Tabular Expressions and Their Relational Semantics

Ryszard Janicki[*] **and Alan Wassyng**[*]

Department of Computing and Software

McMaster University

Hamilton, Ontario, Canada L8S 4K1

janicki,wassyng@mcmaster.ca

Abstract. *Tabular Expressions* (Parnas et al. [20, 28, 32, 33]) 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 most known types of tables used in software engineering, and admits precise classification and definition of new types of tables. The practical importance of the semantics of tabular expressions is also discussed.

1. Introduction

In the classical engineering fields, as well as in mathematics, formulae are seldom longer than a dozen or so lines. In software engineering, the formulae 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 formulae, but not for long ones. One way to deal with long formulae is to use some form of module structure and hierarchical structuring (see [24]). However hierarchical structuring and modularity alone *are not sufficient* (see [32, 33]). The problem is that standard mathematical notation is, in principle, *linear*. This makes it hard to read when many cases have to be considered, when functions have many irregular discontinuities, or when the domain and range of functions are built from elements of different types. The *multi-dimensional tabular notation* makes it easier to consider every case separately while writing or reading a requirements or design document. It

turns out that using tables helps to make mathematics more practical for computing systems applications [20].

The key assumptions behind the idea of tabular expressions are:

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

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

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

In principle, tabular expressions are a generalization of two dimensional tables which are well known and have been around now for many years. Decision tables and state transition tables [14] date back to early years in computer science. In the late 1970s David Lorge Parnas and others at the U.S. Naval Research Laboratories used tabular representations to document requirements for the A-7E aircraft [4, 11, 12, 28, 35]. The ideas were quicly picked up by Grumman, the U.S. Air Force, Bell Laboratories and many others [33, 36, 41]. Since then, a number of projects have used tables to document requirements and software design. The best known example is probably the software for the Darlington Nuclear Generating Station Shutdown System (Ontario, Canada) [3, 30, 31, 41]. At least two organizations, Ontario Power Generation ([27, 39, 41]), and U.S. Naval Research Laboratories ([10]) have fashioned their approach to software development around the use of tabular expressions. Other users include the Software Quality Research Laboratory at McMaster University, [22, 23, 37, 38, 43], ORA Inc. [13], and University of California at Irvine [9, 26].

More than any other person, David Lorge Parnas has championed the use of tabular expressions in documenting software [20, 30, 31, 32, 33]. He also suggested the first semantic analysis of tabular expressions [29].

The semantics discussed in this paper were first proposed in [15] and subsequently developed in [16, 19, 21].

The model presented here covers most of (but *not all*) the known types of tables used in Software Engineering (see [1, 40]).

The central concept in our approach is the so-called *cell connection graph* which characterizes the *information flow* of a given table.

All examples of tables used in this paper are very simple in purpose. More realistic examples (such as loop invariants, program specifications) the reader can find in [1, 33, 38] and others.

The next section contains some introductory examples and informally explains our approach. Section 3, the main section of this paper, describes a formal semantics of tabular expressions. A first rough approximation of the table concept is given in Section 3.1. The crucial concept of the *Cell Connection Graph* and more precise approximations of the table concept are discussed in Sections 3.2, 3.3, 3.4 and 3.5. The formal definition of a *tabular expression* on a syntactic level is given in Section 3.6, and its *semantics* is discussed in Section 3.7. Section 4 contains some ideas on table classification, Section 5 describes why the development of semantics of tabular expressions is important, and final comments are in Section 6.

This paper is an extended and refined version of the results presented as two conference papers [21] and [40], and a continuation of the results presented in [16, 19]. Reference [29] provided a major motivation for this work.

We assume that the reader is familiar with such concepts as function, relation, Cartesian product, etc. Standard mathematical notation is used throughout the paper.

2. Examples and Motivation

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 a classical mathematical notation which sometimes allows us to relax the linearity principle. Lamport [24] proposes similar relaxation rules for more complex cases. In a purely linear notation, the definition of the function f is represented by

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

which is less readable than the classical mathematical notation. However, arguably the *most* readable definition is that represented in Figure 1, where the concept of a *table* is used. Note that in all our examples, value/result cells have *double border lines*.

Consider now the function g defined as

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

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* $G \subseteq \textbf{IN} \times \textbf{OUT}$, where $\textbf{IN} = Reals \times Reals$, $\textbf{OUT} = Reals \times Reals \times Reals$, x_1, x_2 are the variables over $\textbf{IN}$, y_1, y_2, y_3 are variables over $\textbf{OUT}$,

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

$x \geq 0$
$x < 0$

0	y^2	$-y^2$
x	$x + y$	$x - y$

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

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

$y \geq 0$
$y < 0$

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

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

and

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

while the table in Figure 4 defines the function $\varphi : \text{Temperature} \times \text{Weather} \times \text{Windy} \to Activities$, 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 specification tools since the fifties [13, 14].

Figure 5 contains a *generalized decision table* [1, 29]. It represents the function $h : Reals \times Reals \to Reals$ defined as

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

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 simpler local functions.

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

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

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

$x_2 \leq 0$	$x_2 > 0$

$y_1 =$	$x_1 + x_2$	$x_1 - x_2$
$y_2 \vert$	$y_2 x_1 - x_2 = y_2^2$	$x_1 + x_2 y_2 = \vert y_2 \vert$
$y_3 \vert$	$y_3 + x_1 x_2 = \vert y_3 \vert^3$	$y_3 = x_1$

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

go sailing	gtb	gtb	play bridge	garden

Temperature $\in \{hot, cool\}$	$*$	$*$	hot	$*$	$cool$
Weather $\in \{sunny, cloudy, rainy\}$	$sunny \vee cloudy$	$sunny$	$cloudy$	$rainy$	$cloudy$
Windy $\in \{true, false\}$	$true$	$false$	$false$	$*$	$false$

$* = don't\ care$, gtb = go to the beach

Figure 4.   The function φ defined by a (decision) table (from [13]).

