On a Formal Semantics of Tabular Expressions

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

Abstract

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

1 Introduction

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

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

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

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

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

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

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

The papers [38, 39] show in a formal way how the documentation required for the design and use of computing systems can consist of descriptions of a set of relations. Those relations are represented by multi-dimensional expressions called tables. Parnas [36] describes several different classes of tables and provides their formal syntax and semantics. All classes considered in [36] were invented for some specific practical applications. Formal relationship between some important classes of tables has been analyzed in [45]. The overall methodology and recent results of the tabular approach are presented in [24].
The tabular notation is currently used among others by the Software Engineering Research Group (SERG) at McMaster University, Hamilton, Ontario, Canada [43], Ontario Hydro [30], Naval Research Laboratory [14], ORA Inc., [19], and University of California at Irvine [13, 27].

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

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

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

The key assumptions behind the idea of tabular expressions are:

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

Figure 1: A part of the Software Requirements for the Water Level Monitoring System of the A-7 aircraft
• the relation $R$ may itself be complex but it can be built from a collection of relations $R_{\alpha}, \alpha \in I$, where $I$ is a set of indices, each $R_{\alpha}$ can be specified rather easily. In most cases $R_{\alpha}$ can be defined by a simple linear formula that can be held in few cells of a table. Some cells define the domain of $R_{\alpha}$, the others $R_{\alpha}$ itself.

• the tabular expression that describes $R$ is a structured collection of cells containing definitions of $R_{\alpha}$'s. The structure of a tabular expression informs how the relation $R$ can be composed of all the $R_{\alpha}$'s.

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

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

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

2 Tabular Expressions

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

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

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

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

The elements of the set $Components = \{H_1, ..., H_n, G\}$ are called table components.
Figure 2: An example of headers $H_1$, $H_2$, $I^1 = \{1, 2, 3\}$, $I^2 = \{1, 2\}$, and grid $G$.

Figure 2 illustrates the above definitions.

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

The assumption is that every $R_a$ is fully specified by some expressions held in one grid cell $g_a$, and header cells $h_{a|i}^j \in H_j$, where $a|i$ is the $i$th coordinate of $a$ (i.e. if $a = (j_1, \ldots, j_n)$, then $a|3 = j_3$ for $j = 1, \ldots, n$.

For every $a \in I$ we define

$$ Components_s_a = \{h_{a|1}^1, \ldots, h_{a|n}^n, g_a\}. $$

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

We assume that every relation $R_a$, $a \in I$ is specified by an expression of the form$^{1*}$.

$$\text{if } P_a \text{ then } E_a$$

where $P_a$ is the predicate that defines the domain of $R_a$ and $E_a$ is the predicate that defines the values of $R_a$. For example if $x_1 < 0 \land x_2 < 0$ then $y^2 = x_1^2 + x_2^2$, or if $-1 \leq x < 0$ then $y^2 + x = 0 \lor y = 1$ (see Figures 3).

---

$^{1*}$The predicate if $P_a$ then $E_a$ can equivalently be written as $P_a \land E_a$. We shall prefer if-then form because it is more readable, in particular when $P_a$ itself contains "$\land$" operator (see Figure 6). But clearly if $P_a$ then $E_a = P_a \land E_a$. Do not confuse “if $P \text{ then } E$” with “$P \Rightarrow E$”
\[ R_{ij} = \text{if } x_1 < 0 \land x_2 < 0 \text{ then } y^2 = x_1^2 + x_2^2 \]

\[ h^1_{ij} \]

\[ x_1 < 0 \]

\[ g_{ij} \]

\[ y^2 = x_1^2 + x_2^2 \]

\[ R_{ij} = \text{if } -1 \leq x < 0 \text{ then } y_2^2 + x = 0 \lor y_1 = 1 \]

\[ h^1_i \]

\[ y_1 = 1 \]

\[ h^2_{ij} \]

\[ g_{ij} \]

\[ y_2^2 + x = 0 \]

\[ -1 \leq x < 0 \]

Figure 3: Two examples of placing \( P_o \) and \( E_o \) into cells. The cells containing the elements of \( E_o \) have double line borders.

The expressions \( P_o \) and \( E_o \) are built from the other expressions, all the expressions from which \( P_o \) and \( E_o \) are constructed are held in the cells \( h^1_{i1}, \ldots, h^o_{in}, g_o \), where \( o = (i_1, \ldots, i_n) \) (see Figure 3).

