Table Transformations: Theory and Tools[1]

J.I. Zucker

Department of Computing and Software,
McMaster University, Hamilton, Ontario L8S 4L7, Canada
zucker@mcmaster.ca

H. Shen[2]

Fulcrum Technologies Inc.
Ottawa, Ontario K1S 5H4, Canada
Hong.Shen@fulcrum.com

Abstract

We work in a theory of function tables, similar to, and inspired by, that given in the work of D. Parnas. Table transformation algorithms transform one kind of table into another, preserving the semantics. We consider, in particular, two kinds of function tables: normal and inverted. We study effective transformations between tables of these two kinds, as well as transformations which change the dimension of a table. We also consider the interrelationship between these three types of transformation.

Most of these algorithms have been implemented as part of the Table Tool System developed by the Software Engineering Research Group at McMaster University. Some of the issues related to implementation are discussed.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications — *methodologies*; D.2.2 [**Software Engineering**]: Design Tools and Techniques — *decision tables*

General Terms: Documentation

Additional Key Words and Phrases: Table transformations, tabular specifications

[1] Research supported by funding by the Telecommunications Research Institute of Ontario, and by the Natural Sciences and Engineering Research Council of Canada.

[2] Work done while employed at the Software Engineering Research Group at McMaster University

Typeset by $\mathcal{A}_{\mathcal{M}}\mathcal{S}$-TEX

1 Introduction

1.1 Tabular notation in documenting software systems

The method of tabular representations, developed by David Parnas and his collaborators, has been found to be very useful for the formal documentation and inspection of software systems.

The first application of this technique was in the documentation for the revised flight software for the US Navy's A-7 aircraft in the late seventies [HKPS78, Hen80]. Another large project which used the tabular notation was the documentation of the shutdown systems of the Darlington Nuclear Power Generating Station in Ontario, Canada, required by the Atomic Energy Control Board of Canada for that station's licensing, in the late eighties [PAM91, Par94].

The tabular notation is a useful and perspicuous method for defining functions on many sorted algebras. In the course of the projects described above, many kinds of tables were developed, and were found to be useful. A systematic exposition of ten kinds of tabular expressions was given in [Par92].

A natural question which arises is: Given a function, what is the best kind of table to represent it? It has been found that in practice one cannot always predict this. However a better question is: What is the best kind of table to represent a given function *for a specific purpose?* It turns out that the different kinds of table are best for different purposes, namely for developing the specifications, or for testing them, or for using the table. An example is given below, in §1.2. It therefore becomes important to be able to transform the format of a table from one kind to another. Furthermore, such a transformation should be effected mechanically, *i.e.*, implemented by a software tool.

In [Zuc96] two of the kinds of tables enumerated in [Par92] *normal* and *inverted*, were considered. Various algorithms for effectively transforming one kind of table to the other were defined, and some of their interrelationships were studied.

In [She95] the implementation of these transformations is described within the framework of the Table Tool System developed by the Software Engineering Research Group at McMaster University [SERG97]. In addition, another transformation for normalising inverted tables is defined, which (unlike the transformation in [Zuc96]) preserves dimensionality.

The present paper brings together these two investigations. The outline of this paper is as follows. In the remaining subsection (§1.2) of this Introduction, we present an actual example of a function in a software project which was originally defined by a table of one kind, and then transformed by hand to a simpler table of another kind. This demonstrates the significance of table transformations, and the importance of implementing them.

Section 2 contains preliminary material on the formalism of tabular notations, based on many-sorted signatures, and the syntax and semantics of normal and inverted function tables.

An important semantic condition is defined for normal and inverted tables, namely *properness*, which guarantees reasonable semantics. A stricter condition on inverted tables, namely *strict properness*, is needed for some of the results of Sections 3 and 4. An important concept for tables is the relation of *semantic equivalence*:

Two proper tables are semantically equivalent iff they define the same function.

In Section 3 various algorithms for tabular transformations are reviewed. These include algorithms for lowering the dimensionality ("flattening") of a table, inverting a normal table, and normalising an inverted table. In the last case two algorithms are considered: "one-dimensional" (1-D) and "many-dimensional" (m-D) normalisation, which were studied, respectively, in [Zuc96] and [She95]. The latter algorithm, which involves refining the partitioning of the function domain, has the advantage of preserving dimensionality, but the disadvantage of greater complexity.

In Section 4 the interrelationships between these transformations are studied, extending the combined results of [Zuc96] and [She95]. We show:

(1*a*) inversion of a normal table followed by 1-D normalisation is elementarily equivalent to 1-D flattening;

(1*b*) conversely, 1-D normalisation of an inverted table followed by inversion is elementarily equivalent to 1-D flattening;

(2*a*) inversion of a normal table followed by m-D normalisation is elementarily semantically equivalent to the original table;

(2*b*) conversely, m-D normalisation of an inverted table followed by inversion is elementarily semantically equivalent to the original table;

(3*a*) inversion of a normal table commutes with slicing up to elementary equivalence;

(3*b*) m-D normalisation of an inverted table commutes with slicing up to elementary semantic equivalence.

Here *slicing* means constructing a "slice" of a table, *i.e.*, a subtable formed by holding certain dimensions fixed; *elementary equivalence* (used in (1) and (3*a*)) is a decidable equivalence relation between tables which is a refinement of semantic equivalence; and *elementary semantic equivalence* (used in (2) and (3*b*)) is a (generally undecidable) equivalence relation between tables, which extends elementary equivalence, but is also a refinement of semantic equivalence.

Section 5 gives some information on the The Table Transformation Tool, incorporating a number of table transformation algorithms, which was developed as part of the Table Tool System (TTS) at McMaster University [SERG97].

Section 6 contains conclusions and ideas for future work.

1.2 One function in two table formats

In practice, we often find that tables of two different forms can represent the same function. Often, when we construct a table, we may not know at the outset which table form is better, i.e. more compact and/or perspicuous, with regard to the subject matter. As the table is being developed, it may turn out to be larger or more unwieldy than expected. We will then want to know what the table would look like in other forms, which may be more suitable for the problem at hand.

The following is an example of this situation from a large software project. The software requirements document for the A-7E Aircraft [AFB$^+$92] is one of many publications of the Software Cost Reduction (SCR) project conducted by the U.S. Naval Research Laboratory. The following two tables represents one of the functions defined in this document.

The HSI (Horizontal Situation Indicator) is an instrument that displays course, heading, distance and bearing information to the pilot. Our illustrative function defines an external reference point which is used to measure the HSI range and bearing as a function of different modes and conditions. In the A-7E requirements specification document, a mode corresponds to a subset of the set of possible system states of the Operational Flight Program.

In constructing a table for this function, we found that there were four possible outside reference points for the HSI: !*OAP*!, !*Fly-to-point*!, !*target*! and none. In six distinct groups of modes, these reference points are determined by a set of four conditions: /*FLYTOTW*/=0, !*Desig*!, !*Before slewing*! and !*Range*!>30. Just as in designing a flip-flop in digital circuits, we could describe the function in the form of a normal table by listing all the combinations of conditions. The result was the 16×6 normal table shown as Table 1.1.

After producing this table, it was noticed that the same function could also be described by an inverted table, of size 4×6 (the number of outside reference points times the number of group modes), as in Table 1.2. This table, in a more compact form, tells the reader directly which reference point will be taken under a given mode, and so is more convenient for the user. However Table 1.1 also has its uses: in the original development of the table, and in testing it according to the specifications.

When the A-7E requirements were originally written in 1977, the compact form of Table 1.2 was prepared by hand from Table 1.1 [AFB$^+$92, p. 368]. Clearly, a tool for mechanical transformations of this kind would be preferable.

The Table Transformation Tool was developed at McMaster University for just this purpose [SERG97].

The aim of the present paper is to provide the theoretical framework to support such transformation tools.

2 Definitions and basic concepts

2.1 Many-sorted signatures and algebras

A *(many-sorted) signature* Σ consists of a finite set of *sorts* $s, \ldots$, and a finite set of *function symbols* of fixed type, say $F : s_1 \times \cdots \times s_m \to s$, with *arity* $m \geq 0$, *domain sorts* $s_1, \ldots, s_m$ and *range sort* s. (For $m = 0$, this gives a *constant* of sort s.)

As a typical example, consider the signature $\Sigma_{\mathbb{R}}$ of reals, with two sorts: real and bool, and function symbols

$$\begin{aligned} 0,\ 1 :&\ \to \mathsf{real} \\ +,\ -,\ \times,\ \div :&\ \mathsf{real} \times \mathsf{real} \to \mathsf{real} \\ \sqrt{\ } :&\ \mathsf{real} \to \mathsf{real} \\ \wedge,\ \vee :&\ \mathsf{bool} \times \mathsf{bool} \to \mathsf{bool} \\ \neg :&\ \mathsf{bool} \to \mathsf{bool} \\ \mathsf{true},\ \mathsf{false} :&\ \to \mathsf{bool} \end{aligned}$$

This signature (or something like it) is the one most widely used in the practical work cited in this paper, as well as in the examples used throughout the paper.

Next, given a many-sorted signature Σ, a *Σ-algebra* A has, for each sort s of Σ, a (non-empty) *carrier set* A_s, and for each function symbol $F : s_1 \times \cdots \times s_m \to s$ of Σ, a function $F^A : A_{s_1} \times \cdots \times A_{s_m} \to A_s$, the *interpretation* or *meaning* of F in A.

For example, for the signature $\Sigma_{\mathbb{R}}$ given above, the $\Sigma_{\mathbb{R}}$-algebra A, which we shall call the "standard reals algebra", has the two carriers $\mathbb{R}$ (of reals) and $\mathbb{B}$ (of truth values tt and ff). Further, it has *zero* as a distinguished element of $\mathbb{R}$, the operations of *addition*, *subtraction*, *multiplication* and *division* and *square root extraction* on the reals.

Note that in this paper we will work with total functions. In this example, therefore, we must assume that the functions are made total by default values, defining, *e.g.*, $x \div 0 = 0$ for real x and $\sqrt{x} = 0$ for negative x.

All the syntax below will be defined relative to Σ, although this will not always be stated explicitly. (Thus by "sort", *e.g.*, we will mean sort of Σ.) Also all the semantics will be defined relative to a (fixed but unspecified) Σ-algebra A.

2.2 Expressions: Terms and conditions

We assume we are working in a language ***Lang***(Σ) over Σ, which produces *expressions* over Σ or *Σ-expressions*. These include the classes

- ***Term***(Σ) of Σ-terms $t, \ldots$,
- ***Cond***(Σ) of Σ-conditions $C, \ldots$,
- ***Tab***(Σ) of Σ-tables $T, \ldots$.

Terms and *conditions* are used in the construction of tables. *Tables* (as we will see) are used to define functions, "tabular functions".

We will study *terms* and *conditions* (syntax and semantics) in this section, and *tables* of two kinds (normal and inverted) in the following two sections.

2.2.1 Syntax of terms. We define the class $\boldsymbol{Term} = \boldsymbol{Term}(\Sigma)$ of terms over Σ. For each Σ-sort s, let $\boldsymbol{Term}_s$ be the class of terms of of sort s. These are generated in the well-known way, as follows. Assume given a countable set of *variables* $x^s, y^s, \ldots$ for each sort s of Σ. Then:

(a) a variable of sort s is in $\boldsymbol{Term}_s$,

(b) a constant of Σ of sort s is in $\boldsymbol{Term}_s$,

(c) if $F : s_1 \times \cdots \times s_m \to s$ $(m > 0)$ and $t_i \in \boldsymbol{Term}_{s_i}$ for $i = 1, \ldots, m$ then $F(t_1, \ldots, t_m) \in \boldsymbol{Term}_s$.

In clause (c), the symbol F may range over functions in the signature Σ, and also functions *defined* in some formalism over Σ, including, *e.g.*, *tabular functions* or functions defined by tables (as discussed below).

2.2.2 Syntax of conditions. The syntax of the class $\boldsymbol{Cond} = \boldsymbol{Cond}(\Sigma)$ of conditions over Σ does not have to be specified precisely.

Assuming that bool is one of the sorts of Σ (which is always the case in our examples), $\boldsymbol{Cond}$ could, on the one hand, be taken simply as the set of *booleans* over Σ, *i.e.*, the set $\boldsymbol{Term}_{\mathsf{bool}}$.

On the other hand, the definition of $\boldsymbol{Cond}$ could be extended to include other constructions, such as bounded quantification over integers (assuming Σ includes a sort of integers).