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

$x_1 x_2$	$\# < 20$	$\# \geq 20$	$true$
$x_1 \div x_2$	$\# > 30$	$\# < 30$	$\# = 30$

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

The relation G is a composition of local 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 $IN_{1,3} = Reals \times (-\infty, 0)$, $OUT_{1,3} = Reals$, and

$$(x_1, x_2)G_{1,3}y_3 \iff y_3 + x_1x_2 = |y_3|^3.$$

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 their respective output variables.

The function φ is a composition of $\varphi_{i,j}$, $i = 1, ..., 5$, $j = 1, 2, 3$, and for instance $\varphi_{2,2} : \{sunny\} \to \{go\ to\ the\ beach\}$, $\varphi_{2,2}(sunny) = go\ to\ the\ beach$. The domain of $\varphi_{2,2}$ is $\{sunny\}$ rather than $\{sunny, cloudy, rainy\}$ 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 $h_{2,1} : IN_{2,1} \to Reals$, where $IN_{2,1} = \{(x_1, x_2) \mid x_1x_2 > 20\}$ and $h_{2,1}(x_1, x_2) = x_1 - x_2$, while $h_{3,1} : Reals \times Reals \to Reals$ and $h_{3,1}(x_1, x_2) = x_1x_2$.

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

$$h = \bigcup_{i=1}^{3} \bigcap_{j=1}^{2} h_{i,j}, \text{ and we will show that } G = \bigotimes_{j=1}^{3} \bigcup_{i=1}^{2} G_{i,j}, \quad \varphi = \bigcup_{i=1}^{5} \bigotimes_{j=1}^{3} \varphi_{i,j}, \text{ and } h = \bigcup_{i=1}^{3} \bigotimes_{j=1}^{2} h_{i,j},$$

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 detail in Section 3.5.

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 containing an *expression*. The *local 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.

Every *local 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 from 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 $y_3 + x_1x_2 = |y_3|^3$, where $\mathbf{x} = (x_1, x_2)$, for $h_{2,1}$ we have $P_{2,1}(x) = x_1x_2 \geq 20$, $R_{2,1}(\mathbf{x}, y)$, 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.

All the above observations will be used to build a homogeneous semantics of most possible types of tables.

$$\left. \begin{array}{l} \boxed{y < 0} \\[4pt] \boxed{x < 0} \quad \boxed{\boxed{x - y}} \end{array} \right\} \qquad f_{3,2}(x, y) = \textbf{if } y < 10 \wedge x < 0 \textbf{ then } x - y$$

$$\left. \begin{array}{l} \boxed{\boxed{x - y}} \\[4pt] \boxed{y \geq 0} \quad \boxed{0 \leq x < y} \end{array} \right\} \qquad g_{2,1}(x, y) = \textbf{if } y \geq 0 \wedge 0 \leq x < y \textbf{ then } x - y$$

$$\left. \begin{array}{l} \boxed{x_2 \leq 0} \\[4pt] \boxed{\boxed{y_1 =}} \quad \boxed{\boxed{x_1 + x_2}} \end{array} \right\} \qquad (x_1, x_2) G_{1,1} y_1 \iff \textbf{if } x_2 \leq 0 \textbf{ then } y_1 = x_1 + x_2$$

$$\left. \begin{array}{l} \boxed{x_2 \leq 0} \\[4pt] \boxed{\boxed{y_3}} \quad \boxed{\boxed{y_3 + x_1 x_2 = |y_3|^3}} \end{array} \right\} \qquad (x_1, x_2) G_{1,3} y_3 \iff \textbf{if } x_2 \leq 0 \textbf{ then } y_3 + x_1 x_2 = |y_3|^3$$

$$\left. \begin{array}{l} \boxed{\boxed{\text{go to the beach}}} \\[4pt] \boxed{\text{Weather} \in \{sunny, cloudy, rainy\}} \quad \boxed{sunny} \end{array} \right\} \qquad \begin{array}{l} \varphi_{2,2}(\text{Weather}) = \quad \textbf{if } \text{Weather} = sunny \\ \qquad\qquad\qquad\qquad\quad \textbf{then } \text{go to the beach} \end{array}$$