The following two important properties are assumed:

- each cell may hold either a part of \( P_o \) or a part of \( E_o \), but not both.
- the distribution of \( P_o \) and \( E_o \) into appropriate cells is independent of \( o \).

In other words, each table component, a header or grid, can either hold only the elements used to define \( P_o \)'s or the elements used to define \( E_o \)'s.

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

\( Guards \neq \emptyset \), \( Values \neq \emptyset \), \( Components = Guards \cap Values \), \( Guards \cap Values = \emptyset \).

We also define \( Guards_o = Guards \cap Components_o \), and \( Values_o = Values \cap Components_o \), \( o \in I \).

The \( Guards_o \) contains elements of \( P_o \), \( Values_o \) contains elements of \( E_o \). There is only one grid \( G \), so it may belong to either \( Values \) or \( Guards \).
The definition of \textit{Guards} and \textit{Values} enables us to introduce the concept of a \textit{cell connection graph}\textsuperscript{2}

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

Let \(\rightarrow\) be a relation \(\rightarrow \subseteq \text{Components} \times \text{Components}\) satisfying:

\[
\forall A, B \in \text{Components} \quad A \rightarrow B \Rightarrow ((A = G \lor B = G) \land A \neq B). \quad (1)
\]

In other words, each arc that represents \(\rightarrow\) must either start from or end at the grid \(G\).

The relation \(\rightarrow^*\), transitive and reflexive closure of \(\rightarrow\), is a \textit{partial order} [12], so we can talk about both maximal and minimal elements w.r.t. \(\rightarrow^*\).

The relation \(\rightarrow\) is a \textit{cell connection graph} if

1. \(A\) is maximal w.r.t. \(\rightarrow^*\) \(\Rightarrow A \in \text{Values}\),
2. \(A\) is minimal w.r.t. \(\rightarrow^*\) \(\Rightarrow A \in \text{Guards}\),
3. \(\forall A \in \text{Guards}(T), \forall B \in \text{Values}(T). \quad A \rightarrow^+ B\).

The cell connection graph \(\rightarrow\) represents \textit{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 we expect \(A \rightarrow^+ B\), where \(\rightarrow^+\) is the transitive closure of \(\rightarrow\). This means that \textit{the components built from the cell describing the domains are never maximal}, while \textit{the components built from the cells containing formulae for values are never minimal}.

One can also easily prove the following Lemma.

\textbf{Lemma 2.1}

\textit{Only the grid \(G\) can be neutral, and there exists at most one neutral component.}\n
\textbf{\(\blacksquare\)

There are six “topologically” different types of cell connection graphs.

\textsuperscript{2}In earlier papers [1, 22, 23], the cell connection graph was introduced first and the partition of \textit{Components} later.
Type 1. Each element is either maximal or minimal. There is only one maximal element.

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

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

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

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

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

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

It turns out that:

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

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

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

The triple

\[
TSK = (Components, Guards, Values)
\]

will be called a Table Skeleton. A table skeleton represents the structure of a tabular expression that is independent of the particular values of \(R_o\)'s. To define tabular
Type 1. Each element is either maximal or minimal. There is only one maximal element.

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

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

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

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

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

Figure 4: Six different types of cell connection graphs (n = 3).
expressions completely we have to precisely describe how particular cells are filled, how \( P_a \) and \( E_a \) should be constructed from the contents of appropriate cells.

Recall the idea we were using is the following:

- the expressions defining the \textit{relational expression’s} \( E_a \)’s are held in value cells \( Values \).

- the expressions defining the \textit{predicate expression’s} \( P_a \)’s are held in guard cells \( Guards \).

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

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

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

- A predicate expression \( PR(B_1, \ldots, B_r), \) where \( B_1, \ldots, B_r \) are variables, is called a \textit{table predicate rule}. 


A relation expression \( RR(A_1, \ldots, A_s) \), where \( A_1, \ldots, A_s \) are variables, is called a \textit{table relation rule}.

The predicate \( P_\alpha, \alpha \in \mathcal{I} \) can now be derived from \( PR(B_1, \ldots, B_s) \) by replacing each variable \( B_i \) by the content of the cell that belongs to \( \{B_i\} \cap \text{Guard}_\alpha \). Similarly, the relation expression \( E_\alpha \) can now be derived from \( RR(A_1, \ldots, A_s) \) by replacing each variable \( A_i \) by the content of the cell that belongs to \( \{A_i\} \cap \text{Values}_\alpha \).