In fact, in order for the theory of transformations described in this paper to hold, the only assumption we need make on $\boldsymbol{Cond}$ is the following.

Closure Assumption. $\boldsymbol{Cond}$ is closed under *conjunction* and *disjunction*, and contains true and false.

It should be emphasized that whether the syntax of $\boldsymbol{Cond}$ coincides with that of $\boldsymbol{Term}_{\mathsf{bool}}$ or not, their roles are quite different, as we will see below.

2.2.3 States; Semantics of terms and conditions. We need some preliminary notions.

Definitions. (1) For any expression (term, condition or table) E, $\boldsymbol{var}(E)$ is the set of free variables in E. Similarly, $\boldsymbol{var}(E_1, \ldots, E_n)$ is the set of free variables in the expressions $E_1, \ldots, E_n$.

(2) For any set V of variables, a *state over* V (*in* A) is a function σ with domain containing V, such that for all $x \in V$, if x has sort s then $\sigma(x) \in A_s$.

(3) For any expression E, a *state over* E is a state over $\boldsymbol{var}(E)$.

Now a function symbol $F : s_1, \ldots, s_m \to s$ of Σ has an *interpretation* on A:

$$F^A : A_{s_1} \times \cdots \times A_{s_m} \to A_s.$$

given by the definition of A. (For a defined function symbol, such a semantics has to be provided. In particular, the interpretation of a function symbol defined by a table will be discussed below.)

Hence every term or condition E has a value $[\![E]\!]^A\sigma$ in A, relative to a state σ over E in A. If $E \in \boldsymbol{Term}_s$ then $[\![E]\!]^A\sigma \in A_s$, and if $E \in \boldsymbol{Cond}$ then $[\![E]\!]^A\sigma \in \mathbb{B}$. The definition of $[\![E]\!]^A$ is standard, by induction on the complexity of E. We omit the inductive clauses here.

We will often drop the superscript 'A', and not usually refer explicitly to the algebra A. Thus 'state' will mean 'state in A', etc. Finally:

Definition. If σ is a state over a condition C, then $\sigma \models C$ (σ *satisfies* C or C *holds at* σ) iff $[\![C]\!]\sigma = \mathtt{tt}$.

2.3 Normal tables

We will define the class $\boldsymbol{Tab}_N(\Sigma)$ of *normal (function) tables over* Σ.

2.3.1 Preliminary notions. (1) The set $\{1,\ldots,n\}$ of positive integers (for some $n \geq 1$) is called the *segment up to* n, denoted $\boldsymbol{seg}(n)$. Its *length* is n.

(2) An *indexed set* is a function F. The domain of F is an *index set* $I = \boldsymbol{ind}(F)$. An *index* of F is an element of I. The value of F at an index $i \in I$ is usually denoted F_i. We sometimes write a family F with index I as

$$F \;=\; \langle F_i \mid i \in I\rangle.$$

The *entries* of F are the elements of its range $\{F_i \mid i \in I\}$.

In this paper, all index sets and (hence) indexed sets will be *finite*. Also the entries of indexed sets will be *expressions*.

(3) A *variant* $F\langle i/e\rangle$ of an indexed set F, with entry e at index i, is the indexed set F' with the same index set as F, such that $F'_j = F_j$ for all indices $j \neq i$, and $F'_i = e$.

(4) A *tuple* is an indexed set for which the index set is a segment. Its *length* is the length of the segment. We write the tuple $\langle e_i \mid i \in \boldsymbol{seg}(n)\rangle$ as $(e_1,\ldots,e_n)$.

(5) Given a tuple of positive integers $(l_1,\ldots,l_n)$, the elements of the cartesian product $\boldsymbol{seg}(l_1) \times \cdots \times \boldsymbol{seg}(l_n)$ can be *enumerated* by a single segment of length $l_1 \cdot \ldots \cdot l_n$. In other words, there is a bijection between $\boldsymbol{seg}(l_1) \times \cdots \times \boldsymbol{seg}(l_n)$ and $\boldsymbol{seg}(l_1 \cdot \ldots \cdot l_n)$. Let

$$\boldsymbol{enum}_{l_1,\ldots,l_n} : \; \boldsymbol{seg}(l_1) \times \cdots \times \boldsymbol{seg}(l_n) \;\to\; \boldsymbol{seg}(l_1 \cdot \ldots \cdot l_n)$$

be any such bijection. We could, *e.g.*, take

$$\boldsymbol{enum}_{l_1,\ldots,l_n}(k_1,\ldots,k_n) \;=\; \sum_{i=1}^{n-1}((k_i - 1) \cdot l_{i+1} \cdot \ldots \cdot l_n) + k_n.$$

We will generally use the notation

$$[k_1, \ldots, k_n] \;=_{df}\; \boldsymbol{enum}_{\,l_1,\ldots,l_n}(k_1, \ldots, k_n)$$

where the parameters $l_1, \ldots, l_n$ are understood from the context.

We use the notation '$\equiv$' for *syntactic identity.*

2.3.2 Syntax of normal tables

Definition 1. A *grid* G is an indexed set for which the index set is a Cartesian product

$$\boldsymbol{ind}(G) \;=\; \boldsymbol{seg}(l_1) \times \cdots \times \boldsymbol{seg}(l_n),$$

(for some n and some $l_1, \ldots, l_n$) and the entries of G are *expressions.* (Note that a tuple is just a 1-dimensional grid.) Further, with this notation:

- the *dimensionality* of G is $\boldsymbol{dim}(G) = n$, and G is an n*-dimensional* grid;
- the *length* of G in its ith dimension is $\boldsymbol{len}_i(G) = l_i$ (for $i = 1, \ldots, n$);
- the *shape* of G is the tuple $\boldsymbol{shape}(G) = (l_1, \ldots, l_n)$;
- the *size* of G is the cardinality of its index set, *i.e.*,

$$\boldsymbol{size}(G) = \boldsymbol{card}(\boldsymbol{ind}(G)) = l_1 \cdot \ldots \cdot l_n;$$

- a *cell* of G is an *index* of G, *i.e.*, a tuple $\boldsymbol{i} = (i_1, \ldots, i_n) \in \boldsymbol{ind}(G)$;
- the *entry* in the cell $\boldsymbol{i}$ of G is (of course) $G_{\boldsymbol{i}}$;
- a k*-strip of* G (for $1 \leq k \leq n$) is a tuple of entries of G in dimension k of length $\boldsymbol{len}_k(G)$, *i.e.*, a tuple

$$(G_{\boldsymbol{i}\langle k/1\rangle}, \ldots, G_{\boldsymbol{i}\langle k/l_k\rangle})$$

  for some index $\boldsymbol{i}$ of G, where $l_k = \boldsymbol{len}_k(G)$;
- a p*-slice of* G (for $p = 1, \ldots, n$) is a p-dimensional subgrid, *i.e.*, a subgrid in which $n - p$ of the dimensions are held fixed; more precisely, for some fixed $1 \leq k_1 < \cdots < k_{n-p} \leq n$, and some fixed $j_1 \in \boldsymbol{seg}(l_{k_1}), \ldots, j_{n-p} \in \boldsymbol{seg}(l_{k_{n-p}})$, it is the indexed set S where

$$\boldsymbol{ind}(S) \;=\; \{(i_1, \ldots, i_n) \in \boldsymbol{ind}(G) \mid i_{k_1} = j_1, \ldots, i_{k_{n-p}} = j_{n-p}\}$$

  and for all $\boldsymbol{i} \in \boldsymbol{ind}(S)$, $S_{\boldsymbol{i}} = G_{\boldsymbol{i}}$; this is called the p-slice *across dimensions* $k_1, \ldots, k_{n-p}$ at *levels* $j_1, \ldots, j_{n-p}$ respectively, or the p-slice *with dimensions* $k_1, \ldots, k_{n-p}$ *fixed at levels* $j_1, \ldots, j_{n-p}$ respectively;
- a $(-q)$*-slice* of G is an $(n - q)$-slice (where $0 < q < n$), *i.e.*, a slice with q dimensions "missing"; thus a (-1)-slice is an $(n - 1)$-slice;
- more specifically, a $(-1, k)$*-slice* is a (-1)-slice with dimension k fixed, *i.e.*, a (-1)-slice "orthogonal to" the k-strips of G; hence a $(-1, k)$*-slice at level* j is a (-1)-slice with dimension k fixed at level j.

Definition 2. A *normal (function) table* $T \equiv (G, H^1, \ldots, H^n)$ $(n \geq 1)$ over Σ consists of a grid G whose entries are all Σ-terms of the same sort, and *coordinate headers* $H^1, \ldots, H^n$, where $n = \boldsymbol{dim}(G)$ and for $1 \leq i \leq n$, H^i is the header of T in dimension i, namely a tuple of Σ-*conditions* of length $\boldsymbol{len}_i(G)$. We also write, *e.g.*,

$$\begin{aligned}
&\boldsymbol{dim}(T) = \boldsymbol{dim}(G),\\
&\boldsymbol{len}_i(T) = \boldsymbol{len}_i(G) \text{ for } 1 \leq i \leq \boldsymbol{dim}(T),\\
&\text{a } k\textit{-strip of } T \text{ is a } k\text{-strip of } G,\\
&\text{a } \textit{slice} \text{ of } T \text{ is a slice of } G, \text{ and}\\
&T_{\boldsymbol{i}} \equiv G_{\boldsymbol{i}}, \text{ the } \textit{entry} \text{ in the cell } \boldsymbol{i} \text{ of } T \quad (\textit{i.e.}, \text{ cell } \boldsymbol{i} \text{ of } G).
\end{aligned}$$

The j-th entry of header H^k is said to *correspond to* or *label* the $(-1, k)$-slice of T at level j.

Note that for a 2-dimensional table T, *rows* are $(-1, 1)$-slices (which are the same as 2-strips), and *columns* are $(-1, 2)$-slices (which are the same as 1-strips); and H^1 and H^2 as the *row* and *column headers* respectively. Then, for an index (i, j) of T, $T_{(i,j)}$ (also written $T_{i,j}$) is the entry in *row* i and *column* j. This is consistent with the well-known convention for matrices.

2.3.3 Properness. We want a condition on tables which will make their semantics unproblematical.

Definition 1. Let C be a condition. A tuple $(C_1, \ldots, C_n)$ of conditions is called

- (a) *disjoint relative to* C if every state over $\boldsymbol{var}(C, C_1, \ldots, C_n)$ which satisfies C, satisfies *at most one of* $C_1, \ldots, C_n$.
- (b) *universal relative to* C if every state over $\boldsymbol{var}(C, C_1, \ldots, C_n)$ which satisfies C, satisfies *at least one of* $C_1, \ldots, C_n$.
- (c) *proper relative to* C if it is both disjoint and universal relative to C.

Equivalently, $(C_1, \ldots, C_n)$ is proper relative to C iff every state over $\boldsymbol{var}(C, C_1, \ldots$ $\ldots, C_n)$ which satisfies C, satisfies *precisely one of* $C_1, \ldots, C_n$.

An important special case of the above concepts is given by the following

Definition 2. A tuple $(C_1, \ldots, C_n)$ of conditions is called (respectively) *disjoint*, *universal* or *proper* if it is (respectively) disjoint, universal or proper relative to the condition true.

Equivalently, $(C_1, \ldots, C_n)$ is *proper* iff every state over $\boldsymbol{var}(C_1, \ldots, C_n)$ satisfies *precisely one of* $C_1, \ldots, C_n$. (This terminology is due to [Par92].)

Note that these concepts (disjointness, universality and properness) are all relative to the Σ-algebra A. For example, the tuple $(x < 0,\, x = 0,\, 0 < x)$ is not proper in all algebras (of the appropriate signature), but only in those algebras in which the interpretation of '$<$' satisfies the trichotomy law. On the other hand, the tuple $(x < 0,\, x \not< 0)$ is proper in all algebras (of the appropriate signature).

Definition 3. A table T is *proper* if all its headers are proper.

Remark. In a more refined semantic analysis, we would require for properness of T, not only the condition stated in Definition 3, but also (recursively) the properness of all tables used as tabular function symbols in the construction of T. The above definition is sufficient for our present purposes.

Table 2.1 is an example of a proper normal table. Note again that its properness is relative to the Σ-algebra A — in this case, the standard real algebra. We will usually not mention A explicitly.