$$\left. \begin{array}{l} \boxed{\boxed{x_1 - x_2}} \\[4pt] \boxed{x_1 x_2} \quad \boxed{\# \geq 20} \end{array} \right\} \qquad h_{1,2}(x_1, x_2) = \textbf{if } x_1 x_2 \geq 20 \textbf{ then } x_1 - x_2$$
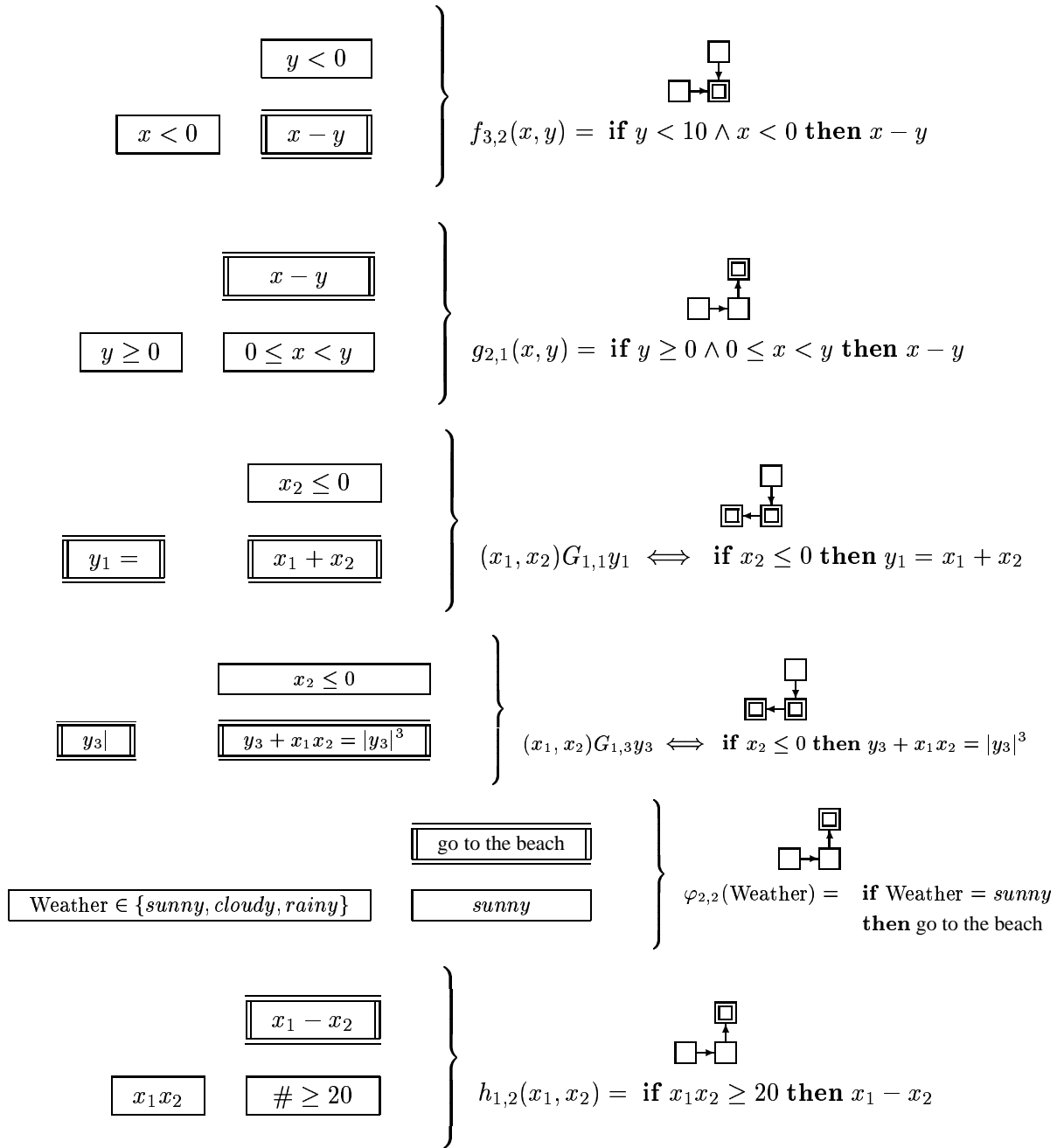
Figure 6.　The functions $f_{3,2}$, $g_{2,1}$, $\varphi_{2,2}$, $h_{1,2}$, the relations $G_{1,1}$, $G_{1,3}$ and their appropriate raw elements, **if-then** descriptions, and cell connection graphs.

3. Tabular Expressions and Their Semantics

The aim of this section is to provide a formal definition of Tabular Expressions and their semantics. This is an improved and revised version of the model that was presented for the first time in [15] and was subsequnetly developed in [16, 19].

We define a *Tabular Expression* (or *Table*) as a tuple:

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

where:

- P_T is a table predicate rule which indicates how predicates that define local representations are to be built from the contents of table cells,

- r_T is a table relation rule which indicates how relations/functions that are local representations are to be built from the contents of table cells,

- C_T is a table composition rules which states how the global relation/function is built from local representations,

- CCG is a cell connection graph which defines the information flow in the table,

- $H_1, ..., H_n$ are headers of the table,

- G is a grid of the table,

- Ψ is a mapping that assigns particular expressions to table cells,

- $\mathbf{IN}$ is the set of inputs, and
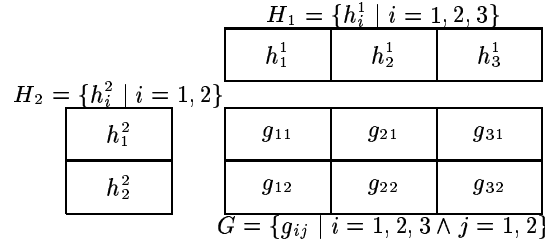
- $\mathbf{OUT}$ is the set of outputs.

The relation R_T which describes the semantics of T, has the property: $R_T \subseteq \mathbf{IN} \times \mathbf{OUT}$.

The subsections below will define and analyse all the notions introduced above.

3.1. 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 $I = \{1, 2, ..., k\}$, for some k, is a set of indices.

- 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*.

$H_1 = \{h_i^1 \mid i = 1, 2, 3\}$		
h_1^1	h_2^1	h_3^1

$H_2 = \{h_i^2 \mid i = 1, 2\}$

h_1^2
h_2^2

g_{11}	g_{21}	g_{31}
g_{12}	g_{22}	g_{32}

$G = \{g_{ij} \mid i = 1, 2, 3 \wedge j = 1, 2\}$

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

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

- A *raw table skeleton* is a tuple
$$T^{raw} = (H_1, ..., H_n, G)$$
  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
  $Comp(T) = \{H_1, ..., H_n, G\}$ are called *table components*.

Figure 7 illustrates the above definitions.

3.2.   Cell Connection Graph and Medium Table Skeleton

The first step in expressing the semantic difference between various types of tables is to define a *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 additional requirement for the relation is that each arc *must either start from or end at the grid G*.

Let $Comp(T) = \{H_1, ..., H_n, G\}$. A *Cell Connection Graph* is an *asymmetric* relation

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

satisfying: for all $A, B \in Comp(T)$,

$$A \longmapsto B \implies ((A = G \vee B = G) \wedge A \neq B), \tag{1}$$

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

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

---

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

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, then $A \longmapsto^+ B$, 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 $Comp(T)$ into $Guards(T)$ and $Values(T)$ must satisfy the following properties:

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

One can also easily show that *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 different types of Cell Connection Graphs when not distinguishing among the headers.

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.

---

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

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 illustrates 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 in [29],

- type 2a corresponds to Inverted, Decision and Generalized Decision Tables [13, 29],

- type 2b corresponds to Vector Tables in [29].

