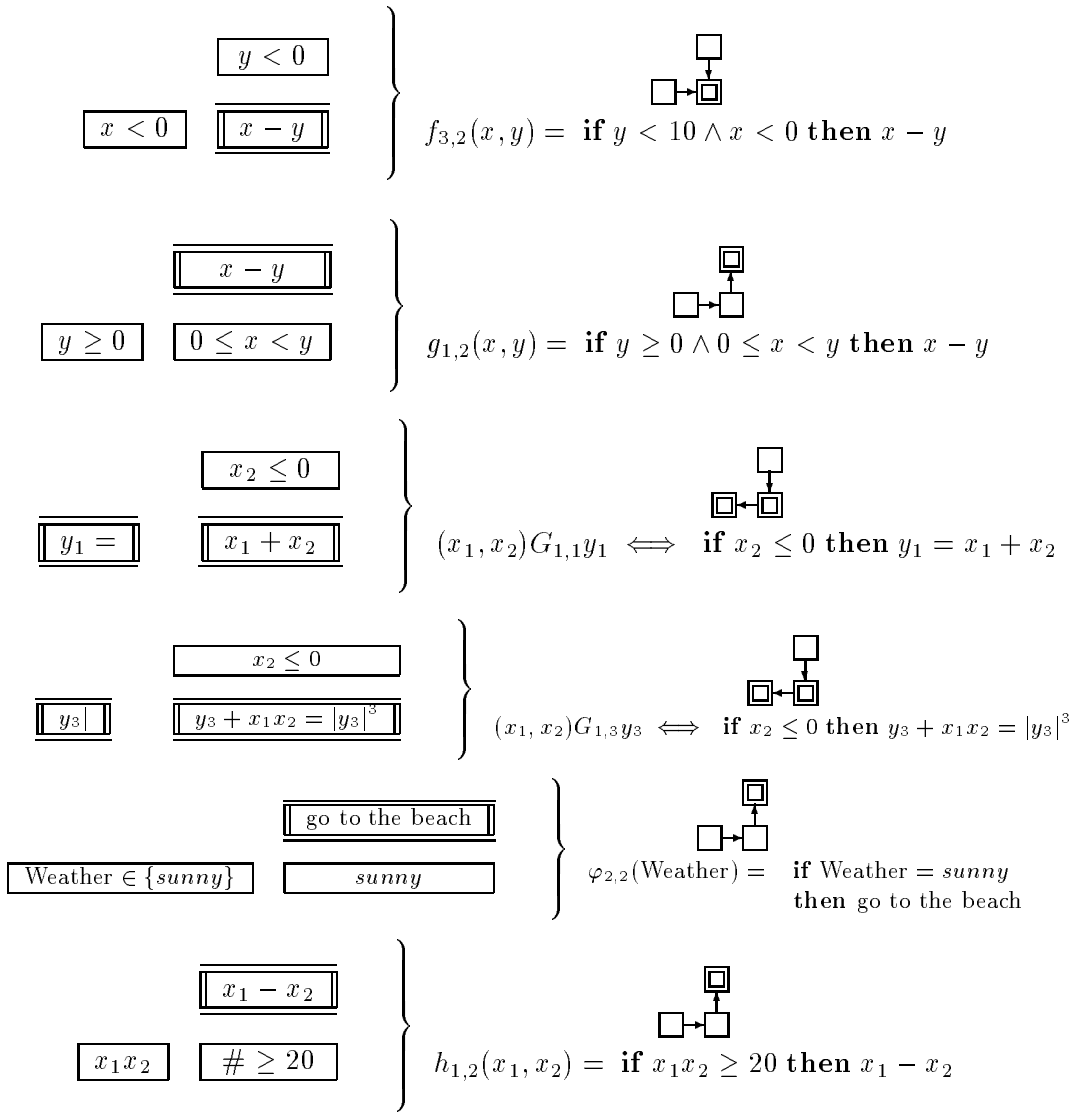


Figure 6: The functions $f_{3,2}$, $g_{1,2}$, $\varphi_{2,2}$, $h_{1,2}$, the relations $G_{1,1}$, $G_{1,3}$ and their appropriate raw elements, **if-then** descriptions, and cell connection graphs.

3 Relations

The tables were designed to represent *relations*, with *functions* as a special case. Formally, a *relation* R is a subset of *Cartesian product* of the set X and the set Y. The concept of Cartesian product has *two* equivalent but *different* meanings in mathematics [6, 16], and we will make use of this differentiation in this paper.

The relations/functions that are represented by tables are defined on $\mathbf{IN} \times \mathbf{OUT}$, where **IN** and **OUT** are the sets of input and output values respectively. However the sets **IN** and **OUT** are frequently themselves the products of different domains, i.e. usually $\mathbf{IN} = IN_1 \times ... \times IN_n$ and $\mathbf{OUT} = OUT_1 \times ... \times OUT_m$. The relation/function R_T defined by the table T is usually built from the relations R_α, $\alpha \in I$, I is an index set, where each $R_\alpha \subseteq IN_\alpha \times OUT_\alpha$ with $IN_\alpha = IN_{i_1} \times ... \times IN_{i_\alpha}$, $OUT_\alpha = OUT_{j_1} \times ... \times OUT_{j_\alpha}$. The standard set notation is not very convenient to handle such cases. We will extend it below.