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

H^2

$x \geq 0$
$x < 0$

H^1

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

G

TABLE 2.1. A proper normal table

2.3.4 Semantics of normal tables.

Suppose T is a proper normal table. By definition, all the entries in the grid of T have the *same sort*, say s.

Suppose $T \equiv (G, H^1, \ldots, H^n)$ with shape $(l_1, \ldots, l_n)$, index set $I = \boldsymbol{seg}(l_1) \times \cdots \times \boldsymbol{seg}(l_n)$, grid $G \equiv \langle t_{\boldsymbol{i}} \mid \boldsymbol{i} \in I \rangle$, where the $t_{\boldsymbol{i}}$ are terms of sort s, and headers $H^j \equiv (C^j_1, \ldots, C^j_{l_j})$, where the C^j_i are conditions.

For any state σ over T in A, the *meaning* $[\![T]\!]^A\sigma$ *of* T *relative to* σ (also written $[\![T]\!]\sigma$) is an element of A_s, given by the following *operational semantics*: For each header H^j, find the index i_j for which the entry $C^j_{i_j}$ holds at σ. Since T is proper, there is a unique such index for each header. These indices determine the cell $\boldsymbol{i} = (i_1, \ldots, i_n)$ of T, for which the entry $t_{\boldsymbol{i}}$ gives the required value $[\![T]\!]\sigma = [\![t_{\boldsymbol{i}}]\!]\sigma$.

We may, equivalently, give the semantics of T in the following way. We can think of T as representing a *term* incorporating (typically) a huge definition by cases at the top level, which can be written (in an informal but suggestive notation):

$$\boldsymbol{term}(T) \;\equiv\; \bigwedge_{i_1=1}^{l_1} \cdots \bigwedge_{i_n=1}^{l_n} \left(C^1_{i_1} \wedge \ldots \wedge C^n_{i_n} \;\rightarrow\; t_{i_1,\ldots,i_n}\right). \tag{2.3.1}$$

Note that properness of T is equivalent to the condition that for any state σ over T, there is precisely one index $(i_1, \ldots, i_n)$ for which $\sigma \models C^1_{i_1} \wedge \ldots \wedge C^n_{i_n}$. This determines the value of T at σ as

$$[\![T]\!]\sigma \;=\; [\![\boldsymbol{term}(T)]\!]\sigma \;=[\![t_{i_1,\ldots,i_n}]\!]\sigma.$$

The problems connected with the semantics of improper tables are indicated in [Zuc96]. In this paper we will not deal with this issue.

Next we will see how a table, relative to a list of variables, can define a *function.*

Definition 1. A list $\boldsymbol{x}$ of variables is said to *cover* T if $\boldsymbol{var}(T) \subseteq \boldsymbol{x}$.

Now let $\boldsymbol{x} = x_1, \ldots, x_m$ be any list of variables which covers T, with x_i of type s_i for $i = 1, \ldots, m$. Then relative to $\boldsymbol{x}$, T *names* or *defines* a *tabular function symbol*

$$f_{T,\boldsymbol{x}}:\; s_1 \times \cdots \times s_m \to s$$

with interpretation on A

$$f^A_{T,\boldsymbol{x}}:\; A_{s_1} \times \cdots \times A_{s_m} \to A_s,$$

as follows. For all $a_1 \in A_{s_1}, \ldots, a_m \in A_{s_m}$, let σ be the state over T defined by $\sigma(x_i) = a_i$ for $i = 1, \ldots, m$. Then

$$f^A_{T,\boldsymbol{x}}(a_1, \ldots, a_m) \;=\; [\![T]\!]\sigma.$$

We may also write, suggestively,

$$f_{T,\boldsymbol{x}} \;\equiv\; \lambda \boldsymbol{x} \cdot T.$$

Definition 2 (Semantic equivalence of tables). Let T_1 and T_2 be two proper normal tables. $T_1 \approx_A T_2$ (T_1 and T_2 are *semantically equivalent on* A) iff for all states σ over $\boldsymbol{var}(T_1, T_2)$ in A, $[\![T_1]\!]^A\sigma = [\![T_2]\!]^A\sigma$.

Note that semantic equivalence is defined here only as a relation between *proper tables.*

Note also that $T_1 \approx_A T_2$ iff for any list $\boldsymbol{x}$ of variables covering T_1 and T_2,

$$f^A_{T_1,\boldsymbol{x}} \;=\; f^A_{T_2,\boldsymbol{x}}\,.$$

Although semantic equivalence depends on the algebra A, we will usually drop the subscript 'A', and leave the dependence on A implicit.

The questions of the *decidability* (or *computability*) of properness of tables, and of semantic eqivalence between proper tables, are discussed in [Zuc96].

2.4 Inverted tables

In this section we consider the class $\boldsymbol{Tab}_I(\Sigma)$ of *inverted (function) tables over* Σ. Such a table T differs from a normal table in the following way:

(i) One of its coordinate headers, say H^k, is called the *value header* or *principal header*. It contains *terms*, all of the same sort, instead of conditions. The other headers are called *condition headers*, and contain conditions as before.

(ii) The cells of T contain *conditions* instead of terms.

The *value dimension* or *principal dimension* of T is the dimension of its value header. A *principal strip* of T is a k-strip, and a *principal slice* of T is a $(-1, k)$-slice, where k is the principal dimension.

The *operational semantics* for T is as follows. For a given state σ over T, search along the condition headers

$$H^1, \ldots, H^{k-1}, H^{k+1}, \ldots, H^n$$

until you find the index in each header for which the entry holds at σ. These indices determine a k-strip. Search along this strip for the cell whose entry has the value $\mathtt{tt}$. The corresponding entry in H^k then gives the value of the function.

The desirability of this search always producing a unique value, leads to the following definitions. Let T be an inverted table.

Definition 1. T is *proper* if

(i) all its condition headers are proper, and

(ii) every principal strip in T is proper *relative to the conjunction of the entries in the condition headers corresponding to that strip.*

Table 2.2 is an example of a proper inverted table. Here the value header is the row header H^1, so that the principal dimension is 1, and the tuples of cells in this dimension are the columns of the table. Notice that the columns are *not* proper, but *are* proper relative to the corresponding entries in the column header H^2.

Suppose $T \equiv (G, H^1, \ldots, H^n)$, with shape $(l_1, \ldots, l_n)$ and principal dimension k. So it has an index set $I = \boldsymbol{seg}(l_1) \times \cdots \times \boldsymbol{seg}(l_m)$, grid $G \equiv \langle C_{\boldsymbol{i}} \mid \boldsymbol{i} \in I\rangle$ with conditions $C_{\boldsymbol{i}}$, value header $H^k \equiv (t_1, \ldots, t_{l_k})$ with terms t_i, all of the same sort, and condition headers $H^j \equiv (C^j_1, \ldots, C^j_{l_j})$ $(j \neq k)$ with conditions C^j_i. Clause (ii) in Definition 1 says that for any index $\boldsymbol{i} = (i_1, \ldots, i_n)$, the k-strip $(C_{\boldsymbol{i}\langle k/1\rangle}, \ldots, C_{\boldsymbol{i}\langle k/l_k\rangle})$ is proper relative to

$$C^1_{i_1} \wedge \ldots \wedge C^{k-1}_{i_{k-1}} \wedge C^{k+1}_{i_{k+1}} \wedge \ldots \wedge C^n_{i_n}.$$

Proposition 1. *(With T as above, and k the principal dimension:) T is proper iff for any state σ over T, there is a unique index $(i_1, \ldots, i_n)$ of T such that*

(i) $\sigma \models C^j_{i_j}$ *for* $j = 1, \ldots, n$, $j \neq k$; *and*

$y \geq 0$	$y < 0$

H^2

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

H^1

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

G

TABLE 2.2. A proper inverted table

(ii) $\sigma \models C_{i_1,\ldots,i_n}$.

We get a stronger notion for inverted tables than properness, namely *strict properness*, if we strengthen clause (ii) in Definition 1:

Definition 2. T is *strictly proper* if

(i) all its condition headers are proper, and

(ii) every principal strip in T is proper.

Proposition 2. *(With T as above, k the principal dimension, and i_j ranging over* $\boldsymbol{seg}(l_j)$ *for $j = 1, \ldots, n$:) T is strictly proper iff for any state σ over T,*

(i) for $j = 1, \ldots, n$, $j \neq k$, there is a unique i_j such that $\sigma \models C^j_{i_j}$; and

(ii) for all $i_1, \ldots, i_{k-1}, i_{k+1}, \ldots, i_n$, there is a unique i_k such that $\sigma \models C_{i_1,\ldots,i_k,\ldots,i_n}$.

The more fundamental notion for our purposes is *properness*, since it is that which permits a total, deterministic (functional) semantics for inverted tables. However strict properness also features in our results (see Theorem 4.2(2)).

Assume now that T is proper. We may give its semantics equivalently in the following way. Assume also (for notational convenience) that the principal dimension is 1. Then T represents the following term:

$$\boldsymbol{term}(T) \;\equiv\; \bigwedge_{i_2=1}^{l_2} \cdots \bigwedge_{i_n=1}^{l_n} \Big(C^2_{i_2} \wedge \ldots \wedge C^n_{i_n} \;\rightarrow\; \bigwedge_{i_1=1}^{l_1} (C_{i_1 i_2 \ldots i_n} \;\rightarrow\; t_{i_1})\Big).$$

which can be rewritten (more simply, but less perspicuously in terms of the operational semantics given above) as

$$\bigwedge_{i_1=1}^{l_1} \bigwedge_{i_2=1}^{l_2} \cdots \bigwedge_{i_n=1}^{l_n} (C_{i_1 i_2 \ldots i_n} \wedge C^2_{i_2} \wedge \ldots \wedge C^n_{i_n} \;\rightarrow\; t_{i_1}). \tag{2.4.1}$$

By Proposition 1, properness of T is equivalent to the condition that for any state σ over T, there is a unique index $(i_1, \ldots, i_n)$ of T for which

$$\sigma \models C_{i_1 i_2 \ldots i_n} \wedge C^2_{i_2} \wedge \ldots \wedge C^n_{i_n}.$$

This determines the value of T at σ as

$$[\![T]\!]\sigma \;=\; [\![\boldsymbol{term}(T)]\!]\sigma \;= [\![t_{i_1}]\!]\sigma.$$

Again, relative to any list of variables $\boldsymbol{x}$ which covers T, this defines a function

$$f_{T,\boldsymbol{x}} \;\equiv\; \lambda \boldsymbol{x} \cdot T.$$

Semantic equivalence is defined for proper inverted tables just as for proper normal tables (§2.3.4 , Definition 2).

Note that in the formula for $\boldsymbol{term}(T)$ above, we assumed that the principal dimension was 1. Likewise, in the examples involving 2-dimensional inverted tables below, we will always assume that the principal dimension is 1; in other words, that the value header is the row header H^1. This is only for convenience; it is not part of the definition of inverted table.

3 Algorithms for transforming tables

We are interested in transforming tables to other, semantically equivalent tables, which may be easier to work with. We will define transformations

$$\varphi : \; \mathcal{C} \;\rightarrow\; \mathcal{C}'$$

of tables from one class $\mathcal{C}$ to another class $\mathcal{C}'$. These transformations must satisfy the following two properties:

(1) φ is *semantics preserving*, in the sense that (although in general not all tables in $\mathcal{C}$ are proper), if $T \in \mathcal{C}$ is proper, then so is $\phi(T)$, and $\phi(T) \approx T$.

(2) φ is *effective* or *computable.*

If $\phi(T) = T'$, then T' is the *transform* of T under φ. In this section we will consider four algorithms for transforming tables: one for lowering the dimensionality of a table (§3.1), one for transforming a normal to an inverted table (§3.2), and two for transforming an inverted to a normal table: the "1-dimensional" and "many-dimensional" algorithms (§3.3 and §3.4 resp.). The first three of these were considered in [Zuc96], and the fourth in [She95].

3.1 Changing the dimensionality; Flattening a table

Note first that any n-dimensional (normal or inverted) table can be trivially transformed to an $(n+1)$-dimensional table, by adding an $(n+1)$-th coordinate header with a single entry, '`true`'.

More interestingly: given a normal n-dimensional table, we can transform it to an $(n-1)$-dimensional table, by "combining" two of the dimensions, *i.e.*, combining two of the headers into a single header, using the enumeration function of §2.3.1.

