Table Transformation Tools: Why and How[1]

H. Shen[2], J. Zucker[3], D.L. Parnas[4]

Abstract

This paper describes a prototype tool for inverting tabular representations of mathematical functions, providing the motivation for its construction, some discussion of its use, and a sketch of its implementation.

1 Introduction

This paper describes a prototype tool for inverting tabular representations of mathematical functions, providing the motivation for its construction, some discussion of its use, and a sketch of its implementation. Table transformation tools of the type discussed here are intended to be useful in the documentation of complex computer systems.

1.1 Functional documentation of computer systems

The importance of having precise documentation for software products is clear to all experienced software developers. Hundreds of "standards" have been proposed; each was intended to improve the consistency, precision and completeness of natural language documents. In spite of these efforts, software documentation in all industries is still inadequate. Because of the vagueness and imprecision of natural languages, even the best software documentation is unclear. Because informal documentation cannot be analysed systematically, it is usually inconsistent and incomplete as well.

Software Engineering, like other forms of Engineering, must make use of mathematics. Mathematical notation is commonly used in engineering documents. It is only through the use of mathematics that we can obtain the precision that is required. In Computer Science, there has also been a great deal of discussion about the use of mathematics to verify the correctness of software. Before program verification can become practical, the use of mathematics in documentation must be well-established. Specifications and design documents state the theorems that should be proven about programs.

Although there is agreement about the importance of documentation, software developers have not reached agreement on which documents are needed or on the contents of various documents. It is quite common to hear developers arguing about whether or not a certain fact should have been included in some document. Often information is included in several documents or not found in any. The first step in improving the quality of documentation is to agree on definitions of the contents of key documents.

In [8] Parnas and Madey proposed that definitions of the contents of a number of common documents can be given by stating that each document must contain a representation of one or more binary relations (sets of ordered pairs). If the document contains enough information to determine whether or not any pair is included in the specified relation, it is complete. Because most of the relations involved are actually functions, we call such documentation "functional documentation".

1.2 Use of tabular notation to describe functions and relations

The idea that each document is a representation of a set of relations is important and powerful but it is not enough. If we use conventional mathematical notation, describing these relations requires expressions that are too complex and hard to parse to be useful in practice. As a consequence of this complexity, mathematical specifications are often incorrect, and, even more often, misunderstood. This has led many people to conclude that it is not practical to provide precise mathematical descriptions of computer systems.

In several practical applications (e.g. [1, 4, 7, 15]), however, we learned to use multidimensional expressions, which we call *tables*. Each table is an indexed set of cells; each cell contains other expressions, possibly tabular ones.

[1] Research funded by a Strategic Grant from the Natural Sciences and Engineering Research Council of Canada and by the Telecommunications Research Institute of Ontario.

[2] CRL, McMaster University, Hamilton, ON, L8S 4K1, Canada. Email: `hshen@mcserg.crl.mcmaster.ca`

[3] Computer Science Dept, McMaster University, Hamilton, ON, L8S 4K1, Canada. Email: `zucker@mcmaster.ca`

[4] CRL, McMaster University, Hamilton, ON, L8S 4K1, Canada.

With experience, we have learned to use a variety of forms of such tables. The most useful forms were described in [9]. Ryszard Janicki provided a more general model of tables in [5]. Other organizations and groups are beginning to find these tables useful. A survey and discussion can be found in [16].

1.3 Why have more than one kind of table?

When writing the document described in [3], we began by trying to use the types of tables described in standard logic design textbooks. We were successful with our first example, but the second example involved so many different variables of different types that the resulting table was too large to be useful and took far too much space for the amount of information it contained. This led us to discover what we now call normal tables [9]. However, a few days later, working on another function description, we again found ourselves with a table that was much too large. In this case we found that by "inverting" this table (using the header to carry information that had been in the main grid of the table and placing information from that header in the main grid) we could obtain a compact representation. In short, we need many types of tables because we want compact, easily used representations and the functions we must describe have such widely varying characteristics that a table form that is suitable for one may not be suitable for another.

1.4 Why would we need to transform a table?

The process of designing software and documenting that design must be understood as an exploratory process. For example, when trying to describe the requirements for a system, we do not usually know what they are when we start – documenting requirements is a process of inquiry.

Often the best way to proceed is to begin by identifying the variables (inputs) that determine the value of the outputs. Once we have done this, we can identify the domain of the mathematical functions that we are going to describe. We can also partition the set of values of the variables in a way that all values within a partition will be treated by the same computational rules. We then prepare a table in which the headers partition the domain of the function/relation so that all possible combinations of the values of individual variables are represented. Generally this will be a very large table, but one that enables us to consider all of the possible cases in turn.