More detailed forms of table predicate rules and table relation rules are discussed in the Section 4. The table predicate and relation rules are sufficient to understand how the expressions \textbf{if} \( P_\alpha \) \textbf{then} \( E_\alpha \) can be built from the contents of appropriate cells. We still do not know how the relation \( R \) should be built from all \( R_\alpha \)’s.

A relation expression \( CR \) of the form \( R = \bigvee_{\alpha \in \mathcal{I}} R_\alpha \) is called a \textit{table composition rule}.

In general, \( \bigvee_{\alpha \in \mathcal{I}} R_\alpha \) is a relational expression built from the expressions defining \( R_\alpha \)’s, and various relational operators. We shall discuss it in detail in Section 4.

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

- A \textit{tabular expression (or table)} is a tuple
  \[
  T = (TSK, PR, RR, CR, IN, OUT)
  \]

  where \( TSK \) is a table skeleton, \( PR, RR, CR \) are respectively table predicate rule, table relation rule, and table composition rule, \( \Psi \) 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 \( IN \times OUT \), where \( IN \) is the set (usually heterogeneous Cartesian product) of inputs, and \( OUT \) is the set (usually heterogeneous Cartesian product) of outputs.

For every tabular expression \( T \), we define the \textit{signature of \( T \)} as:

\[
Sign_T = (PR, RR, CR, \longrightarrow).
\]

The signature describes all the global and structural information about the table. We may say that a tabular expression is a triple: \textit{signature, skeleton} - which describes the number of elements in headers and indexing of the grid, and the mapping \( \Psi \) - which describes the content of all cells.
Examples of tables are presented in Figures 6, 7 and 8. The signatures enriched by information about variables are presented separately in special two column tables. The above definitions describes, more or less, the syntax of tables. However the word ‘syntax’ here has the meaning closer to that used in Linguistics than in Mathematics and Computer Science. In general $\Psi$ 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 $\Psi$.

Let $I$ be the index of $T$, let

$$
P_\alpha = PR[\Psi(e_i)/B_1, ... , \Psi(e_s)/B_s]$$

$$
E_\alpha = RR[\Psi(d_i)/A_1, ... , \Psi(d_r)/A_r]
$$

(2)

where $e_i = B_i \cap Guards, \ i = 1, ..., s$, and $d_i = A_i \cap Values, \ i = 1, ..., r$,

$PR[\Psi(e_1)/B_1, ... , \Psi(e_s)/B_s]$ is obtained from $PR$ by replacing each $B_i$ by $\Psi(e_i)$, and similarly for $RR[...]$.

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

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

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

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

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

where $R_\alpha = if \ P_\alpha \ then \ E_\alpha$.

Figures 6, 7 and 8 illustrate the above definitions.

3 Table Predicate Rules and Table Relation Rules

The predicate expression $PR$ is built from table component names (variables) $B_1, ..., B_r$, where $Guards = \{B_1, ..., B_r\}$, logical operators “∧”, “∨”, “¬” (however “¬” is at present disallowed for implementation reasons in the SERG tool package [1, 43]), the replacement operator, some constant and relation symbols. The replacement operator is of the form $E[E_1/x]$, where $E, E_1$ are expressions, $x$ is a variable or constant, and $E[E_1/x]$ represents a new expression derived from $E$ by replacing every
\[ f(x, y) = \begin{cases} 0 & \text{if } x \geq 0 \land y = 10 \\ x & \text{if } x < 0 \land y = 10 \\ y^2 & \text{if } x \geq 0 \land y > 10 \\ -y^2 & \text{if } x \geq 0 \land y < 10 \\ x + y & \text{if } x < 0 \land y > 10 \\ x - y & \text{if } x < 0 \land y < 10 \end{cases} \]

**Figure 6:** Two examples of tabular expressions - normal (above) and inverted (below)
\( (x_1, x_2) R(y_1, y_2), y_3) \iff \begin{cases} 
  y_1 = x_1 + x_2 \land y_2 x_1 - x_2 = y_2^2 \\
  \land y_3 + x_1 x_2 = |y_3| \\
  y_1 = x_1 - x_2 \land x_1 + x_2 + x_2 y_2 = |y_2| 
\end{cases} \quad \text{if } x_2 \leq 0 \\
\begin{cases} 
  y_1 = x_1 - x_2 \land x_1 + x_2 + x_2 y_2 = |y_2| 
\end{cases} \quad \text{if } x_2 > 0

The symbol "\(=\)" after \(y_1\) in \(H_2\) indicates that the relations \(R_{i1}, i = 1, 2\), are functions. The symbol "\(|\)" after \(y_2\) and \(y_3\) in \(H_2\) indicates that \(R_{i2}\) and \(R_{i3}, i = 1, 2\), are relations with \(y_2\) and \(y_3\) as respective output variables.