More precisely, consider a normal table $T = (G, H^1, \ldots, H^n)$ $(n \geq 2)$ with shape $(l_1, \ldots, l_n)$, grid entries $G_{i_1,\ldots,i_n}$ and headers $H^j \equiv (C^j_1, \ldots, C^j_{l_j})$. This is transformed to a table $T' \equiv (G', H^1, \ldots, H^{n-2}, H'^{n-1})$ with shape $(l_1, \ldots, \ldots, l_{n-2},\ l'_{n-1})$, where $l'_{n-1} = l_{n-1} \cdot l_n$, new header $H'^{n-1} \equiv (C'^{n-1}_1, \ldots, C'^{n-1}_{l'_{n-1}})$, where

$$C'^{n-1}_{[i,j]} \ \equiv\ C^{n-1}_i \wedge C^n_j, \tag{3.1.1}$$

headers $H^1, \ldots, H^{n-2}$ unchanged, and grid entries

$$G'_{i_1,\ldots,i_{n-2},[i_{n-1},i_n]} \ \equiv\ G_{i_1,\ldots,i_{n-2},i_{n-1},i_n}. \tag{3.1.2}$$

Note that in the above description we combined the dimensions $n-1$ and n. This was just for notational convenience. We could combine any two dimensions of the table.

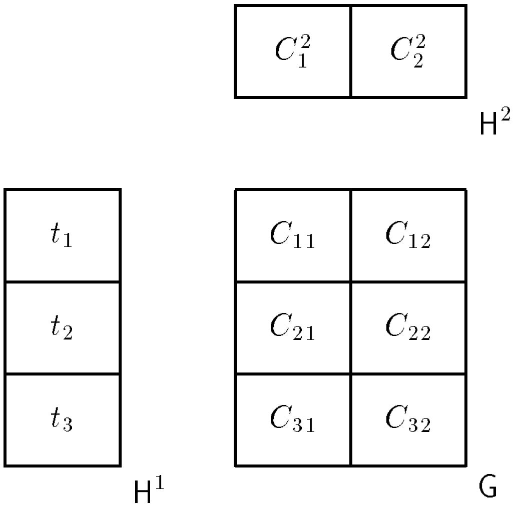
By iterating the above process, we can transform any normal table T to a 1-dimensional normal table of length ***size*** (T), which we call the *1-dimensional flattening* of T. This could be useful in the generation of program code from tables.

As an example of this, the 2-dimensional normal Table 2.1 can be flattened to the 1-dimensional Table 3.1.

H^1	G
$x \geq 0 \wedge y = 10$	0
$x \geq 0 \wedge y > 10$	y^2
$x \geq 0 \wedge y < 10$	$-y^2$
$x < 0 \wedge y = 10$	x
$x < 0 \wedge y > 10$	$x + y$
$x < 0 \wedge y < 10$	$x - y$

TABLE 3.1. 1-dimensional flattening of Table 2.1

Note next that the above transformation could also be applied to an *inverted* table of dimensionality > 2, by combining any two of its *non-principal* dimensions. By iterating this process, we can flatten any such inverted table down to 2 dimensions.



C^2_1	C^2_2

H^2

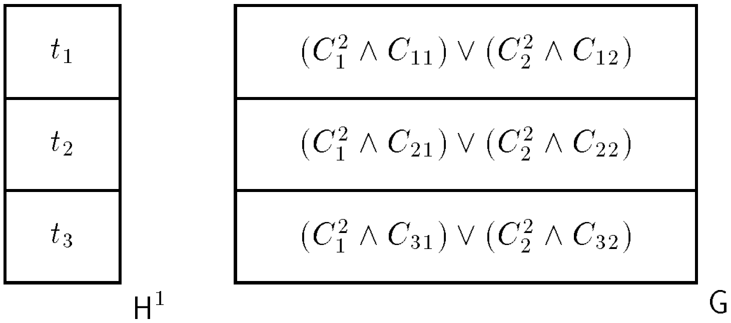
t_1
t_2
t_3

H^1

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

G

Table 3.2. 2-dimensional inverted table



t_1
t_2
t_3

H^1

$(C^2_1 \wedge C_{11}) \vee (C^2_2 \wedge C_{12})$
$(C^2_1 \wedge C_{21}) \vee (C^2_2 \wedge C_{22})$
$(C^2_1 \wedge C_{31}) \vee (C^2_2 \wedge C_{32})$

G

Table 3.3. 1-dimensional flattening of Table 3.2 (Short version)

Finally, we can transform a 2-dimensional inverted table down to 1 dimension, called the *1-dimensional flattening* of the original table, as follows. Consider (for example) a 2-dimensional inverted table, such as Table 3.2, with value header H_1.

This can be flattened to the 1-dimensional inverted table shown as Table 3.3.

More generally, a 2-dimensional inverted table $T = (G, H^1, H^2)$ with shape (l, m), value header $H^1 = (t_1, \ldots, t_l)$, condition header $H^2 = (C^2_1, \ldots, C^2_m)$, and grid cells C_{ij} $(1 \leq i \leq l, 1 \leq j \leq m)$, can be flattened into a 1-dimensional inverted table $T' = (G', H^1)$ with value header H^1 as before, and grid cells $C'_i (1 \leq i \leq l)$ where $C'_i = \bigvee_{j=1}^{m} (C^2_j \wedge C_{ij})$.

Note that the header H^1 of the transformed Table 3.3 still has length l. However the conditions in the grid are relatively complicated (large disjunctions). It may happen that these disjunctions can be simplified by inspection. Otherwise we can effect a tradeoff between complexity of conditions and header length by *"splitting disjunctions"*, as in Table 3.4. This increases the header length to $l \cdot m$. We call Tables 3.3 and 3.4 *short* and *long* flattenings of Table 3.2. (We will normally work with the short version, but our results are easily adaptable to the long version.)

Remarks. (1) Splitting disjunctions may not always be useful in simplifying conditions, since a disjunction $C_1 \vee C_2$ can often be simplified to a formula simpler than either C_1 or C_2, *e.g.*, if C_1 and C_2 are respectively of the form $D \wedge E$ and $D \wedge \neg E$.

(2) This transformation (from Table 3.2 to Table 3.3) will be considered again from a different perspective below (see the Remark in §3.3).

(3) Conversely, we can view the relationship between Tables 3.4 and 3.3 as the result of the elementary transformation of *combining* principal slices in Table 3.4 which have the same entries in the value header. (This will be considered more carefully in §3.2.)

t_1	$C_1^2 \wedge C_{11}$
t_1	$C_2^2 \wedge C_{12}$
t_2	$C_1^2 \wedge C_{21}$
t_2	$C_2^2 \wedge C_{22}$
t_3	$C_1^2 \wedge C_{31}$
t_3	$C_2^2 \wedge C_{32}$
H^1	G

TABLE 3.4. 1-dimensional flattening of Table 3.2 (Long version)

For all the flattening transformations considered above, the following holds.

Theorem 3.1. *Suppose T is proper, and T' is formed by flattening T. Then*

(1) T' is proper;

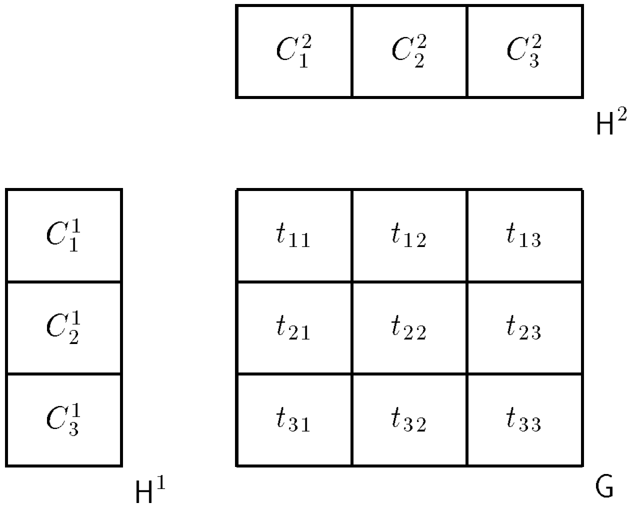
(2) $T' \approx T$.

Proof: This follows from the construction of T'. In the case where T is normal, or inverted of dimensionality > 2, parts (1) and (2) follow from formulas (3.1.1) and (3.1.2) respectively. □

3.2 Inverting a normal table

We first illustrate this with a simple example. Consider the case of a 2-dimensional 3×3 normal table, such as Table 3.5.

This table can be "inverted along dimension 1" (say), to produce Table 3.6.



C^2_1	C^2_2	C^2_3

H^2

C^1_1
C^1_2
C^1_3

H^1

t_{11}	t_{12}	t_{13}
t_{21}	t_{22}	t_{23}
t_{31}	t_{32}	t_{33}

G

Table 3.5. A normal table

We call this transformation the *standard version* of the inversion algorithm, or the *standard inversion* (in contrast to the compact version considered below). To derive the general formula for a table produced by such an inversion, consider a normal table $T \equiv (G, H^1, \ldots, H^n)$ with shape $(l_1, \ldots, l_n)$, grid entries $t_{i_1,\ldots,i_n}$ and headers $H^j \equiv (C^j_1, \ldots, C^j_{l_j})$. This is inverted along dimension 1 (say) to form a table $\tilde{T} \equiv (\tilde{G}, \tilde{H}^1, H^2, \ldots, H^n)$ with shape $(\tilde{l}_1, l_2, \ldots, l_n)$, where $\tilde{l}_1 = \textbf{\textit{size}}(T) = l_1 \cdot \ldots \cdot l_n$, value header $\tilde{H}^1 \equiv (\tilde{t}_1, \ldots \tilde{t}_{\tilde{l}_1})$, with $\tilde{t}_{[i_1,\ldots,i_n]} \equiv t_{i_1,\ldots,i_n}$, condition headers $H^2, \ldots, H^n$ unchanged from T, and grid entries $\tilde{C}_{j_1,j_2,\ldots,j_n}$, where

$$\tilde{C}_{[i_1,\ldots,i_n],j_2,\ldots,j_n} \equiv \begin{cases} C^1_{i_1} & \text{if } i_2 = j_2, \ldots, i_n = j_n \\ \mathsf{false} & \text{otherwise.} \end{cases} \tag{3.2.1}$$

Theorem 3.2. *Suppose T is a proper normal table, and $\tilde{T}$ is an inversion of T. Then*

(1) $\tilde{T}$ is strictly proper;

(2) $\tilde{T} \approx T$.

Proof: Part (1), which amounts to strict properness of the principal strips of $\tilde{T}$, follows from formula (3.2.1) and the properness of T. Part (2) follows from a comparison of ***term***(T) (formula (2.3.1)) and ***term***$(\tilde{T})$, which is constructed by applying formula (2.4.1) to the table $\tilde{T}$ as defined above. □

In general, a normal table can be inverted along any dimension k to produce an inverted table with value header $\tilde{H}^k$, and the other headers unchanged. The practical value of this transformation is, however, unclear, since the new table is much bigger than the original. (The length of the value header in the new table has increased to the size of the original table!)

It is, however, often possible to make the inverted table much more compact by performing the following *elementary transformation* on it. Suppose two or more entries

C_1^2	C_2^2	C_3^2

H^2

t_{11}
t_{21}
t_{31}
t_{12}
t_{22}
t_{32}
t_{13}
t_{23}
t_{33}

$\tilde{\mathsf{H}}^1$

C_1^1	false	false
C_2^1	false	false
C_3^1	false	false
false	C_1^1	false
false	C_2^1	false
false	C_3^1	false
false	false	C_1^1
false	false	C_2^1
false	false	C_3^1

$\tilde{\mathsf{G}}$

TABLE 3.6. Inversion of Table 3.5 (Standard version)

of the value header are identical. Then we *combine* the associated principal slices into one slice by forming pointwise disjunctions of corresponding cells. (An example is given by the transformation of Table 3.4 into Table 3.3; *cf.* Remark (3) of §3.1. The general concept of "elementary transformation" is discussed in Section 4.)

As an example, suppose the grid in Table 3.5 contains only 2 distinct terms, say $t_{11} \equiv t_{12} \equiv t_{22} \equiv t_{31} \equiv t_1$, and $t_{21} \equiv t_{32} \equiv t_{13} \equiv t_{23} \equiv t_{33} \equiv t_2$, as shown in Table 3.7.