3.1 Cartesian Products, Functions, Relations

We define the *Cartesian product* of two sets X and Y, as

$$X \times Y = \{(x, y) \mid x \in X \wedge y \in Y\}.$$

This definition can easily be extended to a finite family of sets, i.e.

$$X_1 \times ... \times X_m = \{(x_1, ..., x_m) \mid x_i \in X_i \text{ for } i = 1, ..., m\},$$

however for $n > 2$ we would prefer the concept of *direct product* defined in the next subsection.

By a *relation* R we mean any subset of the Cartesian product $X \times Y$, i.e. $R \subseteq X \times Y$.
For every relation $R \subseteq X \times Y$, we define:

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

The relation $R \subseteq X \times Y$ is *total* if $dom(R) = X$.

We shall frequently write $R(x, y)$ to denote an *expression* defining R, i.e. $(x, y) \in R \iff R(x, y)$.
A *function* $f : X \to Y$ is a relation $f \subseteq X \times Y$ satisfying:

$$\forall x \in X.\, (x, y) \in f \wedge (x, z) \in f \;\; \Rightarrow \;\; y = z.$$

Since every function is a relation the concepts of *totality*, *dom* and *range* defined above hold for the functions as well.

For every set $A \subseteq X$ and every relation $R \subseteq X \times Y$ we define a relation $A \hookrightarrow R \subseteq X \times Y$, i.e. $\hookrightarrow$ is an operator with A and R as left and right arguments, such that:

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

In other words, if $P_A(x)$ is a predicate that describes the set A, i.e. $x \in A \iff P_A(x) = true$, and $E_R(x,y)$ is a relational expression that defines the relation R, i.e. $(x,y) \in R \iff E_R(x,y)$, then the relation $A \hookrightarrow R$ is described by the expression **if** $P_A(x)$ **then** $E_R(x,y)$ (see Footnote 3 in Chapter 7).

3.2 Direct Products

For every function $f : X \to Y$ and every $Z \subseteq X$, we define the *restriction of* f *to* Z, $f|_Z : Z \to Y$ as $f|_Z(x) = f(x)$ for every $x \in Z$.

Let $\{X_t \mid t \in T\}$ be a family of sets. By a *direct* product, $\prod_{t \in T} X_t$, we mean the set of all functions $f : T \to \bigcup_{t \in T} X_t$ such that

$$\forall t \in T.\ f(t) \in A_t.$$

Technically the sets $X_1 \times X_2$ and $\prod_{i \in \{1,2\}} X_i$ are different. The first is the set of pairs, the second is the set of functions. However they can be seen as equivalent (they are *isomorphic*). Every pair $(x,y) \in X_1 \times X_2$ corresponds to the function $f_{x,y} \in \prod_{i \in \{1,2\}} X_i$ such that $f_{x,y}(1) = x$ and $f_{x,y}(2) = y$. Every function $f \in \prod_{i \in \{1,2\}} X_i$ corresponds to the pair $(f(1), f(2)) \in X_1 \times X_2$. If $f(1) = x$ and $f(2) = y$, then the function f is just a set of pairs $\{(1,x),(2,y)\} \subseteq \{1,2\} \times X_1 \cup X_2$. Hence we can say that $(x,y) \in X_1 \times X_2 \iff \{(1,x),(2,y)\} \in \prod_{i \in \{1,2\}} X_i$.

To clarify the concepts discussed above let us consider the following example. Let $X_1 = \{a,b,c\}$, $X_2 = \{c,d,e\}$, $T = \{1,2\}$. Then

$$\begin{aligned} X_1 \times X_2 = \{&(a,c),(a,d),(a,e),\\ &(b,c),(b,d),(b,e),\\ &(c,c),(c,d),(c,e)\} \end{aligned}$$

The direct product $\prod_{i \in \{1,2\}} X_i$ is the set of all functions $f : \{1,2\} \to \{a,b,c,d,e\}$ such that for all $t \in \{1,2\}$ $f(t) \in X_t$, i.e.

$$\begin{aligned} \prod_{i \in \{1,2\}} X_i = \{&\{(1,a),(2,c)\},\{(1,a),(2,d)\},\{(1,a),(2,e)\},\\ &\{(1,b),(2,c)\},\{(1,b),(2,d)\},\{(1,b),(2,e)\},\\ &\{(1,c),(2,c)\},\{(1,c),(2,d)\},\{(1,c),(2,e)\}\} \end{aligned}$$

The construction $\prod$ is more general. The set T may not be countable, and the construction can easily be extended to more complex structures as algebras or automata. If the members of T do not have an inherent order, for instance $T = \{a, b\}$, then the construction $\prod$ can be interpreted that the order of coordinates is not important. We shall make use of it later. If $T = \{1, 2, ..., m\}$ we may write $X_1 \times ... \times X_m$ instead of $\prod_{t \in T} X_t$, if it will not lead to any misunderstanding. If $T = \{1, 2, ..., m\}$ and $X_t = X$ for all $t \in T$, we shall write X^m.