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

\[ \text{input variables} \quad \begin{array}{c} \text{Temperature: hot, cold} \\
\text{Weather: sunny, cloudy, rain} \\
\text{Windy: true, false} \end{array} \quad \begin{array}{c} \text{output variables} \\
\text{\(\varphi\): go sailing, go to the beach, play bridge, garden} \end{array} \]

\[ \text{Function name} \quad \varphi \quad \text{notation} \quad * = \text{don’t care} \]

\[ H_1 \]

\[ H_2 \quad \begin{array}{c} \text{Temperature } \in \{ \text{hot, cool}\} \\
\text{Weather } \in \{ \text{sunny, cloudy, rain}\} \\
\text{Windy } \in \{ \text{true, false}\} \end{array} \quad \begin{array}{c} \text{go sailing} \quad \text{go to the beach} \quad \text{play bridge} \quad \text{garden} \end{array} \]

\[ G \quad \begin{array}{c} \text{hot} \quad \text{cool} \quad \text{sunny} \quad \text{cloudy} \quad \text{rain} \quad \text{cloudy} \end{array} \]

Figure 7: Next two examples of tabular expressions - vector table (above) and decision table [20] (below)
Figure 8: Another two examples of tabular expressions - generalized decision (above) and type 4 (below).
occurrence of \( x \) in \( E \) by \( E_1 \). The constants and relation symbols depend on the type of input domain of the relation specified, \( \text{dom}(R) \). The relation symbol \( "\Rightarrow" \) can always be used. If the elements of \( \text{dom}(R) \) are ordered, the relation symbols \( "<" \), \( "\gg" \) can be used\(^3\).

The relation expression \( RR \) is built from table component names \( A_1, ..., A_r \) (variables), where \( \text{Values}(T) = \{ A_1, ..., A_r \} \), set operators \( \cup \), \( \cap \), etc., relation operators \( "\Rightarrow" \), \( "<" \), \( "\gg" \), etc., the operator of \( \text{concatenation} \) \( \circ \)\(^4\).

## 4 Composing \( R \) from \( R_a \)

One of the fundamental assumptions behind the concept of tabular expressions is that the relation \( R \) specified by a tabular expression can be composed from the relations \( R_a, \ a \in I \), where all \( R_a \)'s are described in appropriate guard and value cells i.e. \( \forall a \in I \). In this section we study some operations that can be regarded as \( \forall_a \).

We start with introducing some basic concept of the algebra of relation. The next subsection contain the classical definitions and results ([40]). The new concepts and results start from subsection 4.2, where the concept of "being a part of" and some new operations are discussed.

### 4.1 Basic Elements of Relation Algebra

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

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

\[
\text{dom}(R) = \{ x \mid \exists y \in Y. (x, y) \in R \},
\]

\[
\text{range}(R) = \{ y \mid \exists x \in X. (x, y) \in R \},
\]

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

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

\(^4\)For example for Figure 6 we have \( ((y_1 = 1 \circ (x_1 + x_2)) = (y_1 = x_1 + x_2), ((y_0 \circ (y_0 + x_1 x_2 = \lfloor y_0 \rfloor)) = (y_0 \mid y_0 + x_1 x_2 = \lfloor y_0 \rfloor), where (y_0 \mid y_0 + x_1 x_2 = \lfloor y_0 \rfloor) \) means that \( y_0 \) is the (only) output variable in the expression \( y_0 + x_1 x_2 = \lfloor y_0 \rfloor \).
We have five basic relational operations: supremum (union), infimum (intersection), complement, inverse and composition. Let $P_{A^{ab} B}$ and $Q_{B^{ac} C}$ be two relations.

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

\[
P_{A^{ab} B} \cup Q_{B^{ac} C} = \begin{cases} 
\{ (x, y) \mid (x, y) \in P_{A^{ab} B} \lor (x, y) \in Q_{B^{ac} C} \} & \text{if } A = C \\
\text{undefined} & \text{otherwise.}
\end{cases}
\]

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

\[
P_{A^{ab} B} \cap Q_{B^{ac} C} = \begin{cases} 
\{ (x, y) \mid (x, y) \in P_{A^{ab} B} \land (x, y) \in Q_{B^{ac} C} \} & \text{if } A = C \\
\text{undefined} & \text{otherwise.}
\end{cases}
\]

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

\[
\overline{P}_{A^{ab} B} = \{ (x, y) \mid x \in A \land y \in B \land (x, y) \notin P_{A^{ab} B} \}
\]