Then some or all of the principal slices in the inverted Table 3.6 corresponding to a single term can be combined, to produce (in the extreme case) the inverted Table 3.8. Here the grid cell entries have been constructed by forming disjunctions of corresponding cells in Table 3.6, and replacing '$C \vee \mathsf{false}$' and '$\mathsf{false} \vee C$' by 'C'. We call this a *compact version* of the inversion algorithm or a *compact inversion.*

We prefer to think of these as two versions of the same algorithm, rather than two

	C^2_1	C^2_2	C^2_3
C^1_1	t_1	t_1	t_2
C^1_2	t_2	t_1	t_2
C^1_3	t_1	t_2	t_2

H^2

H^1 G

TABLE 3.7. A special case of Table 3.5

	C^2_1	C^2_2	C^2_3
t_1	$C^1_1 \vee C^1_3$	$C^1_1 \vee C^1_2$	false
t_2	C^1_2	C^1_3	$C^1_1 \vee C^1_2 \vee C^1_3$

H^2

$\tilde{\mathsf{H}}^1$ $\tilde{\mathsf{G}}$

TABLE 3.8. Inversion of Table 3.7 (Compact version)

distinct inversion algorithms, since the one is derivable from the other by an elementrary transformation (combining principal slices, as discussed above).

Theorem 3.2 applies to compact as well as standard inversions. The reason is that the transformation of combining principal slices preserves strict properness and semantic equivalence, as can easily be seen.

3.3 Normalising an inverted table: 1-dimensional algorithm

We now come to the converse problem, that of transforming an inverted table to a normal one. The first method we present is simple, but not entirely satisfactory, since the normal table it produces is 1-dimensional.

As a simple example, consider again the 2-dimensional 3 × 2 inverted table shown as Table 3.2, with value header H^1. This can be "normalised" to a 1-dimensional table, shown as Table 3.9

$(C_1^2 \wedge C_{11}) \vee (C_2^2 \wedge C_{12})$	t_1
$(C_1^2 \wedge C_{21}) \vee (C_2^2 \wedge C_{22})$	t_2
$(C_1^2 \wedge C_{31}) \vee (C_2^2 \wedge C_{32})$	t_3
$\hat{\mathsf{H}}^1$	$\hat{\mathsf{G}}$

Table 3.9. 1-D normalisation of Table 3.2 (Short version)

Note that the header $\hat{H}^1$ of the transformed Table 3.9 still has length $\boldsymbol{len}_1(T)$. However the conditions in this header are relatively complicated (large disjunctions). Again, we can again effect a tradeoff between complexity of conditions and header length by splitting disjunctions, as in Table 3.10. This will increases the header length to $\boldsymbol{size}(T)$. We call Tables 3.9 and 3.10 *short* and *long* versions of each other.

Remark. The (1-dimensional) normal Table 3.9 is essentially the same as the (1-dimensional) inverted Table 3.3, which was formed by another tranformation of the inverted Table 3.2, namely lowering its dimensionality to 1 (see Remark 2 in §3.1). These two 1-dimensional tables are transformed to each other simply by interchanging their row header and grid. Similarly, their respective long versions, namely Tables 3.4 and 3.10, are essentially the same.

$C_1^2 \wedge C_{11}$	t_1
$C_2^2 \wedge C_{12}$	t_1
$C_1^2 \wedge C_{21}$	t_2
$C_2^2 \wedge C_{22}$	t_2
$C_1^2 \wedge C_{31}$	t_3
$C_2^2 \wedge C_{32}$	t_3
$\hat{\mathsf{H}}^1$	$\hat{\mathsf{G}}$

Table 3.10. 1-D normalisation of Table 3.2 (Long version)

We call this transformation a *1-dimensional (*or *1-D) normalisation* (with *short* and *long* versions). The general formula for a table produced by this algorithm (using the short version) is derived as follows. Consider an inverted table $T \equiv (G, H^1, \ldots, H^n)$ with shape $(l_1, \ldots, l_n)$, grid entries $C_{i_1,\ldots,i_n}$, value header $H^1 \equiv (t_1, \ldots, t_{l_1})$, and condition headers $H^j \equiv (C^j_1, \ldots, C^j_{l_j})$ $(j = 2, \ldots, n)$. This is transformed to the 1-dimensional normal table $\hat{T} \equiv (\hat{G}, \hat{H}^1)$ with shape (l_1), grid entries $t_1, \ldots, t_{l_1}$, and header $\hat{H}^1 \equiv (\hat{C}_1, \ldots, \hat{C}_{l_1})$, where for $1 \leq i \leq l_1$

$$\hat{C}_i \;\equiv\; \bigvee_{i_2=1}^{l_2} \cdots \bigvee_{i_n=1}^{l_n} (C_{i,i_2,\ldots,i_n} \wedge C^2_{i_2} \wedge \ldots \wedge C^n_{i_n}). \tag{3.3.1}$$

For this transformation (short or long version) we have

Theorem 3.3. *Suppose T is a proper inverted table, and $\hat{T}$ is a 1-D normalisation of T. Then*

(1) $\hat{T}$ is proper;

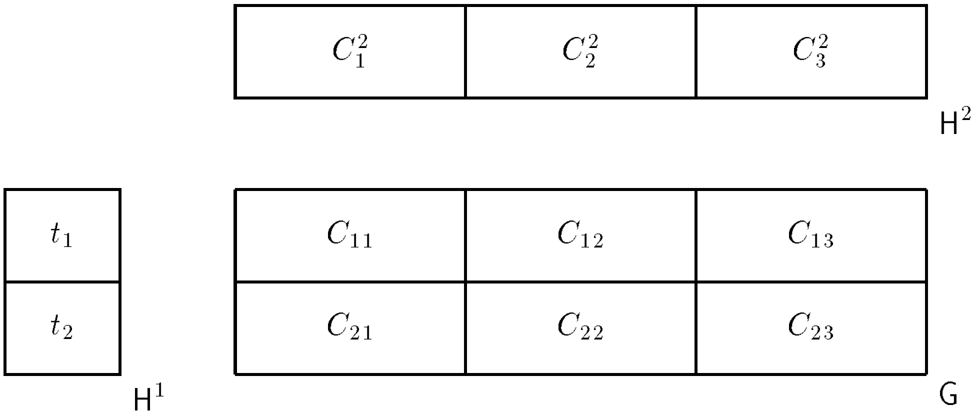
(2) $\hat{T} \approx T$.

Proof: Part (1), which amounts to the properness of the header $\hat{H}^1$, follows from (3.3.1) above and the properness of T. Part (2) (for the short version of normalisation) follows again from a comparison of $\boldsymbol{term}(T)$ (formula 2.4.1) and $\boldsymbol{term}(\hat{T})$, which is constructed by applying formula 2.3.1 to $\hat{T}$ as defined above. This proof can easily be adapted to the long version of normalisation. □

3.4 Normalising an inverted table: Multidimensional algorithm

We present a second, more complex, normalisation method for inverted tables, from [She95]. This method, unlike the first, preserves dimensionality. However it only applies to *strictly proper* inverted tables. (In practice, this is not a big drawback, since most inverted tables used in program documentation are strictly proper.)

As an example, consider the inverted Table 3.11, with value header H^1, which is assumed to be strictly proper. This is normalised to Table 3.12.



C^2_1	C^2_2	C^2_3

H^2

t_1
t_2

H^1

C_{11}	C_{12}	C_{13}
C_{21}	C_{22}	C_{23}

G

TABLE 3.11. An inverted table

C_1^2	C_2^2	C_3^2

H^2

$C_{11} \wedge C_{12} \wedge C_{13}$
$C_{11} \wedge C_{12} \wedge C_{23}$
$C_{11} \wedge C_{22} \wedge C_{13}$
$C_{11} \wedge C_{22} \wedge C_{23}$
$C_{21} \wedge C_{12} \wedge C_{13}$
$C_{21} \wedge C_{12} \wedge C_{23}$
$C_{21} \wedge C_{22} \wedge C_{13}$
$C_{21} \wedge C_{22} \wedge C_{23}$

$\hat{\mathsf{H}}^1$

t_1	t_1	t_1
t_1	t_1	t_2
t_1	t_2	t_1
t_1	t_2	t_2
t_2	t_1	t_1
t_2	t_1	t_2
t_2	t_2	t_1
t_2	t_2	t_2

$\hat{\mathsf{G}}$

TABLE 3.12. m-D normalisation of Table 3.11

We show that (a) Table 3.12 is proper, and (b) it is semantically equivalent to Table 3.11. For (b): consider, *e.g.*, the disjunction of the first 4 entries in $\hat{H}^1$, *i.e.*, all those conditions for which the corresponding term in column 1 is t_1:

$$\begin{aligned}
&(C_{11} \wedge C_{12} \wedge C_{13}) \vee (C_{11} \wedge C_{12} \wedge C_{23}) \vee (C_{11} \wedge C_{22} \wedge C_{13}) \vee (C_{11} \wedge C_{22} \wedge C_{23})\\
&= (C_{11} \wedge C_{12} \wedge (C_{13} \vee C_{23})) \;\vee\; (C_{11} \wedge C_{22} \wedge (C_{13} \vee C_{23}))\\
&= (C_{11} \wedge C_{12}) \vee (C_{11} \wedge C_{22})\\
&= C_{11} \wedge (C_{12} \vee C_{22})\\
&= C_{11}.
\end{aligned} \tag{3.4.1}$$

Here we use the fact that $(C_{13} \vee C_{23}) = \mathsf{true}$ and $(C_{12} \vee C_{22}) = \mathsf{true}$, by strict properness of the columns in Table 3.11. This shows that the condition under which the function value is t_1 when C_1^2 holds is exactly the same in both tables. Similarly for the other cases.

For (a), we must show properness of $\hat{H}^1$. Universality follows from (3.4.1) above and the fact that similarly the disjunction of the last 4 entries in $\hat{H}^1$ is equivalent to C_{21}, and $(C_{11} \vee C_{21}) = \mathsf{true}$, again by strict properness of Table 3.11. The disjointness property holds because, for a conjunction

$$(C_{i_1 1} \wedge C_{i_2 2} \wedge C_{i_3 3}) \ \wedge \ (C_{j_1 1} \wedge C_{j_2 2} \wedge C_{j_3 3}) \tag{3.4.2}$$

of conditions from any two different cells of $\hat{H}^1$, $i_k \neq j_k$ (for $k = 1$, 2 or 3) implies $C_{i_k k} \wedge C_{j_k k} = \mathsf{false}$, again by strict properness of the columns in Table 3.11, and so (3.4.2) evaluates to false.

We call this transformation *multi-dimensional (*or *m-D) normalisation.* The difference between this and 1-D normalisation can be summarised by saying that

> *1-D normalisation involves combining condition headers, and*
> *m-D normalisation involves refining the partitioning of the function domain.*

The general formula for a table produced by m-D normalisation is derived as follows. Consider again an inverted table $T \equiv (G, H^1, \ldots, H^n)$ with shape $(l_1, \ldots, l_n)$, grid entries $C_{i_1,\ldots,i_n}$, value header $H^1 \equiv (t_1, \ldots, t_{l_1})$, and condition headers $H^j \equiv (C^j_1, \ldots, C^j_{l_j})$ $(j = 2, \ldots, n)$. This is transformed to an n-dimensional normal table $\hat{T} \equiv (\hat{G}, \hat{H}^1, H^2, \ldots, H^n)$ with shape $(\hat{l}_1, l_2, \ldots, l_n)$, where

$$\hat{l}_1 \ = \ l_1^{l_2 \cdot l_3 \cdot \ldots \cdot l_n}. \tag{3.4.3}$$

In order to describe the entries in the new header $\hat{H}^1$ and grid $\hat{G}$, we first give some notation for certain index sets. Let

$$\begin{aligned} I &= \boldsymbol{seg}(l_1), \\ \bar{J} &= \boldsymbol{seg}(l_2) \times \cdots \times \boldsymbol{seg}(l_n), \\ K &= \boldsymbol{seg}(\hat{l}_1). \end{aligned}$$

Note that $I^{\bar{J}}$ (the set of functions from $\bar{J}$ to I) has cardinality

$$\boldsymbol{card}(I)^{\boldsymbol{card}(\bar{J})} \ = \ \hat{l}_1 \ = \ \boldsymbol{card}(K).$$