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. Reference [1] provides an excellent survey of most types 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

$$T^{med} = (CCG, H_1, ..., H_n, G)$$

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 the left upper corner of the table. Figure 9 presents examples of medium table skeletons. Note how the CCG shows where to start reading the table (H_1, H_2 in the normal table, H_2, G in the inverted table), and where to find the results (G in the normal, H_1 in the inverted table).

3.3. Raw and Medium Table Elements

Let $T^{med} = (CCG, H_1, ..., H_n, G)$ be a medium table skeleton with 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 $(h^1_{i_1}, ..., h^n_{i_n}, g_\alpha)$ is a *raw element* if and only if $\alpha = (i_1, ..., i_n)$.
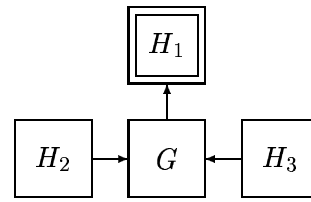
We will denote the raw element $(h^1_{i_1}, ..., h^n_{i_n}, g_\alpha)$ by $T^{raw}|_\alpha$, since it can be interpreted as a kind of *projection* (*restriction*) of T^{raw} onto the index α. The *set* $\{h^1_{i_1}, ..., h^n_{i_n}, g_\alpha\}$ will be denoted by $Comp_\alpha(T^{raw})$.

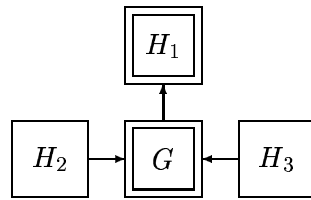Let $\longmapsto_\alpha \subseteq Comp_\alpha(T^{raw}) \times Comp_\alpha(T^{raw})$ be a relation defined as

$$c_1 \longmapsto_\alpha c_2 \iff \exists A_1, A_2 \in Comp(T^{raw}).\ c_1 \in A_1$$
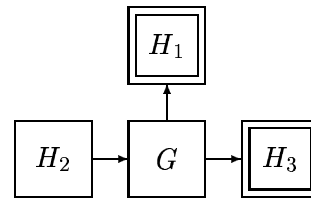$$\wedge\ c_2 \in A_2\ \wedge\ A_1 \longmapsto A_2.$$
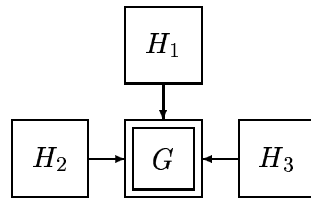
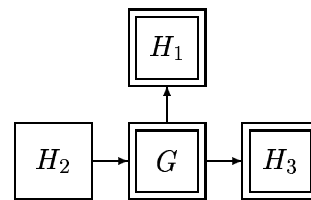*Type 1. Each element is either maximal or minimal. There is only one maximal element.*

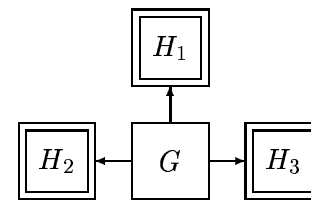*Type 2a. There is only one maximal element and a neutral element. The neutral element belongs to $Guards(T)$.*

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 ($n = 3$). Double boxes indicate value cells, single boxes indicate guard cells
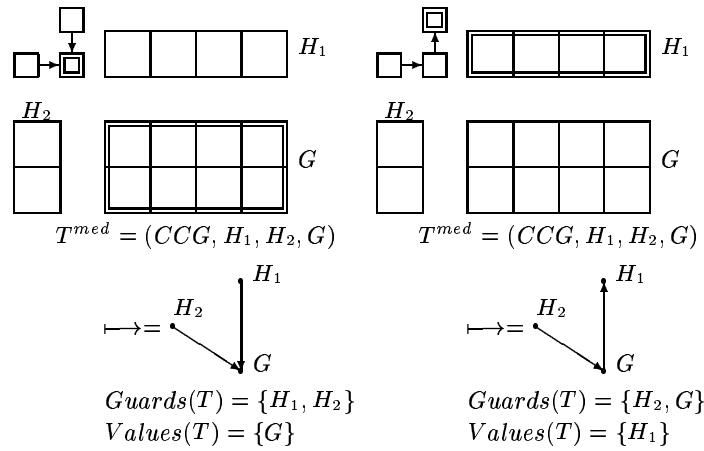
Figure 9.   Medium table skeletons.

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

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

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

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

where $(h_{i_1}^1, ..., h_{i_n}^n, g_{\alpha})$ is a raw element. Again, a medium element can be interpreted as a projection of T^{med} onto α. Figure 10 illustrates a medium element.

3.4.   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 $dom(R) \subseteq X_1 \times ... \times X_k$, for some k, $range(R) \subseteq Y_1 \times ... \times Y_m$, for some m.

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 basic idea behind using tables to specify the relation R is that in practice we can frequently use a *medium element* $\Gamma_\alpha = T^{med}|_\alpha$ 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 $R = Expr(\mathcal{F})$, where $Expr(\mathcal{F})$ is a relational expression built from the elements of $\mathcal{F}$ and relational operators that are defined by the table type. The table structure is supposed to make the understanding of $Expr(\mathcal{F})$ natural and simple.

Let $\mathbf{x}$ be a (possible scalar or 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 described completely within that domain 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** $E_\alpha(\mathbf{x}, \mathbf{y})$ into the *medium element* $\Gamma_\alpha = T^{med}|_\alpha$. Figure 10 shows how it can be done for the example of **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 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.?

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.

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

---

[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 prefer the **if-then** form because it is more readable, in particular when $P_\alpha(\mathbf{x})$ itself contains the "$\wedge$" operator (see Figure 12). But clearly **if** $P_\alpha(\mathbf{x})$ **then** $E_\alpha(\mathbf{x}, \mathbf{y}) = P_\alpha(\mathbf{x}) \wedge E_\alpha(\mathbf{x}, \mathbf{y})$.
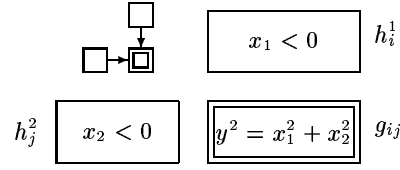
Figure 10. An example of a (partially) interpreted medium element.