- The **converse** of relation $P_{A^{ab} B}$, $P_{A^{ab} B}^c$, is defined as:

\[
P_{A^{ab} B}^c = \{ (x, z) \mid (z, x) \in P_{A^{ab} B} \}.
\]

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

\[
P_{A^{ab} B} \circ Q_{C^{ac} D} = \begin{cases} 
\{ (x, y) \mid \exists z \in B \cdot ((x, z) \in P_{A^{ab} B} \land (z, y) \in Q_{C^{ac} D}) \} & \text{if } B = C; \\
\text{undefined} & \text{otherwise.}
\end{cases}
\]

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

We have three special kind of relations: identity, universal relation, and the empty relation.
• For every set $A$, the relation $I_{A\leftrightarrow A} = \{(x, x) \mid x \in A\}$ is called the identity on $A$ (of type $A \leftrightarrow A$).

• For every two sets $A, B$, the relation $\top_{A\leftrightarrow B} = A \times B$ is called the universal relation (of type $A \leftrightarrow B$).

• For every two sets $A, B$, the relation $\bot_{A\leftrightarrow B} = \{(x, y) \mid \text{false}\}$ is called the empty relation (of type $A \leftrightarrow B$).

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

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

• The relation $R_{A\leftrightarrow B}$ is total if $\text{dom}(R_{A\leftrightarrow B}) = A$.

• The relation $R_{A\leftrightarrow B}$ is a function (univalent relation) if 
  \[ \forall x \in A, \forall y, z \in B. \ (x, y) \in R \land (x, z) \in R \Rightarrow y = z. \]

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

**Corollary 4.1** For each relation $R$ :

1. $R$ is total $\iff \top = R \cap \top$;

2. $R$ is function $\iff R^\circ : R \subseteq I \iff \top : R \subseteq \top R$.

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

• For every set $A \subseteq X$ and every relation $R_{X\leftrightarrow Y}$ we define a relation $R|_A \subseteq X \times Y$, restriction of $R$ to $A$, as:
  \[ \forall x \in X, \forall y \in Y. \ (x, y) \in R|_A \iff x \in A \land (x, y) \in R. \]

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

For the rest of the paper, we assume $T$ to be an universal set of indexes and $\{D_i \mid i \in T\}$ to be an appropriate set of domains.

We shall also use the following notation:

18
• For every $I \subseteq T$, let $D_I = \prod_{i \in I} D_i$, where $\prod_{i \in I}$ is a direct product over $I$.

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

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

We will always write explicitly $f|_K$ to denote the restriction of the function $f$ to $K$. We shall now recall the concept of a projection.

• Let $J \subseteq I \subseteq T$. The projection from $D_I$ onto $D_J$, denoted by $\iota \Pi_J$, is defined as:

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

Note that $\iota \Pi_J$ is a relation, $\iota \Pi_J : D_I \leftrightarrow D_J$. From this definition, it follows immediately that, $\iota \Pi_J = 1_{D_I \times D_J}$, and $\iota \Pi_J : \Pi_J = 1_{D_I \times D_J}$ and $\iota \Pi_J$ is total.

### 4.2 The Relation Part of

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

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

Intuitively, in most cases, $R_a$ is a part of $R$.

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

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

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

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

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

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

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

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

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

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

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

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

- Let $i_\delta_j$, called a *cylindrification relation*, be the following relation

$$i_\delta_j = i_{\Pi_j} \cap i_{\Pi_j^c}$$

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

Clearly we have $i_\delta_j = i_\delta_j^c$, and if $K \subseteq J$ then $i_\delta_j \cap i_\delta_K = i_\delta_K$.

The relation $i_\delta_j$ can also be defined element-wise.

**Lemma 4.2**

$$(f, g) \in i_\delta_j \iff f \in D_I \land g \in D_I \land f |_J = g |_J.$$  

**Proof.** Directly from the definition of projection and composition.
Corollary 4.3 Let $I, J, K$, and $L$ be subsets of $T$ such that $K \subseteq I$ and $L \subseteq J$. Let $P : D_K \leftrightarrow D_L$ and $Q : D_J \leftrightarrow D_J$ be two relations.

1. $\pi_{K : I} ; P : \pi_{L : J} = \{(f, g) \mid f \in D_I \land g \in D_J \land (f|_K, g|_L) \in P\}$

2. $\delta_K : Q : \pi_{L : J} = \{(f, g) \mid f \in D_I \land g \in D_J \land \exists (h, t) \in Q : (h|_K = f|_K \land t|_L = g|_L)\}$

We can now define $\subseteq$ in terms of projections and cylindricals.