Let $f \in \prod_{t \in T} X_t$, $Y \subseteq \prod_{t \in T} X_t$, $S \subseteq T$, $S \neq \emptyset$. We define the *projections of* f and Y *onto* S, $\pi_S(x) \in \prod_{t \in S} X_t$ and $\pi_S(Y) \subseteq \prod_{t \in S} X_t$, as:

$$\pi_S(f) = f|_S \quad \text{and} \quad \pi_S(Y) = \{g|_S \mid g \in Y\}.$$

In other words $\pi_S(Y) = \{f \mid f : S \to \bigcup_{t \in S} X_t \wedge \exists g \in Y.\, g|_S = f\}$. For example $\pi_{\{i_1,...,i_k\}}((a_1, ..., a_m)) = (a_{i_1}, ..., a_{i_k})$ and $\pi_{\{i_1,...,i_k\}}(X_1 \times ... \times X_m) = X_{i_1} \times ... \times X_{i_k}$, if $\{i_1, ..., i_k\} \subseteq \{1, 2, ..., m\}$, $a_i \in X_i$, for $i = 1, ..., m$.
We shall write $\pi_t(Y)$ instead of $\pi_{\{t\}}(Y)$ for $t \in T$, and then $\pi_t(Y)$ is a *projection of* Y *on the* t*-th coordinate*. All the above definitions are classical [6, 16]. The new concepts are below.

Let $A \subseteq \prod_{t \in T} X_t$, and B be a set. We write

$$B \sqsubseteq A \iff \exists S \subseteq T.\, B \subseteq \pi_S(A).$$

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

Let $\{\mathcal{U}_t \mid t \in \mathcal{T}\}$ be a family of sets, and let $\mathcal{U} = \prod_{t \in \mathcal{T}} \mathcal{U}_t$ be a *direct product* fixed for the rest of this section.
Let $\mathcal{C}(\mathcal{U})$ denote the set of all A such that $A \sqsubseteq \mathcal{U}$, i.e. $\mathcal{C}(\mathcal{U}) = \{A \mid A \sqsubseteq \mathcal{U}\}$. Clearly $2^{\mathcal{U}} \subseteq \mathcal{C}(\mathcal{U})$, and if $|\mathcal{T}| = 1$ then $2^{\mathcal{U}} = \mathcal{C}(\mathcal{U})$.

For every $A \in \mathcal{C}(\mathcal{U})$, the set $\tau(A) \subseteq \mathcal{T}$, is defined as the subset of $\mathcal{T}$ such that $A \subseteq \prod_{t \in \tau_{\mathcal{U}}(A)} \mathcal{U}_t$. The set $\tau(A)$ will be called the *Cartesian index* of A in $\mathcal{U}$. Very often we shall write $A^{(S)} \in \mathcal{C}(\mathcal{U})$ to indicate that $\tau(A) = S$. For example, if $\mathcal{U} = \mathcal{U}_1 \times \mathcal{U}_2 \times \mathcal{U}_3$, and $A \subseteq \mathcal{U}_1 \times \mathcal{U}_2$, $B \subseteq \mathcal{U}_1 \times \mathcal{U}_3$ then $\tau(A) = \{1, 2\}$, $\tau(B) = \{1, 3\}$.

We define the two important operations on the elements of $\mathcal{C}(\mathcal{U})$. They can be seen as a generalization of $\cup$, $\cap$ and $\setminus$. Let:

$$\begin{aligned} A^{(S)} \oplus B^{(P)} &= \{x \in \textstyle\prod_{t \in S \cup P} \mathcal{U}_t \mid \pi_S(x) \in A \vee \pi_P(x) \in B\}, \\ A^{(S)} \otimes B^{(P)} &= \{x \in \textstyle\prod_{t \in S \cup P} \mathcal{U}_t \mid \pi_S(x) \in A \wedge \pi_P(x) \in B\}, \\ A^{(S)} \ominus B^{(P)} &= \{x \in \textstyle\prod_{t \in S \cup P} \mathcal{U}_t \mid \pi_S(x) \in A \wedge \pi_P(x) \notin B\}. \end{aligned}$$

Clearly $\tau(A \oplus B) = \tau(A \otimes B) = \tau(A) \ominus \tau(B) = \tau(A) \cup \tau(B)$. If $\tau(A) \neq \tau(B)$ then $A \cup B \notin \mathcal{C}(\mathcal{U})$, and $A \cap B \notin \mathcal{C}(\mathcal{U})$, but always $A \oplus B \in \mathcal{C}(\mathcal{U})$ and $A \otimes B \in \mathcal{C}(\mathcal{U})$. If $\tau(A) = \tau(B)$ then $A \oplus B = A \cup B$, $A \otimes B = A \cap B$, and $A \ominus B = A \setminus B$. If $\tau(A) \cap \tau(B) = \emptyset$ then $A \otimes B = A \times B$, and $A \oplus B = A \times \prod_{t \in P} \mathcal{U}_t \cup \prod_{t \in S} \mathcal{U}_t \times B$. If $\tau(A) \cap \tau(B) \neq \emptyset$ and $\tau(A) \neq \tau(B)$ then $A \otimes B = A \bowtie B$, where $\bowtie$ is a *natural join* operation used in relational data bases [2].