To say precisely how the *medium element* can be used to specify the predicate expression **if** $P_\alpha(\mathbf{x})$ **then** $E_\alpha(\mathbf{x}, \mathbf{y})$, not only do we need to divide cells into value and guard types, but we also need to describe 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 $T = (CCG, H_1, \ldots, H_n, G)$ be a medium table skeleton. Assume that $Guards(T) = \{B_1, \ldots, B_r\}$, $Values(T) = \{A_1, \ldots, A_s\}$.

- A predicate expression $P_T(\mathbf{B_1}, ..., \mathbf{B_r})$, where $\mathbf{B_1}, ..., \mathbf{B_r}$ are variables corresponding to the components $B_1, ..., B_r$, is called a *table predicate rule*.

- A relation expression $r_T(\mathbf{A_1}, ..., \mathbf{A_s})$, where $\mathbf{A_1}, ..., \mathbf{A_s}$ are variables corresponding to the components $A_1, ..., A_s$, is called a *table relation rule*.

The predicate $P_\alpha(\mathbf{x})$ can now be derived from $P_T(\mathbf{B_1}, ..., \mathbf{B_s})$ by replacing each variable $\mathbf{B_i}$ 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 be derived from $r_T(\mathbf{A_1}, ..., \mathbf{A_s})$ by replacing each variable $\mathbf{A_i}$ 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) $\mathbf{B_1}, ..., \mathbf{B_r}$, where $Guards(T) = \{B_1, ..., B_r\}$, together with logical operators "$\wedge$", "$\vee$", "$\neg$", the replacement operator, and 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 $\mathbf{A_1}, ..., \mathbf{A_r}$ (variables), where $Values(T) = \{A_1, ..., A_r\}$, together with set operators "$\cup$", "$\cap$", etc., relation operators "$=$", "$<$", "$>$", etc., and the operator of "concatenation" "$\circ$"[5].

The table predicate and relation rules are sufficient to understand how the predicate 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

---

[4]The survey [1] indicates that "$\wedge$", "$\vee$", "$=$" and "$E[E_1/x]$" suffice in most cases.

[5]For example for Figure 13 (top half) 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 $y_3 + x_1 x_2 = |y_3|^3$.

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 us 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 built 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

$$T^{well} = (P_T, r_T, C_T, CCG, H_1, ..., H_n, G),$$

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.

In essence, a well done table skeleton defines the complete 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 11.

3.5.   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 defined in the standard way (see [34]).

In many cases R_α's are heterogeneous relations [34] 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 operations $\oplus, \otimes, \ominus$ as follows

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

Normal

Type 4

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

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

Vector

Generalized Decision

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

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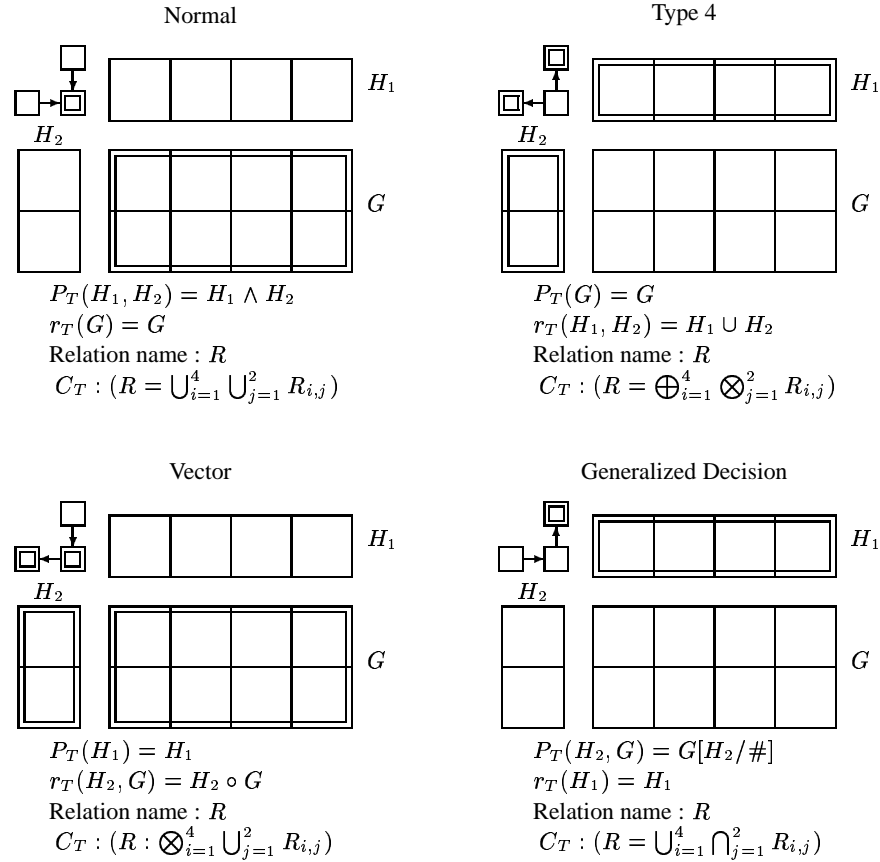$C_T : (R = \bigcup_{i=1}^{4} \bigcap_{j=1}^{2} R_{i,j})$

Figure 11.   Four examples of well done table skeletons.

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

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

then we have

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

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 (see [19] for more details). 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. This observation provided a major motivation for an attempt to formulate a mereology for direct product and relations [17, 18].

Why is the composition of R from R_α important? There are a number of practical considerations that are affected by this.