A typical element of $I^{\bar{J}}$ is itself a grid of indices

$$\langle i_{\bar{\jmath}} \mid \bar{\jmath} \in \bar{J} \rangle$$

indexed by $\bar{J}$, where for all $\bar{\jmath} = (i_2, \ldots, i_n) \in J$, $1 \leq i_{\bar{\jmath}} \leq l_1$. Now think of K as an "enumerated" version of $I^{\bar{J}}$ under the $\boldsymbol{enum}$ function, with the corresponding element $k \in K$ denoted by

$$k \ = \ [i_{\bar{\jmath}} \mid \bar{\jmath} \in \bar{J}] \ =_{df} \ \boldsymbol{enum}(\langle i_{\bar{\jmath}} \mid \bar{\jmath} \in \bar{J} \rangle).$$

Now the entries in the new header $\hat{H}^1$ are indexed by K; they are given by

$$\hat{C}^1_{[i_{\bar{\jmath}} | \bar{\jmath} \in \bar{J}]} \ \equiv \ \bigwedge_{\bar{\jmath} \in \bar{J}} C_{(i_{\bar{\jmath}}, \bar{\jmath})} \tag{3.4.4}$$

where for each index $[i_{\bar{\jmath}} \mid \bar{\jmath} \in \bar{J}]$, the rhs of (3.4.4) is a conjunction of a list of conditions of length $\boldsymbol{card}(\bar{J}) = l_2 \cdot \ldots \cdot l_n$, and for $\bar{\jmath} = (i_2, \ldots, i_n) \in \bar{J}$,

$$(i_{\bar{\jmath}}, \bar{\jmath}) \;=\; (i_{\bar{\jmath}}, i_2, \ldots, i_n) \in \boldsymbol{seg}(l_1) \times \boldsymbol{seg}(l_2) \times \cdots \times \boldsymbol{seg}(l_n).$$

Finally, the entries in the grid $\hat{G}$ are indexed by $K \times \bar{J}$; they are given by

$$\hat{t}_{[i_{\bar{\jmath}} \mid \bar{\jmath} \in \bar{J}], \bar{\jmath}'} \;\equiv\; t_{i_{\bar{\jmath}'}}.$$

Theorem 3.4. *Suppose T is a strictly proper inverted table, and $\hat{T}$ is an m-D normalisation of T. Then*

(1) $\hat{T}$ is proper;

(2) $\hat{T} \approx T$.

We have given the proof in the special case that T is Table 3.11 and $\hat{T}$ is Table 3.12. The proof in the general case can easily be constructed from this.

Notice that although the dimensionality is preserved by this method, the length of the changed header is increased considerably. (For convenience we will continue to assume that H^1 is the value header in the original table.) It turns out, however, that typically many of the entries in this new header evaluate to false, and the corresponding $(-1, 1)$-slices can be eliminated.

Since the conditions in the new header $\hat{H}_1$ are conjunctions of length $l_2 \cdot \ldots \cdot l_n$, we would prefer to combine this transformation with *evaluations* of these conditions, *i.e.*, finding simpler semantically eqivalent formulae, and, more especially, identify those conditions which evaluate to false, since the corresponding $(-1, 1)$-slices can then be deleted.

Note that there are two ways in which such a conjunction (3.4.4) can evaluate to false: either (a) one of the conjuncts, *i.e.*, a grid entry in the given table, is itself (equivalent to) false, or (b) a conjunction of some *sublist* of the list of conjuncts in the rhs of (3.4.4) is (equivalent to) false. Identification and elimination of either of these results in a considerable reduction in length for $\hat{H}^1$: for (a), by a factor of $1/l_1$, and for (b), by a factor of $1/l_1^k$, where k is the size of the sublist whose conjunction evaluates to false, since the conjunction of any list which includes a false sublist must itself be false. (In fact (a) is a special case of (b), with $k = 1$.)

There are two problems with this method, in connection with its complexity:

(i) The *mechanical evaluation* of conjunctions of conditions as in (3.4.4), especially the testing of equivalence to false. In general this is an undecidable problem, even for unquantified conditions (see [Zuc96], §10); but one can try to identify special cases in which it is decidable, and feasible.

(ii) The *large number* of such evaluations ($\hat{l}_1$: see (3.4.3)) of conjunctions for entries in $\hat{H}^1$. This number can be reduced by testing conjunctions of initial sublists of the lists of conditions on the rhs of (3.4.4) for equivalence to false (and generating such sublists in a suitable order), since then (as noted above) all conjunctions which extend such a false conjunction must themselves be false, and can be removed from further consideration.

Thus a solution for (*ii*) depends essentially on a solution for (*i*), *i.e.*, evaluation of conjunctions. This problem is a subject for ongoing research. It has not been solved satisfactorily yet, and so this method has not yet been implemented.

4 Interrelationship between transformations

We explore the relationships between some of the transformations and operations on tables considered in this paper. In §4.1 we find connections between inversion, 1-D normalisation and flattening. In §4.2 we consider the composition of an inversion and m-D normalisation. In both these cases, we will show that the composition of certain transformations on a given table is "elementarily equivalent" (in a sense to be made precise) to the table itself, or to some (other) transformation on it. We will use two notions of "elementary equivalence": a "syntactic" notion (in §4.1) and a "semantic" notion (in §4.2), both stricter than semantic equivalence. The difference between the two is discussed in §4.2. In §4.3 we show that the operations of *slicing* and *inverting* commute, as do the operations of *slicing* and *m-D normalising*.

4.1 Inversion, 1-D normalisation and 1-D flattening

Definition 1. An *elementary transformation* of a table is any one of the following operations:

(*a*) *structural transformation*:

(*i*) permuting two (-1)-slices together with their corresponding header entries;

(*ii*) deleting a (-1)-slice with 'false' in the corresponding header entry;

(*iii*) (in an inverted table) deleting a principal slice with only 'false' entries;

(*iv*) (in an inverted table) splitting a principal slice by "splitting a disjunction" in the corresponding header entry;

(*v*) (in an inverted table) combining two or more principal slices with the same value header entry into a single slice.

(*b*) *transformation of a condition cell entry*, as follows:

(*i*) permuting components of a conjunction or disjunction

(*ii*) distributing a conjunction over a disjunction,

(*iii*) simplifying conditions '$C \wedge \mathsf{false}$' to 'false',

(*iv*) simplifying conditions '$C \vee \mathsf{false}$' to 'C',

Note that in two dimensions the structural transformations (*a*) amount to permuting, deleting, repeating and combining rows or columns.

Definition 2. *Elementary equivalence* of tables is the equivalence relation on $\boldsymbol{Tab}_N$ and $\boldsymbol{Tab}_I$ generated by the class of elementary transformations. In other words, two tables are *elementarily equivalent* if one can be obtained from the other by a sequence of (0 or more) elementary transformations and/or their inverses.

We write $T_1 \approx_e T_2$ to denote that T_1 and T_2 are elementarily equivalent.

Remarks. (1) Elementary equivalence between tables is *decidable.*

(2) For *proper tables*, elementary transformations are easily seen to be semantics-preserving; hence elementary equivalence implies semantic equivalence (but not conversely).

(3) The long and short flattenings (§3.1) of a given table are elementarily equivalent to each other, as are the standard and compact inversions (§3.2) of a normal table, and the long and short 1-D normalisations (§3.3) of an inverted table.

(4) Warning: Properness (in particular, disjointness) is not necessarily preserved under elementary transformation $(a)(iv)$: splitting disjunctions. (This point was overlooked in [Zuc96] and [She95]. The relation of elementary equivalence must therefore be *restricted to pairs of proper tables.* In any case, properness is not violated in splitting disjunctions to transform short to long variants of the flattening and 1-D normalising algorithms, as used in Theorem 4.1 below.

Elementary equivalence between tables is preserved by all the transformations considered in Section 3. In other words:

Proposition. *If φ is any of the table transformations considered in Section 3, then for any two proper tables T_1 and T_2 in the domain of φ,*

$$T_1 \approx_e T_2 \Longrightarrow \varphi(T_1) \approx_e \varphi(T_2).$$

This is straightforward but tedious to check.

Now, given a normal table T, *inversion* followed by *1-D normalisation* is essentially the same as the 1-D flattening of T; and conversely, starting with an inverted table T, *1-D normalisation* followed by *inversion* is essentially the same as the 1-D flattening of T. This is made precise in the following theorem.

Theorem 4.1. *(1) If T is a proper normal table, then the result of an inversion of T followed by a 1-D normalisation is elementarily equivalent to a 1-D flattening of T.*

(2) If T is a proper inverted table, then the result of a 1-D normalisation followed by an inversion is elementarily equivalent to a 1-D flattening of T.

Note: The inversions in the statement of the theorem may be standard or compact, and the 1-D normalisations and flattenings may be short or long, by Remark 3 and the Proposition above. In the proof below, we assume for convenience that the inversions are standard and the normalisations and flattening are short.

Proof: These can be proved by examining the transformation formulas given in Sections 3.1—3.4. More specifically, for (1): given a proper normal table $T \equiv (G, H^1, \ldots, H^n)$

with shape $(l_1, \ldots, l_n)$, grid entries $t_{i_1,\ldots,i_n}$ and headers $H^j \equiv (C^j_1, \ldots, C^j_{l_j})$, the 1-D flattening of T is is the 1-dimensional normal table $\hat{T} = (\hat{G}, \hat{H})$ with shape $(\hat{l}_1)$, $\hat{l}_1 = l_1 \cdot \ldots \cdot l_n$, grid entries

$$\hat{t}_{[i_1,\ldots,i_n]} \;\equiv\; t_{i_1,\ldots,i_n}$$

and header entries

$$\hat{C}_{[i_1,\ldots,i_n]} \;\equiv\; C^1_{i_1} \wedge \ldots \wedge C^n_{i_n}.$$

It is easy to check that this is elementarily equivalent to the result of an inversion of T followed by a 1-D normalisation.

For (2): given an inverted table $T \equiv (G, H^1, \ldots, H^n)$ with shape $(l_1, \ldots, l_n)$, grid entries $C_{i_1,\ldots,i_n}$, value header $H^1 \equiv (t_1, \ldots, t_{l_1})$, and condition headers $H^j \equiv (C^j_1, \ldots, C^j_{l_j})$ $(j = 2, \ldots, n)$, the 1-D flattening of T is (elementarily equivalent to) the 1-dimensional inverted table $\tilde{T} \equiv (\tilde{G}, H^1)$ with shape (l_1), value header H^1 exactly as in T, and grid entries

$$\tilde{C}_i \;\equiv\; \bigvee_{i_2=1}^{l_2} \cdots \bigvee_{i_n=1}^{l_n} (C_{i,i_2,\ldots,i_n} \wedge C^2_{i_2} \wedge \ldots \wedge C^n_{i_n}).$$

Again, it is easy to check that this is also the result of a 1-D normalisation of T followed by an inversion (see Remark in Section 3.4.). □

Remark 5. Another reasonable candidate for a structural elementary transformation is:

(*vi*) removing a trivial dimension, *i.e.*, a dimension (and corresponding header) of length 1.

However this was not needed for the proof of Theorem 4.1.

4.2 Inversion and m-D normalisation

First we must modify the notion of elementary transformation and elementary equivalence used in §4.1:

Definition 1. An *elementary semantic transformation* of a table is *either*

(*a*) a *structural transformation* as in part (*a*) of Definition 1 in §4.1, *or*

(*b*) a *semantic transformation* of a condition cell entry, namely the replacement of a condition by a semantically equivalent condition.

Definition 2. *Elementary semantic equivalence* of tables is the equivalence relation on $\boldsymbol{Tab}_N$ and $\boldsymbol{Tab}_I$ generated by the class of elementary semantic transformations.

We write $T_1 \approx_{es} T_2$ to denote that T_1 and T_2 are elementarily sematically equivalent.

Remarks. (1) The semantic transformation (*b*) above incorporates all the transformations (*b*) listed in Definition 1 of §4.1, and much more. Thus elementary equivalence is a refinement of elementary semantic equivalence.

(2) Elementary semantic equivalence is, in turn, a refinement of semantic equivalence (for proper tables).

(3) Unlike elementary equivalence, elementary semantic equivalence is, in general, *undecidable.*

Again, elementary semantic equivalence between tables is preserved by all the transformations considered in Section 3. In other words:

Proposition. *If φ is any of the table transformations considered in Section 3, then for any two proper tables T_1 and T_2 in the domain of φ,*

$$T_1 \approx_{es} T_2 \implies \varphi(T_1) \approx_{es} \varphi(T_2).$$

Now, *inversion* composed with *m-D normalisation* (in either order) is essentially the identity transformation, in the following sense.