Let $A \subseteq \prod_{i \in \{1,3,5\}} \mathcal{U}_i$, $B \subseteq \prod_{i \in \{1,2,4\}} \mathcal{U}_i$ (i.e. $A \subseteq \mathcal{U}_1 \times \mathcal{U}_3 \times \mathcal{U}_5$, $B \subseteq \mathcal{U}_1 \times \mathcal{U}_2 \times \mathcal{U}_4$). Then

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

Lemma 3.1

1. $A \otimes (B \oplus C) = (A \otimes B) \oplus (A \otimes C)$
2. $A \oplus (B \otimes C) = (A \oplus B) \otimes (A \oplus C)$
3. $A^{(S)} \otimes (B^{(P)} \cup C^{(P)}) = (A^{(S)} \otimes B^{(P)}) \cup (A^{(S)} \otimes C^{(P)})$.

■

The easy standard proof is left to the reader.

3.3 Input-Output Relations

Let $\mathbf{IN} = \prod_{t \in \mathcal{IN}} X_t$ be a set of *inputs*, and $\mathbf{OUT} = \prod_{t \in \mathcal{OUT}} Y_t$ be the set of *outputs*. Without loss of generality we may assume that $\mathcal{IN} \cap \mathcal{OUT} = \emptyset$. Any *total relation* $R \subseteq \mathbf{IN} \times \mathbf{OUT}$ is called an *Input/Output relation* (*I/O-relation*). The sets X_t, $t \in \mathcal{IN}$ are called *input domains* while the sets Y_t, $t \in \mathcal{OUT}$ are called *output domains*. The relation R is total, i.e. $dom(R) = \prod_{t \in \mathcal{IN}} X_t$. It might happen that $|\mathcal{IN}| = 1$ and/or $|\mathcal{OUT}| = 1$. Tabular expressions were designed as a tool to specify some kind of Input/Output relations.

The set $\mathbf{IN} \times \mathbf{OUT}$ is equivalent to $\prod_{t \in \mathcal{IN} \cup \mathcal{OUT}} Z_t$, where $Z_t = X_t$ if $t \in \mathcal{IN}$, and $Z_t = Y_t$ if $t \in \mathcal{OUT}$, so the operations $\oplus$, $\otimes$, and $\ominus$ and the rest of concepts introduced in the previous section can be used.

Let $\mathbf{C}(\mathbf{IN} \times \mathbf{OUT})$ be the subset of $\mathcal{C}(\mathbf{IN} \times \mathbf{OUT})$ defined as

$$R \in \mathbf{C}(\mathbf{IN} \times \mathbf{OUT}) \iff \exists A \sqsubseteq \mathbf{IN}. \exists B \sqsubseteq OUT.\, R \subseteq A \times B.$$

In other words $R \in \mathbf{C}(\mathbf{IN} \times \mathbf{OUT}) \iff R \sqsubseteq \mathbf{IN} \times \mathbf{OUT} \wedge \tau(R) \cap \mathcal{IN} \neq \emptyset \wedge \tau(R) \cap \mathcal{OUT} \neq \emptyset$.

Let $\mathcal{F} = \{R_\alpha \mid \alpha \in I\}$ be a finite set of relations from $\mathbf{C}(\mathbf{IN} \times \mathbf{OUT})$, and let $Closure(\mathcal{F})$ be the set of all relations that can be built from the elements of $\mathcal{F}$ by using the operations $\oplus$, $\ominus$, and $\otimes$. One may show that the set $Closure(\mathcal{F})$ is finite and has a normal form. The proof is a copy of the similar proof for the sets closed under the operations $\cup$, $\setminus$ and $\cap$ (see [16]).

A family of relations $\mathcal{F} = \{R_\alpha \mid \alpha \in I\} \subseteq \mathbf{C}(\mathbf{IN} \times \mathbf{OUT})$ is a *representation* of the relation $R \in \mathbf{C}(\mathbf{IN} \times \mathbf{OUT})$ if and only if $R \in Closure(\mathcal{F})$. We shall write $R = Expr(\mathcal{F})$ to denote that R is represented by an expression built form the elements of $\mathcal{F}$ and the operators $\oplus$, $\ominus$, and $\otimes$.

Let $R \subseteq \mathbf{IN} \times \mathbf{OUT}$ be an I/O-relation. Out of many possible representations, we will distinguish four special types: *local*, *plain*, *input-vector* and *output-vector*.
A representation $\mathcal{F} = \{R_\alpha \mid \alpha \in I\}$ of R is called *local* if $R_\alpha \sqsubseteq R$ for all $\alpha \in I$.
A representation $\mathcal{F} = \{R_\alpha \mid \alpha \in I\}$ of R is called *plain* if

$$\forall \alpha \in I.\, R_\alpha \sqsubseteq \mathbf{IN} \times \mathbf{OUT} \text{ and } R = \bigcup_{\alpha \in I} R_\alpha = \bigoplus_{\alpha \in I} R_\alpha.$$