- One of the major advantages of tabular expressions in documenting functions and relations is that the table can be checked for "correctness". For example, in Figure 1, the table can be checked to ensure that the entire input domain is covered, and that the table components are disjoint. Since $P_T(H_1, H_2) = H_1 \wedge H_2$, and $C_T : (f = \bigcup_{i=1}^{3} \bigcup_{j=1}^{2} f_{i,j})$, we can see that to satisfy input domain coverage we must have

$$\bigcup_{i=1}^{3} dom(H_i^1) = Reals, \text{ and } \bigcup_{i=1}^{2} dom(H_i^2) = Reals$$

  and for disjointedness we must have $dom(H_1^1) \cap dom(H_2^1) = \emptyset$, $dom(H_1^1) \cap dom(H_3^1) = \emptyset$, $dom(H_2^1) \cap dom(H_3^1) = \emptyset$, and $dom(H_1^2) \cap dom(H_2^2) = \emptyset$.

  The practical importance of this is clear. Tabular expressions are typically used to specify behaviour, and so complete input domain coverage ensures that we have specified responses to every input combination, while disjointness ensures that the responses are unambiguously defined.

- We do not know of any practical systems that can be documented by a single (readable) table. We thus need to be able to document systems by a collection of tables. In some cases we may want a cell in the table to refer to another table. If we do not have a knowledge of the composability of the individual tables we cannot hope to produce a mathematically consistent description.

- In a similar vein, it is sometimes important to be able to link different kinds of tables to describe the total behaviour of a system. These links usually take the form of a cell in one table referring to a cell in a different kind of table. We have also seen a cell in one table reference a row (or column) in another table.

3.6. Tabular Expressions

We are now able to fully define the concept of a table expression that was introduced at the beginning of Section 3.

- 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 $\mathbf{IN}$, the relation expressions have variables over $\mathbf{IN} \times \mathbf{OUT}$, where $\mathbf{IN}$ is the set (usually heterogeneous product) of inputs, and $\mathbf{OUT}$ is the set (usually heterogeneous 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).

Figures 12, 13 and 14 show examples of tabular expressions. The signatures enriched by information about variables are presented in special two column tables. The above definitions describe, more or less, the *syntax* of tables. In general Ψ may assign a *predicate expression*, or part of one, to each guard cell, and a *relation expression*, or part of one, to each value cell. We do not provide a complete denotational semantics of Ψ over the syntactic expressions in cells. It can be done, but in most cases it is rather obvious, and quite lengthy in the general case. Formally we need to introduce first an operator $val(c)$ which simply returns the contents of cell c as an uninterpreted string of symbols (i.e. syntactic units), and then define Ψ in a standard denotational manner.

Let I be the index of T, let

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

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

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, and indices. The operators $\oplus, \otimes, \ominus, \cup, \cap, \setminus$ are special cases. The survey [1] shows that the patterns $\otimes_j \cup_i R_{i,j}$, $\cup_\alpha R_\alpha$, and $\cup_i \otimes_j R_{i,j}$ are sufficient in most cases.

3.7. 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:

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

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_α, E_α, as

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

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

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

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

H_1

H_2

$y \geq 0$
$y < 0$

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

G

Figure 12.   Two examples of tabular expressions - normal (above) and inverted (below). They correspond to Figures 1 and 2.

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

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

$y_1 =$
$y_2|$
$y_3|$

$x_1 + x_2$	$x_1 - x_2$
$y_2 x_1 - x_2 = y_2^2$	$x_1 + x_2 y_2 = |y_2|$
$y_3 + x_1 x_2 = |y_3|^3$	$y_3 = x_1$

G

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 respective output variables.

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

input variables	Temperature: $\{hot, cold\}$
	Weather: $\{sunny, cloudy, rainy\}$
	Windy: $\{true, false\}$
output variables	$action$:$\{$go sailing, go to the beach, play bridge, garden$\}$
CCG	⬚→⬚
P_T	$H_2 = G$
r_T	H_1
Function name	φ and $action = \varphi(Temperature, Weather, Windy)$
C_T	$\bigcup_{i=1}^{5} \bigotimes_{j=1}^{3} \varphi_{i,j}$
notation	$* = don't\ care$, gtb = go to the beach

H_1

go sailing	gtb	gtb	play bridge	garden

H_2

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

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

G

Figure 13. Next two examples of tabular expressions - vector table (above) and decision table [14] (below). They correspond to Figures 3 and 4.

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

input variables	$x_1, x_2 : Reals$
output variables	$z : Reals$
CCG	
P_T	$G[H_2/\#]$
r_T	H_1
Function name	h and $z = h(x_1, x_2)$
C_T	$\bigcup_{i=1}^{3} \bigotimes_{j=1}^{2} h_{i,j}$

H_2

$x_1 + x_2$	$x_1 - x_2$	$x_1 x_2$	H_1

$x_1 x_2$
$x_1 \div x_2$

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

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

input variables	$x : Reals$
output variables	$y_1, y_2 : Reals$
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