The exploratory process then requires "filling in" the table by finding out what to put in each of the entries. With requirements analysis this may be done by asking potential users, by examining the characteristics of the environment or by looking at earlier systems. When documenting existing code, the answers are found by examining that code. When the table is complete, we know what the function is but often have a poor representation of it. Looking at the results of our exploration, we often see that if we had known at the start what we know at the end of the documentation process, we could have picked a better form of table. When working on [1] we had to transform tables by hand to get them down to a reasonable size and a usable format. This was tedious and exacting work and the support of a tool such as that described in [14] would have been very helpful.

The next section contains an example that illustrates two different types of tables and where a transformation between them was helpful.

2 An example from A–7E requirement document [1]

The following example shows how two different tables can be used to express the same function.

2.1 The first description of the function

The example is one of the software functions which specify the software requirements for the A–7E Aircraft [1]. 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[5] and conditions.

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 [13], 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.

2.2 A more compact inverted table

The same function can 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 2. When the A–7E requirements were originally written in 1977, the compact form of Table 2 was prepared manually from Table 1. [1, p368]

[5] A mode corresponds to a subset of the set of possible system states of the Operational Flight Program. A mode class is a set of modes which partitions the system state space. For a given mode class, transitions may occur between its modes.

	M1	M2	M3	M4	M5	M6
$c1 \wedge c2 \wedge c3 \wedge c4$	none	!target!	!target!	!Fly–to point!	!target!	!target!
$c1 \wedge c2 \wedge c3 \wedge \neg c4$	none	!target!	!target!	!Fly–to point!	!target!	!target!
$c1 \wedge c2 \wedge \neg c3 \wedge c4$	none	!target!	!target!	!Fly–to point!	!target!	!target!
$c1 \wedge c2 \wedge \neg c3 \wedge \neg c4$	none	!target!	!target!	!target!	!target!	!target!
$c1 \wedge \neg c2 \wedge c3 \wedge c4$	none	none	!Fly–to point!	!Fly–to point!	none	!Fly–to point!
$c1 \wedge \neg c2 \wedge c3 \wedge \neg c4$	none	none	!Fly–to point!	!Fly–to point!	none	!OAP!
$c1 \wedge \neg c2 \wedge \neg c3 \wedge c4$	none	none	!OAP!	!Fly–to point!	none	!Fly–to point!
$c1 \wedge \neg c2 \wedge \neg c3 \wedge \neg c4$	none	none	!OAP!	!target!	none	!OAP!
$\neg c1 \wedge c2 \wedge c3 \wedge c4$	!Fly–to point!	!target!	!target!	!Fly–to point!	!target!	!target!
$\neg c1 \wedge c2 \wedge c3 \wedge \neg c4$	!Fly–to point!	!target!	!target!	!Fly–to point!	!target!	!target!
$\neg c1 \wedge c2 \wedge \neg c3 \wedge c4$	!Fly–to point!	!target!	!target!	!Fly–to point!	!target!	!target!
$\neg c1 \wedge c2 \wedge \neg c3 \wedge \neg c4$	!Fly–to point!	!target!	!target!	!target!	!target!	!target!
$\neg c1 \wedge \neg c2 \wedge c3 \wedge c4$	!Fly–to point!	!Fly–to point!	!Fly–to point!	!Fly–to point!	none	!Fly–to point!
$\neg c1 \wedge \neg c2 \wedge c3 \wedge \neg c4$	!Fly–to point!	!Fly–to point!	!Fly–to point!	!Fly–to point!	none	!OAP!
$\neg c1 \wedge \neg c2 \wedge \neg c3 \wedge c4$	!Fly–to point!	!Fly–to point!	!OAP!	!Fly–to point!	none	!Fly–to point!
$\neg c1 \wedge \neg c2 \wedge \neg c3 \wedge \neg c4$	!Fly–to point!	!Fly–to point!	!OAP!	!target!	none	!OAP!

c1: /FLYTOTW/=0
c2: !Desig!
c3: !Before slewing!
c4: !Range!>30

M1: All alignment and nav modes, none of the mode following: *Walleye* *A/G Guns*
M2: *Nattack*, *HUDdown1*, *Snattack*, *SHUDdown1*
M3: *Noffset*, *HUDdown2*, *Snoffset*, *SHUDdown2*
M4: *BOC*, *SBOC*
M5: *BOCFlyto0*, *SBOCFlyto0*
M6: *BOCoffset*, *SBOCoffset*