Theorem 4.2. *(1) If T is a proper normal table, then the result of an inversion of T followed by an m-D normalisation is elementarily semantically equivalent to T.*

(2) If T is a strictly proper inverted table, then the result of an m-D normalisation of T followed by an inversion is elementarily semantically equivalent to T.

Note: Again, the inversion in the statement of the Theorem may be standard or compact, by Remark 3 in §4.1 and the Proposition above. In the proof below, we assume for convenience that the inversions are standard.

Proof: (1) Given a normal table $T \equiv (G, H^1, \ldots, H^n)$ with shape $(l_1, \ldots, l_n)$, grid entries $t_{i_1,\ldots,i_n}$ and headers $H^j \equiv (C^j_1, \ldots, C^j_{l_j})$, let $\tilde{T}$ be the table formed by inverting T along dimension 1, so that (from §3.2) $\tilde{T} \equiv (\tilde{G}, \tilde{H}^1, H^2, \ldots, H^n)$ with value header $\tilde{H}^1 \equiv (\tilde{t}_1, \ldots \tilde{t}_{\tilde{l}_1})$, of length $\tilde{l}_1 = l_1 \cdot \ldots \cdot l_n$, where $\tilde{t}_{[i_1,\ldots,i_n]} \equiv t_{i_1,\ldots,i_n}$, and grid entries

$$\tilde{C}_{[i_1,\ldots,i_n],j_2,\ldots,j_n} \equiv \begin{cases} C^1_{i_1} & \text{if } i_2 = j_2, \ldots, i_n = j_n \\ \mathsf{false} & \text{otherwise.} \end{cases} \tag{4.2.1}$$

Let $\hat{\tilde{T}}$ be the m-D normalisation of $\tilde{T}$. Then (from §3.4) $\hat{\tilde{T}} \equiv (\hat{\tilde{G}}, \hat{\tilde{H}}^1, H^2, \ldots, H^n)$ where $\hat{\tilde{H}}^1$, of length $\hat{\tilde{l}}_1 = \tilde{l}_1^{l_2 \cdot l_3 \cdot \ldots \cdot l_n}$, has entries

$$\hat{\tilde{C}}^1_{[i_{\bar{\jmath}} | \bar{\jmath} \in J]} \equiv \bigwedge_{\bar{\jmath} \in J} \tilde{C}_{(i_{\bar{\jmath}}, \bar{\jmath})} \tag{4.2.2}$$

where $i_{\bar{\jmath}} \in I = \boldsymbol{seg}(\tilde{l}_1)$ and $\bar{\jmath} \in J = \boldsymbol{seg}(l_2) \times \cdots \times \boldsymbol{seg}(l_n)$. From (4.2.1)

$$\tilde{C}_{(i_{\bar{\jmath}}, \bar{\jmath})} \equiv \begin{cases} C^1_{i_1} & \text{if } i_{\bar{\jmath}} = [i_1, \bar{\jmath}] \text{ for some } i_1 \in \boldsymbol{seg}(l_1) \\ \mathsf{false} & \text{otherwise.} \end{cases} \tag{4.2.3}$$

Hence the conjunction in (4.2.2) is semantically equivalent to false if *either* (i) one of the components $\tilde{C}_{(i_{\bar{\jmath}}, \bar{\jmath})}$ is false, *or* (ii) two of the components have the form $C^1_{i_1}$ and

$C^1_{i_2}$ for $i_1 \neq i_2$, by properness of T (specifically, disjointness property of H^1). Thus the conjunction in (4.2.2) is *not* equivalent to false *only if* there exists $i_1 \in \boldsymbol{seg}(l_1)$ such that for all $\bar{\jmath} \in J$, $i_{\bar{\jmath}} \equiv [i_1, \bar{\jmath}]$, in which case (by (4.2.3)) it has the form

$$\begin{aligned} \hat{\tilde{C}}^1_{[i_{\bar{\jmath}} \mid \bar{\jmath} \in J]} \;&\equiv\; C^1_{i_1} \wedge C^1_{i_1} \wedge \ldots \wedge C^1_{i_1} \\ &\approx_{es} C^1_{i_1}. \end{aligned}$$

The corresponding grid entries (for $i_1 = 1, \ldots, l_1$) are

$$\hat{\tilde{t}}_{[i_{\bar{\jmath}} \mid \bar{\jmath} \in J], \bar{\jmath}'} \;\equiv\; \tilde{t}_{i_{\bar{\jmath}'}} \;\equiv\; t_{(i_1, \bar{\jmath}')}$$

i.e., the same as in T. Finally we can eliminate all other $(-1, 1)$-slices by

(i) a semantic transformation, replacing the corresponding header entry by 'false',

(ii) a structural transformation ($(a)(ii)$ in Definition 1 of §4.1).

We are left with the original normal table T.

(2) In the opposite direction: suppose given a strictly proper inverted table $T \equiv (G, H^1, \ldots, H^n)$ with shape $(l_1, \ldots, l_n)$, grid entries $C_{i_1, \ldots, i_n}$, value header $H^1 \equiv (t_1, \ldots, t_{l_1})$, and condition headers $H^j \equiv (C^j_1, \ldots, C^j_{l_j})$ $(j = 2, \ldots, n)$. This is transformed (as in §3.4) to an n-dimensional normal table $\hat{T} \equiv (\hat{G}, \hat{H}^1, H^2, \ldots, H^n)$ with shape $(\hat{l}_1, l_2, \ldots, l_n)$, where

$$\hat{l}_1 \;=\; l_1^{l_2 \cdot l_3 \cdot \ldots \cdot l_n}.$$

Writing again

$$\begin{aligned} I \;&=\; \boldsymbol{seg}(l_1), \\ \bar{J} \;&=\; \boldsymbol{seg}(l_2) \times \cdots \times \boldsymbol{seg}(l_n), \\ K \;&=\; \boldsymbol{seg}(\hat{l}_1), \quad \text{an enumeration of } I^{\bar{J}}, \end{aligned}$$

the entries of $\hat{H}^1$ (indexed by K) are given by

$$\hat{C}^1_{[i_{\bar{\jmath}} \mid \bar{\jmath} \in \bar{J}]} \;\equiv\; \bigwedge_{\bar{\jmath} \in \bar{J}} C_{(i_{\bar{\jmath}}, \bar{\jmath})}$$

and the entries in the grid $\hat{G}$ (indexed by $K \times \bar{J}$) are given by

$$\hat{t}_{[i_{\bar{\jmath}} \mid \bar{\jmath} \in \bar{J}], \bar{\jmath}'} \;\equiv\; t_{i_{\bar{\jmath}'}}.$$

Applying a standard inversion (as in §3.2) to $\hat{T}$ along dimension 1, we obtain the table $\tilde{\hat{T}} \equiv (\tilde{\hat{G}}, \tilde{\hat{H}}^1, H^2, \ldots, H^n)$ with shape $(\tilde{\hat{l}}_1, l_2, \ldots, l_n)$, where

$$\tilde{\hat{l}}_1 \;=\; \hat{l}_1 \cdot l_2 \cdot \ldots \cdot l_n \;=\; l_1^{l_2 \cdot \ldots \cdot l_n} \cdot l_2 \cdot \ldots \cdot l_n$$

and value header $\tilde{\hat{H}}^1 \equiv (\tilde{\hat{t}}_1, \ldots \tilde{\hat{t}}_{\tilde{\hat{l}}_1})$, with

$$\tilde{\hat{t}}_{[\hat{\imath}_1, i_2, \ldots, i_n]} \;\equiv\; \hat{t}_{\hat{\imath}_1, i_2, \ldots, i_n}, \tag{4.2.4}$$

where $1 \leq \hat{\imath}_1 \leq \hat{l}_1 = l_1^{l_2 \cdot \ldots \cdot l_n}$ and $1 \leq i_j \leq l_j$ $(j = 2, \ldots, n)$, and grid entries

$$\tilde{\hat{C}}_{[\hat{\imath}_1, i_2, \ldots, i_n], j_2, \ldots, j_n} \;\equiv\; \begin{cases} \hat{C}^1_{\hat{\imath}_1} & \text{if } \; i_2 = j_2, \ldots, i_n = j_n \\ \textsf{false} & \text{otherwise.} \end{cases} \tag{4.2.5}$$

Putting $\hat{\imath}_1 = [i_{\bar{\jmath}} \mid \bar{\jmath} \in \bar{J}]$, and $\bar{\jmath}' = (i_2, \ldots, i_n)$, we have, from (4.2.4):

$$\tilde{\hat{t}}_{[[i_{\bar{\jmath}} | \bar{\jmath} \in \bar{J}],\, \bar{\jmath}']} \;\equiv\; \hat{t}_{[i_{\bar{\jmath}} | \bar{\jmath} \in \bar{J}],\, \bar{\jmath}'} \;\equiv\; t_{i_{\bar{\jmath}'}} \tag{4.2.6}$$

and from (4.2.5):

$$\begin{aligned} \tilde{\hat{C}}_{[[i_{\bar{\jmath}} | \bar{\jmath} \in \bar{J}],\, \bar{\jmath}'],\, \bar{\jmath}''} \;&\equiv\; \begin{cases} \hat{C}^1_{[i_{\bar{\jmath}} | \bar{\jmath} \in \bar{J}]} & \text{if } \; \bar{\jmath}' = \bar{\jmath}'' \\ \textsf{false} & \text{otherwise} \end{cases} \\ &\equiv\; \begin{cases} \bigwedge_{\bar{\jmath} \in J} C_{(i_{\bar{\jmath}},\, \bar{\jmath})} & \text{if } \; \bar{\jmath}' = \bar{\jmath}'' \\ \textsf{false} & \text{otherwise} \end{cases} \end{aligned} \tag{4.2.7}$$

Now, for any fixed $i_1 \in I$, *combine* all principal slices in $\tilde{\hat{G}}$ with header entry t_{i_1} (structural transformation (v)). Then by (4.2.6) and (4.2.7), for a given $\bar{\jmath}' \in \bar{J}$, the entry in cell $(i_1, \bar{\jmath}')$ is the *disjunction* of $\textsf{false}$ and $\bigwedge_{\bar{\jmath} \in \bar{J}} C_{i_{\bar{\jmath}}, \bar{\jmath}}$ over all $[i_{\bar{\jmath}} \mid \bar{\jmath} \in \bar{J}] \in K$ for which $i_{\bar{\jmath}'} = i_1$, *i.e.*,

$$\bigvee_{\substack{[i_{\bar{\jmath}} | \bar{\jmath} \in \bar{J}] \in K \\ i_{\bar{\jmath}'} = i_1}} \left(\bigwedge_{\bar{\jmath} \in J} C_{i_{\bar{\jmath}}, \bar{\jmath}} \right)$$

which is equivalent (by pulling out the common conjunct $C_{i_1, \bar{\jmath}'}$ from all the disjuncts) to

$$C_{i_1, \bar{\jmath}'} \;\wedge\; \bigvee_{\substack{[i_{\bar{\jmath}} | \bar{\jmath} \in \bar{J}] \in K \\ i_{\bar{\jmath}'} = i_1}} \left(\bigwedge_{\bar{\jmath} \neq \bar{\jmath}'} C_{i_{\bar{\jmath}}, \bar{\jmath}} \right)$$

which is equivalent (by the reverse of distributing $\vee$ over $\wedge$) to

$$C_{i_1, \bar{\jmath}'} \;\wedge\; \bigwedge_{\bar{\jmath} \neq \bar{\jmath}'} \left(\bigvee_{i_1 \in I} C_{i_1, \bar{\jmath}} \right). \tag{4.2.8}$$

Now since T is strictly proper, the disjunctions $\bigvee_{i_1 \in I} C_{i_1, \bar{\jmath}}$ are all equivalent to $\textsf{true}$. Hence by a *semantic transformation* ((b) in Definition 1 of §4.1) we can replace (4.2.8) by $C_{i_1, \bar{\jmath}} \wedge \textsf{true}$, or just $C_{i_1, \bar{\jmath}}$, and so we recover the original table T. □

4.3 Slicing and inverting; Slicing and m-D normalising

Given a table T and a p-slice S of T, a p-dimensional table $\overline{T}$ is associated with S in an obvious way: first, a p-dimensional grid $\overline{G}$ is formed from S by an obvious modification of the index set, then the p headers of T from the unfixed dimensions of S are attached to $\overline{G}$ to form $\overline{T}$.