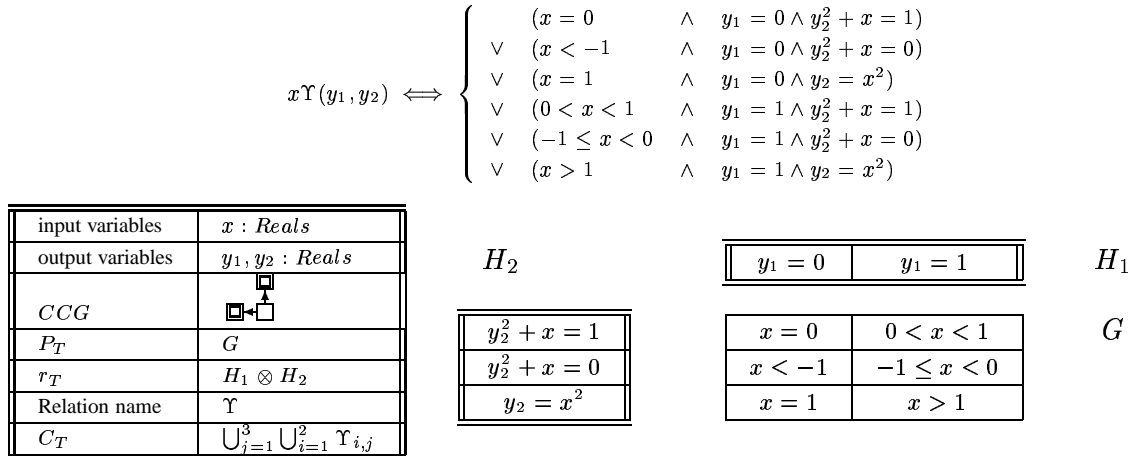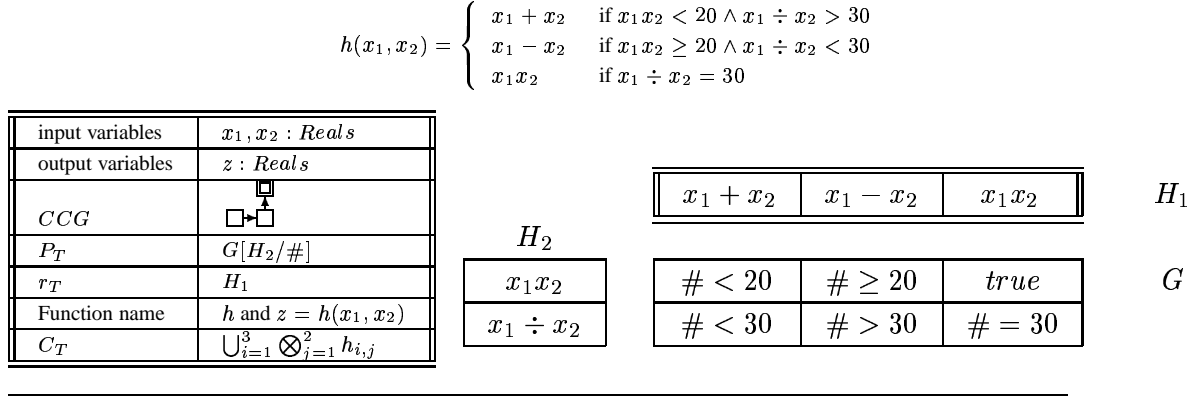| $x = 0$	$0 < x < 1$	G
$x < -1$	$-1 \leq x < 0$	
$x = 1$	$x > 1$	

Figure 14.   Another two examples of tabular expressions - generalized decision (above) and type 4 (below). The top one corresponds to Figure 5.

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

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

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 12, 13 and 14 illustrate the above definitions.

4. On Table Classification

Tabular expressions can be classified according to:

- the cell connection graph, CCG,
- the table composition rule, C_T,
- table predicate and relation rules, P_T and r_T,
- the mapping Ψ which assigns meaning to 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. Type 1 tables are called *normal*, and type 2a are called *inverted*. The relationship between normal and inverted tables is analyzed in detail in [44].

The table is called *plain* if $dom(R_\alpha) \subseteq dom(R)$, for all $\alpha \in I$, and $R_T = \bigcup_{\alpha \in I} R_\alpha$. The table is called *output-vector* if $R_T = \bigotimes_j \bigoplus_i R_{i,j}$. The table is called *input-vector* if $R_T = \bigoplus_i \bigotimes_j R_{i,j}$. In general, $\bigotimes_j \bigoplus_i R_{i,j} \neq \bigoplus_i \bigotimes_j R_{i,j}$, even though occassionally such equality might be true. All tables modeled in [15] are plain. The vector tables of [29] are of output-vector type, and most of (but not all) decision tables [13, 14, 29] 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, equality, and replacement $E[E'/\#]$, *disjunctive tables*, *conjunctive tables*, *equality tables*, and *#-replacement tables* are natural candidates for special table types.

Classification on the basis of Ψ is a different type of classification from each of the above. It depends on what the contents of cells are, 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 tables [29, 38, 44], is based mainly on Ψ, but also on r_T, P_T, and C_T. Some popular types such as vector, decision, and generalized decision require a special type of Ψ, a special type of C_T and a special type of P_T and/or r_T.

5. The Importance of Semantics of Tabular Expressions

There are some compelling reasons for developing semantics of tabular expressions.

- Software tools for creating, checking, transforming and presenting tabular expressions are a practical necessity. Building those tools without a semantic model of the tables is doomed to failure.

- One of the strongest motivations for using tabular expressions is that it enables us to use mathematical precision in the documentation of software requirements and design in a way that is quite intuitive to a broad base of readers/users. It is vital that everyone reading these documents has the same unambiguous understanding of the functionality described by these tables. Even in the early days of using tables, there was an attempt to tell readers how to interpret the tables, i.e. very simple/crude semantics for the specific tables were provided. Now that tabular expressions are becoming more popular, are used to document larger and more diverse systems, and are used within the context of various different mathematical models, it has become imperative that we find more general and sophisticated ways of telling readers how to interpret tabular expressions.

- It is natural that users of these tables will find ways of extending and/or modifying them so that they become more and more useful. In recent years we have made a concerted effort to make tables easy and intuitive to read and understand. In some cases we have modified the structure to take advantage of the visual advantages that tables provide. For example, we can take tables of the form:

$H_1 = Cases$

$Case_1$	$Case_2$

$H_2 = Conditions$ $G = Results$

Conditions	$Case_1$	$Case_2$
$Cond_1 \wedge Sub\_Cond_1$	$res_{1.1}$	$res_{1.1}$
$Cond_1 \wedge Sub\_Cond_2$	$res^1_{1.2}$	$res^2_{1.2}$
$Cond_2$	res_2	res_2
$Cond_3$	res^1_3	res^2_3
...	...	...
$Cond_n$	res^1_n	res^2_n

and introduce a visual enhancement to highlight the structure of predicates and results:

$H_1 = Cases$

$Case_1$	$Case_2$

$H_2 = Conditions$ $G = Results$

Conditions		$Case_1$	$Case_2$
$Cond_1$	Sub_Cond_1	$res_{1.1}$	
	Sub_Cond_2	$res^1_{1.2}$	$res^2_{1.2}$
$Cond_2$		res_2	
$Cond_3$		res^1_3	res^2_3
...		...	...
$Cond_n$		res^1_n	res^2_n

Without an understanding of the formal semantics of tables, it is likely that some of these modifications will be ill-formed, resulting eventually in ambiguous interpretations.