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

$$P \subseteq Q \iff \pi_{K : I} ; P : \pi_{L : J} \subseteq \pi_{K : I} ; Q : \pi_{L : J}.$$ 

Proof.

$$\left\{ (f, g) \mid f \in D_I \land g \in D_J \land (f|_K, g|_L) \in P \right\} \subset \left\{ (f, g) \mid f \in D_I \land g \in D_J \land \exists (g_1, g_2) \in Q : (g_1|_K = f|_K \land g_2|_L = g|_L) \right\}$$

$$\Rightarrow \quad \text{(relabelling $f|_K$ and $g|_L$ as $f_1$ and $f_2$, respectively)}$$

$$\forall (f_1, f_2) \in P, \exists (g_1, g_2) \in Q : (g_1|_K = f_1 \land g_2|_L = f_2)$$

$$\Rightarrow \quad \text{(definition of $\subseteq$)}$$

$$P \subseteq Q$$

$\blacksquare$

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

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

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

- $P \oplus Q = \{(f, g) \in D_{I \cup K} \times D_{J \cup L} \mid (f|_I, g|_J) \in P \land (f|_K, g|_L) \in Q\}$
- $P \otimes Q = \{(f, g) \in D_{I \cup K} \times D_{J \cup L} \mid (f|_I, g|_J) \in P \lor (f|_K, g|_L) \in Q\}$
- $P \ominus Q = \{(f, g) \in D_{I \cup K} \times D_{J \cup L} \mid (f|_I, g|_J) \in P \land (f|_K, g|_L) \not\in Q\}$
Let $P : X_1 \times X_3 \leftrightarrow X_5$ and $Q : X_1 \times X_2 \leftrightarrow X_4$ where $X_1 = X_2 = X_3 = X_4 = X_5 = \text{Reals}$. Suppose that

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

and

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

Then we have

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

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

and

$$P \ominus Q = \{((x_1, x_2, x_3), (x_4, x_5)) \mid ((x_1, x_3), x_5) \in P \land ((x_1, x_2), x_4) \not\in Q\}.$$

**Lemma 4.5**

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

1. $P \oplus Q = \biguplus_{i \in I} P_i \bigcup_{j \in J} P_j \bigcap_{k \in K} Q_k \bigcap_{l \in L} Q_l$

2. $P \otimes Q = \biguplus_{i \in I} P_i \bigcap_{j \in J} P_j \bigcap_{k \in K} Q_k \bigcap_{l \in L} Q_l$

3. $P \ominus Q = \biguplus_{i \in I} P_i \bigcup_{j \in J} P_j \bigcup_{k \in K} Q_k \bigcup_{l \in L} Q_l$

**Proof.**

1. $P \oplus Q = \biguplus_{i \in I} P_i \bigcup_{j \in J} P_j \bigcap_{k \in K} Q_k \bigcap_{l \in L} Q_l$

   \[
   = \{ (f, g) \in D_I \times D_J \mid (f|_I, g|_J) \in P \lor (f|_K, g|_L) \in Q \}
   \]

   The proofs of 2 and 3 are similar. \qed

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

22
Lemma 4.6

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

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

The proof follows from the fact that $\Pi_j$ is total and univalent.

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

Lemma 4.7

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

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

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

1. $P \odot (Q \oplus R)$
   
   $= (\text{definition of } \oplus)$
   
   $P \odot \left( (\cup_{(KUM)} \Pi_K : Q_{(LUN)} \Pi^-_L \cup \cup_{(KUM)} \Pi_M : R_{(LUN)} \Pi^-_N \right)$
   
   $= (\text{definition of } \odot)$
   
   $\cap_{(KUM)} \Pi_I : P_{(LUM)} \Pi^-_J$
   
   $\cap_{(KUM)} \Pi_K : Q_{(LUN)} \Pi^-_L \cup \cap_{(KUM)} \Pi_M : R_{(LUN)} \Pi^-_N$
   
   $= (\cup - \text{distributivity of } \cap \& \cup - \text{distributivity of } \cap)$
   
   $\cap_{(KUM)} \Pi_I : P_{(LUM)} \Pi^-_J$
   
   $\cap_{(KUM)} \Pi_K : Q_{(LUN)} \Pi^-_L \cup \cap_{(KUM)} \Pi_M : R_{(LUN)} \Pi^-_N$
   