In this case

$$Expr(\mathcal{F}) = \bigoplus_{\alpha \in I} R_\alpha.$$

A representation $\mathcal{F} = \{R_\alpha \mid \alpha \in I\}$ of R is called *input-vector* if $I = \mathcal{IN} \times J$ (or $I = J \times \mathcal{IN}$, the order does not matter, it may differ for different applications) and

$$\forall (t, \beta) \in I.\, R_{(t,\beta)} \subseteq X_t \times \mathbf{OUT} \text{ and } R = \bigoplus_{\beta \in J} \bigotimes_{t \in \mathcal{IN}} R_{(t,\beta)} = \bigcup_{\beta \in J} \bigotimes_{t \in \mathcal{IN}} R_{(t,\beta)}.$$

In this case we have

$$Expr(\mathcal{F}) = \bigoplus_{\beta \in J} \bigotimes_{t \in \mathcal{IN}} R_{(t,\beta)}.$$

A representation $\mathcal{F} = \{R_\alpha \mid \alpha \in I\}$ of R is called *output-vector* if $I = J \times \mathcal{IN}$ (or $I = \mathcal{OUT} \times J$, the order does not matter, it may differ for different applications) and

$$\forall (\beta, t) \in I.\, R_{(\beta,t)} \subseteq \mathbf{IN} \times Y_t \text{ and } R = \bigotimes_{t \in \mathcal{IN}} \bigoplus_{\beta \in J} R_{(\beta,t)} = \bigotimes_{t \in \mathcal{IN}} \bigcup_{\beta \in J} R_{(\beta,t)}.$$

In this case we have

$$Expr(\mathcal{F}) = \bigotimes_{t \in \mathcal{IN}} \bigoplus_{\beta \in J} R_{(\beta,t)}.$$

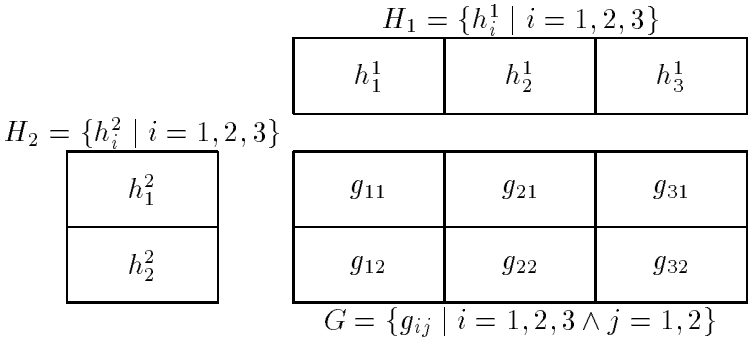


Figure 7: An example of a raw table skeleton $T = (H_1, H_2, G)$.

Every plain representation is local, but both input-vector and output-vector representations might not be local.

The representation $\mathcal{F}_G = \{G_{i,j} \mid (i,j) \in \{1,2\} \times \{1,2,3\}\}$ of the relation G from Figure 3 is *output-vector* representation. In this case $\mathcal{OUT} = \{1,2,3\}$, $J = \{1,2\}$, $\mathbf{IN} = Reals^2$, $\mathbf{OUT} = Reals^3$, and $G_{i,j} \subseteq \mathbf{IN} \times \pi_j(\mathbf{OUT})$.

The representations $\mathcal{F}_\varphi = \{\varphi_{i,j} \mid (i,j) \in \{1,...,5\} \times \{1,2,3\}\}$ of the function φ from Figure 4 and $\mathcal{F}_h = \{h_{i,j} \mid (i,j) \in \{1,2\} \times \{1,2,3\}\}$ of the function h from Figure 5 are *input-vector* representations. For $\mathcal{F}_\varphi$ we have here $\mathcal{IN} = \{1,2,3\}$, $J = \{1,2,3,4,5\}$, $\mathbf{IN} = \{hot, cool\} \times \{sunny, cloudy, rain\} \times \{true, false\}$, $\mathbf{OUT} =$ {go sailing, go to the beach, play bridge, garden}, and each $\varphi_{i,j} \subseteq \pi_i(\mathbf{IN}) \times \mathbf{OUT}$. For $\mathcal{F}_h$ we have $\mathcal{IN} = \{1,2\}$, $J = \{1,2,3\}$, $\mathbf{IN} = Reals^2$, $\mathbf{OUT} = Reals$, and each $h_{i,j} \subseteq \pi_i(\mathbf{IN} \times \mathbf{OUT} = \mathbf{IN} \times \mathbf{OUT}$.

The representations $\mathcal{F}_f = \{f_{i,j} \mid (i,j) \in \{1,2,3\} \times \{1,2\}\}$ and $\mathcal{F}_g = \{g_{i,j} \mid (i,j) \in \{1,2,3\} \times \{1,2\}\}$ of the functions f and g from Figures 1 and 2 are *plain*. For the function f we have $\mathcal{IN} = \{1,2\}$, $\mathcal{OUT} = \{1\}$, $\mathbf{IN} = Reals^2$, $\mathbf{OUT} = Reals$, and each $f_{i,j} \subseteq \mathbf{IN} \times \mathbf{OUT}$, while for the function g, we have $\mathcal{IN} = \{1,2\}$, $\mathcal{OUT} = \{1\}$, $\mathbf{IN} = Reals^2$, $\mathbf{OUT} = Reals$, and each $g_{i,j} \subseteq \mathbf{IN} \times \mathbf{OUT}$.