Table 1 Normal Table for the HSI Reference Points in SCR Notation

	none	!target!	!Fly–to point!	!OAP!
All alignment and nav modes, none of the modes below: *Walleye* *A/G Guns*	/FLYTOTW/=0	**false**	¬ /FLYTOTW/=0	**false**
Nattack *HUDdown1* *Snattack* *SHUDdown1*	¬ !Desig! ∧ /FLYTOTW/=0	!Desig!	¬ !Desig! ∧ ¬ /FLYTOTW/=0	**false**
Noffset *HUDdown2* *Snoffset* *SHUDdown2*	**false**	!Desig!	¬ !Desig! ∧ !Before slewing!	¬ !Desig! ∧ ¬ !Before slewing!
BOC *SBOC*	**false**	¬ !Before slewing! ∧ ¬ !Range!>30	!Before slewing! ∨ !Range!>30	**false**
BOCFlyto0 *SBOCFlyto0*	¬ !Desig!	!Desig!	**false**	**false**
BOCoffset *SBOCoffset*	**false**	!Desig!	!Range!>30 ∨ ¬ !Desig!	¬ !Range!>30 ∨ ¬ !Desig!

Table 2 Inverted Table semantically equivalent to Table 1

3 Realization of table transformations mechanically

3.1 Algorithms for transformations

The importance of table transformations has been discussed in §1.4. In this section we will survey five transformations: one for *decreasing the dimensionality* of a table, two for *inverting* a normal table, and two for *normalizing* an inverted table[6]. The first four were studied in [17], and the fifth in [14].

All the tables considered below are assumed to be *proper*, which means that their condition headers (C_1, ..., C_n) satisfy the properties of *disjointness* ($C_i \wedge C_j$ = **false** for $i \neq j$) and *universality* ($C_1 \vee \ldots \vee C_n$ = **true**).

Also, the rows of conditions in the main grid of the inverted table (see Table 8) are assumed to satisfy a similar condition. An exact formulation of these conditions is given in [17, 14], where it is also shown that transformations of proper tables preserves their semantics.

3.1.1 Decreasing the dimensionality of a table

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.

By iterating this procedure, we can transform any normal table to a 1–dimensional normal table — albeit very long (with length equal to the number of cells in the original table). This could be useful in the generation of programs from tables.

3.1.2 Inverting a normal table: Method 1

We illustrate the method with a simple example. Consider the case of a 2–dimensional 3×3 normal table, as in Table 3. (The header entries C^i_j are conditions, and the grid entries t_{ij} are terms.) This can be "inverted along dimension 2" (say), to produce Table 4.

In general, a normal table can be inverted along any dimension *k* to produce an inverted table with value header

[6] In a *normal* table, the conditions are contained in the headers and the output terms are in the main grid. In an *inverted* table, the output terms are contained in one of the headers (the "value header") and the conditions are in the other headers and the main grids. Table 1 and Table 2 are examples of normal and inverted tables respectively.

$\tilde{H}^k$, and the other headers unchanged. Note that the new table is much bigger than the original. The length of the value header in the new table has increased to the number of cells in the original table!

3.1.3 Inverting a normal table: Method 2

We present a second method for inversion, which leads to better (*i.e.* smaller) inverted tables when there are not many distinct terms in the main grid. Here we use one element of the new header for each *distinct* term in the main grids. Suppose, *e.g.*, Table 3 contains only 2 distinct terms in the main grid, say t_1 and t_2, as shown in Table 5.

We invert Table 5 along dimension 2 (say) as in Table 6.

In this way we get a smaller table, since the length of $\tilde{H}^2$ in the new table has shrunk to the number of distinct terms in the grid of Table 3. However the conditions in the grid of the inverted table can be quite large disjunctions. These disjunctions can perhaps be simplified by inspection. Otherwise we can make a *tradeoff* between length of value header and complexity of these conditions, by "splitting disjunctions" as in Table 6 so as to get Table 7. (This is not always as useful as it may seem, 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 $D \wedge \neg E$)

Note that we could simplify Table 7 by deleting columns that contain only **false** entries. Note also that this inversion is different from that given by Method 1 (*cf.* Table 4).

3.1.4 Normalizing an inverted table: Method 1

This algorithm is simple but not quite satisfactory: it transforms an inverted table to a 1–dimensional normal one. As a simple example, consider the 2–dimensional 2×3 inverted table Table 8, with value header H^2. This can be normalized "along dimension 2" to a 1–dimensional table, as shown in Table 9.