   $= (\text{for } K \subseteq J \subseteq I \text{ we have } \Pi_J : \Pi_K = \Pi_K$
   
   $\& (P \Pi Q) = Q^- P^\sim$)
   
   $= (\text{for } K \subseteq J \subseteq I \text{ we have } \Pi_J : \Pi_K = \Pi_K$
   
   $\& (P \Pi Q) = Q^- P^\sim$)

23
\[
(\bigcup_{i \in \mathcal{M}} \Pi_i \cap \bigcup_{j \in \mathcal{K}} \Pi_j^c : P')_{(\mathcal{U} \cap \mathcal{K})} \cap (\bigcup_{i \in \mathcal{M}} \Pi_i) : Q_{(\mathcal{U} \cap \mathcal{K})} \cap (\bigcup_{j \in \mathcal{K}} \Pi_j^c) : R_{(\mathcal{U} \cap \mathcal{K})} \cap (\bigcup_{i \in \mathcal{M}} \Pi_i) : (Q \cap R)_{(\mathcal{U} \cap \mathcal{K})} \cap (\bigcup_{j \in \mathcal{K}} \Pi_j^c) : \hat{Q}_{(\mathcal{U} \cap \mathcal{K})}
\]

= \langle \text{definition of } \otimes \rangle
\]
\[
(\bigcup_{i \in \mathcal{M}} \Pi_i \cap \bigcup_{j \in \mathcal{K}} \Pi_j^c : (P \otimes Q) \cap (\bigcup_{i \in \mathcal{M}} \Pi_i) : (Q \cap R)_{(\mathcal{U} \cap \mathcal{K})} \cap (\bigcup_{j \in \mathcal{K}} \Pi_j^c) : \hat{Q}_{(\mathcal{U} \cap \mathcal{K})}
\]

2. The proof is similar to the previous, except that we use the following facts:
\[P : (Q \cap R) : P^c = P : Q : P^c \cap P : R : P^c.\]

\[\Box\]

4.4 Classification on the Basis of Table Composition Rule

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

- The table is called \textit{plain} if \( R = \bigcup_{a \in A} R_a. \)
- The table is called \textit{output-vector} if \( R = \bigotimes_j \bigoplus_i R_{i,j}. \)
- The table is called \textit{input-vector} if \( R = \bigoplus_i \bigotimes_j R_{i,j}. \)

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

5 Final Comment

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

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

Acknowledgments

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

References


Appendix

This Appendix contains some illustrative examples of direct product, restriction operation, projection and cylindification relations. We start with the example of a direct product.
Example 5.1 Let \( T = \{ u, v, w \} \), \( D_u = \{ 1, 2 \} \), \( D_v = \{ \alpha, \beta \} \) and \( D_w = \{ m, n \} \).

\[
D_T = \prod_{i \in T} D_i = D_u \times D_v \times D_w = \{ (a, b, c) \mid a \in D_u \land b \in D_v \land c \in D_w \}
\]

The set \( D_T \) can be seen as the following relation.

\[
\left\{ f \mid f : \{ u, v, w \} \to \{ 1, 2, \alpha, \beta, m, n \} \land f(u) \in D_u \land f(v) \in D_v \land f(w) \in D_w \right\}
\]

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

\[
\begin{array}{ccc}
\begin{array}{ccc}
1 & \alpha & m \\
1 & \beta & m \\
2 & \alpha & m \\
2 & \beta & m \\
\end{array} & \leftrightarrow & \begin{array}{ccc}
1 & m & \alpha \\
1 & n & \alpha \\
1 & m & \beta \\
1 & n & \beta \\
\end{array} & \begin{array}{ccc}
1 & \alpha & m \\
1 & \alpha & n \\
1 & \beta & m \\
1 & \beta & n \\
\end{array}
\end{array}
\]

The next example is an example of a function restriction.

Example 5.2 If \( I = \{ 1, 2, 3, 4 \} \), \( K = \{ 2, 4 \} \), \( D_I = X_1 \times X_2 \times X_3 \times X_4 \), \( f = (x_1, x_2, x_3, x_4) \in D_I \) (or \( f : \{ 1, 2, 3, 4 \} \to D_I \), \( f(i) = x_i \), \( i = 1, 2, 3, 4 \)), then \( f_K : \{ 2, 4 \} \to D_2 \cup D_4 \), \( f_K(i) = x_i \), \( i = 2, 4 \), i.e. \( f_K = (x_2, x_4) \).

We shall now illustrate the concept of projection relation.