4 Raw Table Skeleton

Intuitively, a table is an *organized collection of sets of cells, each cell contains an appropriate expression* (i.e. Figures 1 - 5, compare [23]). 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 $I = \prod_{i=1}^{n}$ (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 7 illustrates the above definitions.

5 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 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* [6, 16]. A component $A \in Components(T)$ is *maximal* if $A \longmapsto^* B$ implies $B = A$ for every

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

$B \in Components(T)$. 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 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 5.1

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 $Guards(T)$.

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

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 8 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 [23],
- type 2a corresponds to Inverted, Decision and Generalized Decision Tables [11, 23],
- type 2b corresponds to Vector Tables of [23].

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.

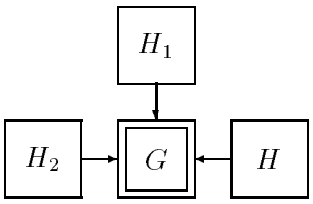
By adding the Cell Connection Graph we obtain the next approximation of the table skeleton concept.

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

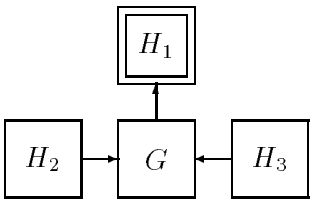
$$T = (CCG, H_1, ..., H_n, G)$$

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

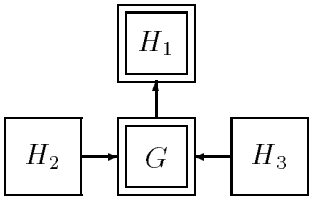
The type of Cell Connection Graph will usually be identified by a small icon resembling an appropriate graph from Figure 8. The icon is placed in left upper corner of the table. Figure 9 presents examples of medium table skeletons.



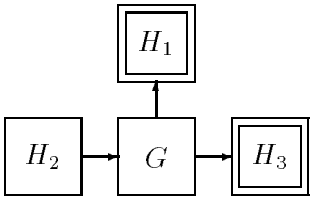
Type 1. Each element is either maximal or minimal. There is only one maximal element.



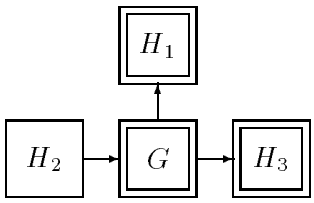
Type 2a. There is only one maximal element and a neutral element. The neutral element belongs to $Guards(T)$.



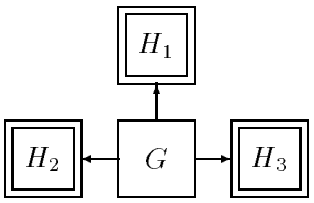
Type 2b. There is only one maximal element and a neutral element. The neutral element belongs to $Values(T)$.



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

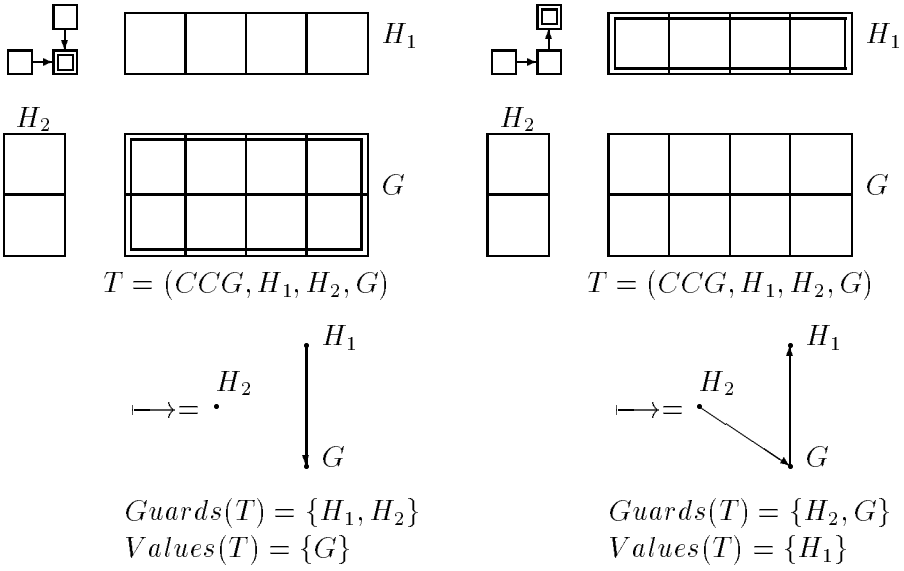


Figure 9: Medium table skeletons.

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

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

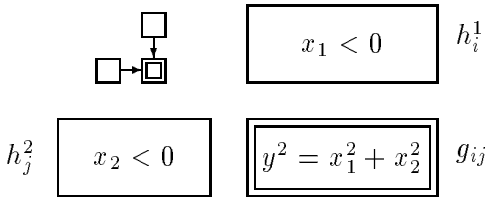


Figure 10: An example of (partially) interpreted medium element.

will be called the *cell connection graph of* $\pi_\alpha(T^{raw})$.

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

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

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 10 and Figure 6 illustrate medium elements.