6.   Final Comments

The paper provides a relational semantics for Tabular Expressions. The Tabular Expressions have proven to be invaluable in documenting requirements and software designs, and very useful in testing and verification (see [40, 41]).

The model presented in the paper covers most but not all types of tables currently used in Software Engineering. It also allows us to define precisely new table types. 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 improper tables [29]. 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 address this problem. The model also does not address default cases such as "no change" semantics often used when defining state transition semantics. The "sparse matrix" representation of a function [1] is also not easily covered.

The approach presented here is complementary to that of [29]. The classification provided in [29] 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 unique 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, no substantive assumptions about the forms of P_T and r_T are 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, 33]), one may observe that in many cases the distribution of input and output variables among headers is not arbitrary, but are arranged to serve a particular purpose. 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, arrays of tabular expressions and the concept of time, are not addressed in this paper.

Semantics for tabular expressions are crucial to their use in practice and also to the development of software tools for creating, editing and transformimg tables. The model presented in the paper is not the only one that has been proposed. An algebra of arrays of relations has been used in [5, 6] to present an alternative semantics for tabular expressions. A strictly compositional semantics was proposed in [22], and an algebraic and recursive model was presented in [42].

A real strength of tabular expressions is their sequence independent view of behaviour. This enables us to deliver true black-box descriptions of behaviour. This strength is also a weakness. We have yet to find a way to use tabular expressions to document algorithms, simply because in an algorithm, the sequence of operations is supremely important [40].

Recently, tabular expressions have been succesfully used in requirements analysis, especially in conjunction with a scenario-based approach [7, 8, 23, 41], to specify and verify safety critical software [25, 41], and to deal with refinement problems [37].

There are some topics still that need further research in term of semantics for tabular expressions. For example, semantics that cope with arrays; semantics that deal with concurrency (see [43]) and for dealing with time are still in the early stages, if they exist at all.

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] J. Desharnais, R. Khédri, A. Mili, Interpretation of Tabular Expressions Using Arrays of Relations, In E. Orłowska, A. Szałas (eds.) *Studies in Fuzziness and Soft Computing*, Springer 2001, pp. 3-14.

[7] J. Desharnais, R. Khédri, A. Mili, Representation, Validation and Intergration of Scenarios Using Tabular Expressions, *Formal Methods in System Design*, to appear.

[8] J. Desharnais, R. Khédri, A. Mili, Computing the Demonic Meet of Relations Represented by Predicate Expressions Tables, Technical Report CAS-04-03-RK, McMaster University, Hamilton, Ontario, Canada, 2004.

[9] 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.

[10] 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.

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

[12] 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.

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

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

[15] 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.

[16] R. Janicki, On Formal Semantics of Tabular Expressions, CRL Report 355, McMaster University, Hamilton, Ontario 1997.

[17] R. Janicki, Remarks on Mereology of Direct Products and Relations, in J. Desharnais, M. Frappier, W. MacCaull (eds.), *Relational Methods in Computer Science*, Methodos Publ. 2002,pp.65-84.

[18] R. Janicki, On a Mereological System for Relational Software Specifications, Proc. of MFCS'2002 (Mathematical Foundations of Computer Science), *Lecture Notes in Comp. Science* 2420, Springer 2002, pp. 375-386.

[19] R. Janicki, R. Khédri, On Formal Semantics of Tabular Expressions, *Science of Computer Programming*, 39(2001), 189-214.

[20] 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.

[21] R. Janicki, A. Wassyng, On Tabular Expressions, *Proc. of. CASCON'2003* (13th IBM Centre for Advanced Studies Conference), Toronto, Canada 2003, pp. 38-52.

[22] W. Kahl, Compositional Syntax and Semantics of Tables, SQRL Report No. 15, McMaster University, Hamilton, Ontario, Canada, 2003, to appear in *Formal Methods in System Design*

[23] R. Khédri, Requirements Scenarios Formalization Technique: N Versions Towards One Good Version, in W. Kahl, D. L. Parnas, G. Schmidt (eds.), *Relational Methods in Software*, Elsevier 2001, 19-37.

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

[25] M. Lawford, J. McDougall, P. Froebel, G. Moum, Practical Application of Functional and Relational Methods for the Specification and Verification of Safety Critical Software, Proc. of AMAST'2000, *Lecture Notes in Computer Science* 1816, Springer 2000, pp. 73-88.

[26] 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.

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

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

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

[30] 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.

[31] 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.

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

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

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

[35] 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.

[36] A. J. van Schouwen, D. L. Parnas, J. Madey, Documentation of Requirements for Computer Systems, *International Symposium on Requirements Engineering*, IEEE Computer Society, San Diego, California, 1993, pp. 198-207.

[37] E. Sekerinski, Exploring Tabular Verification and Refinement, *Formal Aspect of Computing* 15 (2003), 215-236.

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

[39] A. Wassyng, GARD Research Consulting Inc., Software Requirements for AECB Project 2.314.1, 23141-DOC-4, Revision 2, 1995.

[40] A. Wassyng, R. Janicki, Tabular Expressions in Software Engineering, *Proc. of ICSSEA'03* (Intern. Conf. on Software and System Engineering), Vol. 4, Paris, France 2003, pp.1-46.

[41] A. Wassyng, M. Lawford, Lessons Learned from a Successful Implementation of Formal Methods in an Industrial Project, *Proc. of FME'03* (Formal Methods Europe), *Lecture Notes in Computer Science* 2805, Springer 2003, pp. 133-153.

[42] A. J. Wilder, Recursive Tables and Effective Definition Schemes, *The Journal of Logic and Algebraic Programming*, 51 (2002), 101-121

[43] Y. Yang, R. Janicki, On Concurrency and Tabular Expressions, Proc. of SERP'2004 (Software Engineering Research and Practice), Las Vegas, USA, 2004, pp. 455 - 461.

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