We use the same terminology for $\overline{T}$ as for the associated slice S; thus we say, *e.g.*, that $\overline{T}$ is the p-slice of T formed by fixing dimensions $k_1, \dots, k_{n-p}$ at $j_1, \dots, j_{n-p}$, or $\overline{T}$ is the $(-1, k)$-slice of T at level j, etc.

For the slicing of an inverted table to make sense, we must assume that it is not sliced across the principal dimension; in other words:

Assumption. *In slicing an inverted table, the principal dimension is not one of the fixed dimensions.*

It is easy to see that *properness is preserved* by slicing a *normal* table, but *not* (necessarily) by slicing an inverted table. For *inverted* tables, it is *strict properness* which is preserved:

Proposition. *(1) A slice of a proper normal table is proper.*

(2) A slice of a strictly proper inverted table is strictly proper.

Theorem 4.3. *(1) The operations of slicing and inverting a proper normal table commute up to elementary equivalence.*

(2) The operations of slicing and m-D normalising a strictly proper inverted table commute up to semantic elementary equivalence.

We will prove part (1). (The proof of part (2) is conceptually simple but notationally forbidding.)

Note (for part (2)) that one cannot speak of commuting slicing with 1-D normalisation, since 1-D normalisation does not preserve dimensionality.

Regarding part (1): note that since the inversion of a proper table is strictly proper by Theorem 3.2, and strict properness is preserved by slicing, the result of inverting and slicing (in any order) will be a strictly proper table.

Further, since any $(-q)$-slicing can be formed by q iterations of (-1)-slicing, it is sufficient to prove part (1) for (-1)-slicings. Thus Theorem 4.3(1) follows from

Lemma. *If T is a proper normal table, then the result of (-1)-slicing T followed by an inversion is elementarily equivalent to the result of an inversion of T followed by (-1)-slicing.*

(We are assuming here, of course, that both slicings are across the same dimension at the same level, and both inversions are along the same dimension, different from that of the slicing.)

Proof of Lemma: Let T be a proper normal table, say $T \equiv (G, H^1, \ldots, H^n)$, with shape $(l_1, \ldots, l_n)$, grid entries $t_{i_1,\ldots,i_n}$ and headers $H^j \equiv (C^j_1, \ldots, C^j_{l_j})$.

(i) Consider first *slicing* T, followed by *inverting*. For convenience, consider a (-1)-slice across dimension n, at level j_0. This produces a sliced table $\overline{T} \equiv (\overline{G}, H^1, \ldots, H^{n-1})$ with headers $H^1, \ldots, H^{n-1}$ as before, and grid entries $\bar{t}_{i_1,\ldots,i_{n-1}} \equiv t_{i_1,\ldots,i_{n-1},j_0}$.

Inverting $\overline{T}$ (by a standard inversion) along dimension 1 produces the table

$$\tilde{\overline{T}} \;\equiv\; (\tilde{\overline{G}},\ \tilde{\overline{H}}^1, H^2, \ldots, H^{n-1})$$

with value header $\tilde{\overline{H}}^1 \equiv (\tilde{\bar{t}}_1, \ldots, \tilde{\bar{t}}_{\tilde{\bar{l}}_1})$ of length

$$\tilde{\bar{l}}_1 \;=\; l_1 \cdot \ldots \cdot l_{n-1} \tag{4.3.1}$$

where

$$\begin{aligned} \tilde{\bar{t}}_{[i_1,\ldots,i_{n-1}]} \;&\equiv\; \bar{t}_{i_1,\ldots,i_{n-1}} \\ &\equiv\; t_{i_1,\ldots,i_{n-1},j_0} \end{aligned} \tag{4.3.2}$$

and grid entries

$$\tilde{\bar{C}}_{[i_1,\ldots,i_{n-1}],j_2,\ldots,j_{n-1}} \;\equiv \begin{cases} C^1_{i_1} & \text{if } \ i_2 = j_2, \ldots, i_{n-1} = j_{n-1} \\ \mathsf{false} & \text{otherwise.} \end{cases} \tag{4.3.3}$$

(ii) Consider now *inverting* T followed by *slicing*. Inverting T along dimension 1 produces the table $\tilde{T} \equiv (\tilde{G}, \tilde{H}^1, H^2, \ldots, H^n)$ with value header $\tilde{H}^1 \equiv (\tilde{t}_1, \ldots, \tilde{t}_{\tilde{l}_1})$, of length $\tilde{l}_1 = l_1 \cdot \ldots \cdot l_n$, where

$$\tilde{t}_{[i_1,\ldots,i_n]} \equiv t_{i_1,\ldots,i_n}, \tag{4.3.4}$$

and grid entries

$$\tilde{C}_{[i_1,\ldots,i_n],j_2,\ldots,j_n} \;\equiv \begin{cases} C^1_{i_1} & \text{if } \ i_2 = j_2, \ldots, i_n = j_n \\ \mathsf{false} & \text{otherwise.} \end{cases}$$

Slicing $\tilde{T}$ across dimension n at level j_0 produces the table

$$\bar{\tilde{T}} \;\equiv\; (\bar{\tilde{G}},\ \tilde{H}^1, H^2, \ldots, H^{n-1})$$

with value header $\tilde{H}^1$ as in $\tilde{T}$, of length

$$\tilde{l}_1 = l_1 \cdot \ldots \cdot l_n, \tag{4.3.5}$$

and entries as in (4.3.4), and grid entries

$$\begin{aligned}\bar{\tilde{C}}_{[i_1,\ldots,i_n],j_2,\ldots,j_{n-1}} &\equiv \tilde{C}_{[i_1,\ldots,i_n],j_2,\ldots,j_{n-1},j_0} \\ &\equiv \begin{cases} C^1_{i_1} & \text{if } \; i_2 = j_2, \ldots, i_{n-1} = j_{n-1} \; \text{ and } \; i_n = j_0 \\ \mathsf{false} & \text{otherwise.} \end{cases}\end{aligned} \tag{4.3.6}$$

From (4.3.1) and (4.3.5), it follows that the principal dimension of $\bar{\tilde{T}}$ is l_n times as long as that of $\tilde{\overline{T}}$. However, it can be seen that most of the principal slices of $\bar{\tilde{T}}$ consist of 'false' only, and so can be deleted, resulting in elementary equivalent tables. More precisely: for $i_n \neq j_0$, it follows from (4.3.6) that the principal slices of $\bar{\tilde{T}}$ at levels $[i_1, \ldots, i_{n-1}, i_n]$ contains only 'false' grid entries, and so may be deleted as an *elementary transformation*. The remaining slices of $\bar{\tilde{T}}$, at levels $[i_1, \ldots, i_{n-1}, j_0]$, have the same grid and value header entries as the slices of $\tilde{\overline{T}}$ at levels $[i_1, \ldots, i_{n-1}]$, as can be seen by comparing (4.3.2) with (4.3.4), and (4.3.3) with (4.3.6). □

5 Table Transformation Tool

To facilitate the use of tabular notation, the Table Tool System (TTS) described in [SERG97] has been developed at the Software Engineering Research Group (SERG), McMaster University. The goal of the TTS project is to develop an integrated, extensible system of tools — that is, a set of tools that work together, so that a designer or programmer can manipulate tables easily in computer systems documentation.

As one of the tools in the TTS, the Table Transformation Tool implemented the table transformation algorithms proposed in Section 3. The TTS provides two modules: the Table Holder, which hides the representation of the data structures used to implement the Σ-algebra, and the Information Module, which hides the representation of the syntax.

By using these two modules, the implementation of the transformation algorithms was greately simplified, since we need concern ourselves only with the structure of the algorithms, rather than the specific representation of the data and syntax.

General steps of the algorithm implementations include: (1) checking whether the given expression is a so called "well-formed" table, i.e., syntactically correct for the algorithm; (2) constructing the shape of the transformed table, each grid and the length along each dimension of the table; (3) copying the unchanged headers if any; (4) calculating the changed headers; and (5) calculating the main grid entries.

The code is written follow the programming guidelines proposed in [PMI94]. A set of *displays*, which specify the relations the programs should satisfy, is provided along with the code. The display method facilitates integration of the table transformation algorithms described in this paper with the TTS to form the Table Transformation Tool.

6 Conclusions; Future work

Since the work done in [She95] some improvements have been made by the Software Engineering Research Group at McMaster University (SERG) to the Table Transformation Tool (TTT) as part of the Table Tool System (TTS); much remains to be done.

6.1 Recent work on the TTT

The following improvements have been made on the TTT since the publication of [She95]:

(1) A graphic user interface has been developed for the TTT, consistent with the TTS interface.

(2) A utility to facilitate the interpretation of tables has been developed [Abr97].

6.2 Ideas for future work

Some of the following ideas are being planned in the Software Engineering Research Group; others are more speculative at this stage. An asterisk indicates that a start has already been made on that topic (typically as a Master's project).

(1)* *Simplification* of output tables (in any format) is clearly important. We refer here both to simplifying the global structure if the table, and the terms and formulas in the cells. This can best be done interactively, in conjunction with a symbolic computation system such as Maple. A start in this direction has been made in [Ras98].

(2)* It is important to develop tools for (partial) testing of *properness* of tables, at least in special cases, since this problem is not decidable or feasible for many common signatures [Zuc96].

(3)* Similarly, it is important to develop (partial) tests for *semantic equivalence* of tables, or of terms occurring within tables.

(4) An important special case of (3) is the evaluation of *conditions*, *i.e.*, testing whether they are equivalent to true or false. This is useful in two ways: (*a*) to test for properness as in (2); (*b*) to implement the m-D normalisation algorithm (§3.4).

(5)* Continuing the development in §6.1(2), a utility should be developed for the *type checking* of input tables.

(6) The main reason for table transformations is to simplify tables (in some sense). *Heuristic guidelines* should therefore be developed for deciding which version of a given table is likely to be simplest.

(7)* Finally, tools should be developed for transformations from, and to, other kinds of tables. Some work in this direction is currently being done by SERG on "decision tables".

References

[Abr97] R.F. Abraham. Evaluating generalized tabular expressions in software documentation. M.Eng. Thesis, Department of Electrical & Computer Engineering, McMaster University, 1997. CRL Report 346, Communications Research Laboratory, McMaster University.

[AFB$^+$92] T.A. Alspaugh, S.R. Faulk, K.H. Britton, R.A. Parker, D.L. Parnas, and J.E. Shore. Software requirements for the A-7E Aircraft. NRL Report NRL/FR/5530-92-9194, U.S. Naval Research Laboratory, Washington, DC, 1992.

[Hen80] K.L. Heninger. Specifying software requirements for complex systems: New techniques and their application. *IEEE Transactions on Software Engineering*, SE-6:2–13, 1980.

[HKPS78] K.L. Heninger, J. Kallander, D.L. Parnas, and J.E. Shore. Software requirements for the A-7E Aircraft. NRL Memorandum Report 3876, U.S. Naval Research Laboratory, Washington, DC, 1978.

[PAM91] D.L. Parnas, G.J.K. Asmis, and J. Madey. Assessment of safety-critical software in nuclear power plants. *Nuclear Safety*, 32:189–198, 1991.

[Par92] D.L. Parnas. Tabular representation of relations. CRL Report 260, Communications Research Laboratory, McMaster University, 1992.

[Par94] D.L. Parnas. Inspection of safety-critical software using program-function tables. In *Proceedings of the IFIP World Congress, August 1994, Volume III*, pages 270–277, 1994.

[PMI94] D.L. Parnas, J. Madey, and M. Iglewski. Formal documentation of well-structured programs. *IEEE Transactions on Software Engineering*, 20:948–976, 1994.

[Ras98] P. Rastogi. Specialization: An approach to simplifying tables in software documentation. M.eng. thesis, Department of Electrical & Computer Engineering, McMaster University, 1998. CRL Report 360, Communications Research Laboratory, McMaster University.

[(SE97] McMaster University Software Engineering Research Group (SERG). Table tool system developer's guide. CRL Reports 339 & 340, Communications Research Laboratory, McMaster University, 1997.

[She95] H. Shen. Implementation of table inversion algorithms. M.Eng. Thesis, Department of Electrical & Computer Engineering, McMaster University, 1995. CRL Report 315, Communications Research Laboratory, McMaster University.

[Zuc96] J.I. Zucker. Transformations of normal and inverted function tables. *Formal Aspects of Computing*, 8:679–705, 1996.