7 Well Done Table Skeleton

Let $R \subseteq \mathbf{IN} \times \mathbf{OUT}$ be a *relation* and let $\mathcal{F} = \{R_\alpha \mid \alpha \in I\}$ be its *representation*. 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 (vector) variable over $\mathbf{IN}$, $\mathbf{y}$ be a (vector) variable over $\mathbf{OUT}$, and let $P(\mathbf{x})$ be a predicate defining the domain of R_α, i.e.

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

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 satisfies

$$R_\alpha = dom(R_\alpha) \hookrightarrow E_\alpha,$$

and is entirely described by the following predicate expression[3]

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

We have to now fit the *predicate* expression **if** $P_\alpha(\mathbf{x})$ **then** $E_\alpha(\mathbf{x},\mathbf{y})$ (or *relational* expression $dom(R_\alpha) \hookrightarrow E_\alpha$) into the *medium element* $\Gamma_\alpha = \pi_\alpha(T)$. Figure 10 shows how it can be done for the expression **if** $x_1 < 0 \wedge x_2 < 0$ **then** $y^2 = x_1^2 + x_2^2$, Figure 6 shows how it is done for the six other expressions.

The idea we will be using is the following:

- the relation $\longrightarrow^*$ defined on cells corresponds to $\hookrightarrow$.
- 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 10. 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.?

The cell connection graphs are *identical* for the tables from Figures 2, 3, 4 and 5, so are the medium elements containing the functions $g_{1,2}$, $\varphi_{2,2}$, $h_{1,2}$ and the relations $G_{1,1}$, $G_{1,3}$ of Figure 6. For the function $g_{1,2}$ we have $P_{1,2}(x,y) = y \geq 0 \wedge 0 \leq x < y$, with h_1^2 containing $y \geq 0$ and $g_{1,2}$ containing $0 \leq x < y$. For the function $\varphi_{2,2}$ we have decided that $P_{2,2}(\text{Weather})$ is of the form $\text{Weather} = sunny$, where Weather is in the cell h_2^2 and $sunny$ in $g_{2,2}$, while for the function $h_{1,2}$ we have $P_{1,2}(x_1,x_2) = x_1 x_2 \geq 0$, even so the cell h_1^2 contains $x_1 x_2$ and the cell $g_{1,2}$ contains $\# \geq 0$.

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 $E_\alpha(\mathbf{x},\mathbf{y})$. The relations $G_{1,1}$ and $G_{2,3}$ from Figure 6 illustrate the problem.

[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 $f_{3,2}$ in Figure 6). But clearly **if** $P_\alpha(\mathbf{x})$ **then** $E_\alpha(\mathbf{x},\mathbf{y}) = P_\alpha(\mathbf{x}) \wedge E_\alpha(\mathbf{x},\mathbf{y})$.

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 details will be discussed in the next section.

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 $R \subseteq \mathbf{IN} \times \mathbf{OUT}$ should be built from all R_α's that create a *representation* $\mathcal{F}$ of R. As Figures 1 - 5 indicate, there is nothing in the middle table skeleton to say how all those R_α's should be composed.

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

The final approximation of a table skeleton is the following.

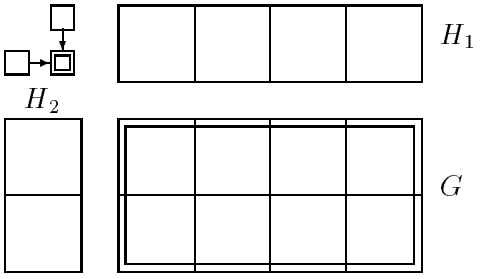
- A *well done table skeleton* is a tuple

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

  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.

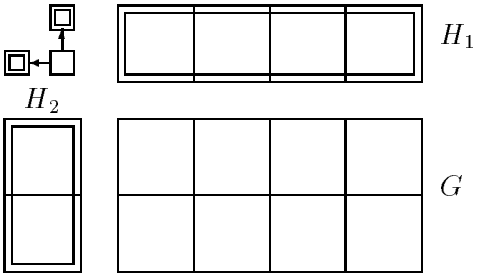
The definition is illustrated in Figure 11.

Normal



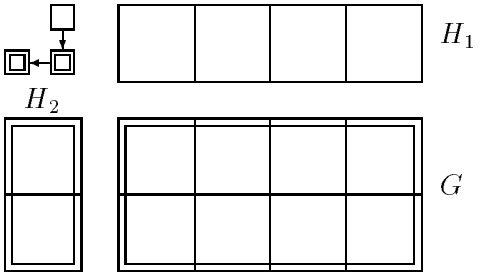
$P_T(H_1, H_2) = H_1 \wedge H_2$
$r_T(G) = G$
Relation name : R
$C_T : (R = \bigcup_{i=1}^{4} \bigcup_{j=1}^{2} R_{i,j})$

Type 4



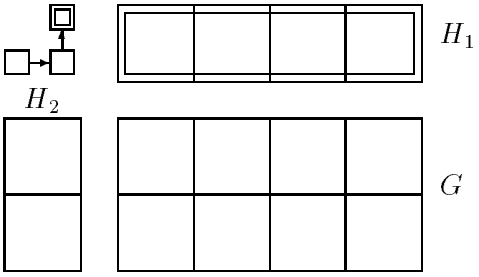
$P_T(G) = G$
$r_T(H_1, H_2) = H_1 \cup H_2$
Relation name : R
$C_T : (R = \bigoplus_{i=1}^{4} \bigotimes_{j=1}^{2} R_{i,j})$

Vector



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

Generalized Decision



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

Figure 11: Four examples of well done table skeletons.