Example 5.3 We continue with the terms of Example 5.1. Let us take \( I = \{ v, w \} \). The following tables illustrate the relation \( \pi I \). A row from the first table represents a function \( f \in D_T \) and the corresponding row from the second table represents a function \( g \in D_I \). Hence, a whole row represents a member of \( \pi I \), i.e., third row, \( ((1, \beta, m), (\beta, m)) \in \pi I \).
\[
\begin{array}{|c|c|c|}
\hline
f(u) & f(v) & f(w) \\
\hline
1 & \alpha & m \\
1 & \alpha & n \\
1 & \beta & m \\
1 & \beta & n \\
2 & \alpha & m \\
2 & \alpha & n \\
2 & \beta & m \\
2 & \beta & n \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
g(v) & g(w) \\
\hline
\alpha & m \\
\alpha & n \\
\beta & m \\
\beta & n \\
\alpha & m \\
\alpha & n \\
\beta & m \\
\beta & n \\
\hline
\end{array}
\]

In this case we can either write:

\[
\tau \Pi_I = \{ (f, g) \mid f \in D_T \land g \in D_I \land f(v) = g(v) \land f(w) = g(w) \},
\]

or, by using tuples instead of functions,

\[
\tau \Pi_I = \{ ((a, b, c), (b', c')) \mid b' = b \land c' = c \}.
\]

The last example is an example of cydlnification relation.

**Example 5.4** Let us take the projection \( \tau \Pi_I \) of Example 5.3. We have

\[
\tau \delta_I = \tau \Pi_I ; \tau \Pi_I
\]

= \langle Example 5.3 \rangle

\[
\{(a, b, c), (b', c') \mid b' = b \land c' = c \} ; \{(b, c), (a', b', c') \mid b' = b \land c' = c \}
\]

= \langle definition of ; \rangle

\[
\{(a, b, c), (a', b', c') \mid b' = b \land c' = c \}
\]

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

30
<table>
<thead>
<tr>
<th></th>
<th>α</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>α</td>
<td>m</td>
</tr>
<tr>
<td>1</td>
<td>α</td>
<td>n</td>
</tr>
<tr>
<td>1</td>
<td>α</td>
<td>n</td>
</tr>
<tr>
<td>1</td>
<td>β</td>
<td>m</td>
</tr>
<tr>
<td>1</td>
<td>β</td>
<td>m</td>
</tr>
<tr>
<td>1</td>
<td>β</td>
<td>n</td>
</tr>
<tr>
<td>1</td>
<td>β</td>
<td>n</td>
</tr>
<tr>
<td>2</td>
<td>α</td>
<td>m</td>
</tr>
<tr>
<td>2</td>
<td>α</td>
<td>m</td>
</tr>
<tr>
<td>2</td>
<td>α</td>
<td>n</td>
</tr>
<tr>
<td>2</td>
<td>α</td>
<td>n</td>
</tr>
<tr>
<td>2</td>
<td>β</td>
<td>m</td>
</tr>
<tr>
<td>2</td>
<td>β</td>
<td>m</td>
</tr>
<tr>
<td>2</td>
<td>β</td>
<td>n</td>
</tr>
<tr>
<td>2</td>
<td>β</td>
<td>n</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>α</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>α</td>
<td>m</td>
</tr>
<tr>
<td>2</td>
<td>α</td>
<td>m</td>
</tr>
<tr>
<td>1</td>
<td>α</td>
<td>n</td>
</tr>
<tr>
<td>1</td>
<td>α</td>
<td>n</td>
</tr>
<tr>
<td>1</td>
<td>β</td>
<td>m</td>
</tr>
<tr>
<td>1</td>
<td>β</td>
<td>m</td>
</tr>
<tr>
<td>1</td>
<td>β</td>
<td>n</td>
</tr>
<tr>
<td>1</td>
<td>β</td>
<td>n</td>
</tr>
<tr>
<td>2</td>
<td>α</td>
<td>m</td>
</tr>
<tr>
<td>2</td>
<td>α</td>
<td>m</td>
</tr>
<tr>
<td>2</td>
<td>α</td>
<td>n</td>
</tr>
<tr>
<td>2</td>
<td>α</td>
<td>n</td>
</tr>
<tr>
<td>2</td>
<td>β</td>
<td>m</td>
</tr>
<tr>
<td>2</td>
<td>β</td>
<td>m</td>
</tr>
<tr>
<td>2</td>
<td>β</td>
<td>n</td>
</tr>
<tr>
<td>2</td>
<td>β</td>
<td>n</td>
</tr>
</tbody>
</table>