8 Tabular Expressions

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

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

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

  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 $\mathbf{IN} \times \mathbf{OUT}$.

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.

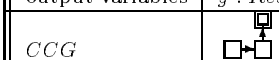
Examples of tables are presented in Figures 12 and 13. The signatures enriched by information about variables are presented separately in special two column tables. The above definitions describes, more or less, the *syntax* of tables. However the word 'syntax' here has the meaning closer to that used 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. However we can say a little about the syntax of P_T, r_T and C_T.

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, 29]), 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 (sub-) domains $\mathcal{C}(\mathbf{IN})$. The relation symbol "=" can always be used. If the

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_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

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	$\bigcup_{i=1}^{3} \bigcup_{j=1}^{2} g_{i,j}$

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

G

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

H_1

G

Figure 12: Three examples of tabular expressions. They correspond to those of Figures 1, 2 and 3.

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
$*$	$*$	hot	$*$	$cool$
$sunny \vee cloudy$	$sunny$	$cloudy$	$rain$	$cloudy$
$true$	$false$	$false$	$*$	$false$

H_2

Temperature $\in \{hot, cool\}$
Weather $\in \{sunny, cloudy, rain\}$
Windy $\in \{true, false\}$

G

---

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

$\# < 20$	$\# \geq 20$	$true$
$\# < 2$	$\# > 2$	$\# = 2$

G

---

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

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

$y_1 = 0$	$y_1 = 1$

H_1

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

G

Figure 13: Another three examples of tabular expressions. The two above correspond to those of Figures 4 and 5.

elements of $\mathcal{C}(\mathbf{IN})$ 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 "$\cup$", "$\cap$", "$\oplus$", "$\otimes$", etc., relation operators "=", "<", ">", etc., the operator of "concatenation" "$\circ$"[5].

Let I be the index of T, let

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

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

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 composition expression C_T is built from the relation/function names, indexes, and operators $\oplus$, $\otimes$, $\ominus$ ($\cup$, $\cap$, $\setminus$ are special cases, see Section 3). 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.

9 Semantics of Tabular Expressions

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

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

Figure 10 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 $A_\alpha \sqsubseteq \mathbf{IN}$, $E_\alpha \sqsubseteq \mathbf{OUT}$, as

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

[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 [13] of the approach presented here.

[5]For example for Figure 12 we have $((y_1 =) \circ (x_1 + x_2)) = (y_1 = x_1 + x_2)$, $((y_3|) \circ (y_3 + x_1 x_2 = |y_3|^3) = (y_3 \mid y_3 + x_1 x_2 = |y_3|^3)$, where $y_3 \mid y_3 + x_1 x_2 = |y_3|^3$ means that y_3 is the (only) output variable in the expression $y_3 + x_1 x_2 = |y_3|^3$.

Every interpreted medium element $\pi_\alpha(T)$ describes now the relation $R_\alpha = A_\alpha \hookrightarrow E_\alpha$, 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* $\pi_\alpha(T)$.
- The *semantics* of a *tabular expression* T is defined by:

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

Figures 6, 12 and 13 illustrate the above definitions.

10 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 8. The type 1 is standardly called *normal*, and type 2a is called *inverted*.

The table is called *plain* if C_T defines the plain representation of R_T (see Chapter 3.3), i.e. if $R_T = \bigcup_{\alpha \in I} R_\alpha$. It is *locally represented* if C_T defines the local representation, i.e. if $R_\alpha \sqsubseteq R_T$ for every $\alpha \in I$. The table is called *output-vector* if C_T defines the output-vector representation of R_T, i.e. if $R_T = \bigotimes_j \bigoplus_i R_{i,j}$. The table is called *input-vector* if C_T defines the input-vector representation of R_T (see , i.e. if $R_T = \bigoplus_i \bigotimes_j R_{i,j}$. All tables modeled in [13] are plain. The vector tables of [23] are of output-vector type, the most of (but not all) decision tables [11, 12, 23] 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, 23], 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 [23, 29, 30], 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.

11 Final Comment

In the paper a formal semantics for tabular expressions is proposed. The tables introduced here are generalizations of those from [23, 13] and [1]. As opposed to [23], one model covers all cases. An introduction of CCG, C_T, P_T and r_T gives us a tool to define various types of tables, some of them could really be useful. The cell connection graph CCG and the composition rule C_T 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 CCG, and three major types have been distinguished according to C_T. This paper is an extension and continuation of [13], where 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. 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 [23]. 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 [23]. The classification provided in [23] 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 12 and bottom table in Figure 13). 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 [14] might be useful here.

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.

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

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

[7] 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.

[8] 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.

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

[10] 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.

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

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

[13] 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.

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

[15] 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.

[16] K. Kuratowski, A. Mostowski, *Set Theory*, North Holland 1976.

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

[18] 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.

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

[20] 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.

[21] 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.

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

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

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

[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] 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.

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

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