Note that the header $\hat{H}^2$ of Table 9 has the same length as H^2. However the conditions in this header are quite complicated. We can, however, effect a tradeoff between complexity of conditions and header length by "splitting" disjunctions, as in Table 10.

3.1.5 Normalizing an inverted table: Method 2

This algorithm, although more complex than Method 1 of normalizing, has the advantage of preserving dimensionality [14]. The idea of the method is that by re–partitioning the function domain, we can form a new header in the transformed normal table, which corresponds to the value header in the original inverted table. Table 8 and Table 11 illustrate how the algorithm works. (Table 8 has been normalized along dimension 2.)

H^1 \ H^2 (G)	C_1^2	C_2^2	C_3^2
C_1^1	t_{11}	t_{12}	t_{13}
C_2^1	t_{21}	t_{22}	t_{23}
C_3^1	t_{31}	t_{32}	t_{33}

Table 3 A Normal Table

H^1 \ $\tilde{H}^2$ ($\tilde{G}$)	t_{11}	t_{12}	t_{13}	t_{21}	t_{22}	t_{23}	t_{31}	t_{32}	t_{33}
C_1^1	C_1^2	C_2^2	C_3^2	**false**	**false**	**false**	**false**	**false**	**false**
C_2^1	**false**	**false**	**false**	C_1^2	C_2^2	C_3^2	**false**	**false**	**false**
C_3^1	**false**	**false**	**false**	**false**	**false**	**false**	C_1^2	C_2^2	C_3^2

Table 4 Inversion of Table 3: Method 1

	C_1^2	C_2^2	C_3^2
C_1^1	t_1	t_2	t_1
C_2^1	t_1	t_1	t_2
C_3^1	t_2	t_2	t_2

H^2 H^1 G

Table 5 A Special Case of Table 3

	t_1	t_2
C_1^1	$C_1^2 \vee C_3^2$	C_2^2
C_2^1	$C_1^2 \vee C_2^2$	C_3^2
C_3^1	**false**	$C_1^2 \vee C_2^2 \vee C_3^2$

$\tilde{H}^2$ H^1 $\tilde{G}$

Table 6 Inversion of Table 5 : Method 2

	t_1	t_1	t_1	t_2	t_2	t_2
C_1^1	C_1^2	C_3^2	**false**	C_2^2	**false**	**false**
C_2^1	C_1^2	C_2^2	**false**	C_3^2	**false**	**false**
C_3^1	**false**	**false**	**false**	C_1^2	C_2^2	C_3^2

$\tilde{H}^2$ H^1 $\tilde{G}$

Table 7 Inversion of Table 5: Variant of Method 2

	t_1	t_2	t_3
C_1^1	C_{11}	C_{12}	C_{13}
C_2^1	C_{21}	C_{22}	C_{23}

H^2 H^1 G

Table 8 A 2–dimensional Inverted Table

$(C_1^1 \wedge C_{11}) \vee (C_2^1 \wedge C_{21})$	$(C_1^1 \wedge C_{12}) \vee (C_2^1 \wedge C_{22})$	$(C_1^1 \wedge C_{13}) \vee (C_2^1 \wedge C_{23})$	$\hat{H}^2$
t_1	t_2	t_3	$\hat{G}$

Table 9 Normalization of Table 8: Method 1

$(C_1^1 \wedge C_{11})$	$(C_2^1 \wedge C_{21})$	$(C_1^1 \wedge C_{12})$	$(C_2^1 \wedge C_{22})$	$(C_1^1 \wedge C_{13})$	$(C_2^1 \wedge C_{23})$	$\hat{H}^2$
t_1	t_1	t_2	t_2	t_3	t_3	$\hat{G}$

Table 10 Normalization of Table 8: Variant of Method 1

H^1	$C_{11}\wedge C_{21}$	$C_{11}\wedge C_{22}$	$C_{11}\wedge C_{23}$	$C_{12}\wedge C_{21}$	$C_{12}\wedge C_{22}$	$C_{12}\wedge C_{23}$	$C_{13}\wedge C_{21}$	$C_{13}\wedge C_{22}$	$C_{13}\wedge C_{23}$	$\tilde{H}^2$
C_1^1	t_1	t_1	t_1	t_2	t_2	t_2	t_3	t_3	t_3	
C_2^1	t_1	t_2	t_3	t_1	t_2	t_3	t_1	t_2	t_3	$\tilde{G}$

Table 11 Normalization of Table 8: Method 2

3.2 Implementation of table inversion

As a sub–project of the Table Tool Project [10] at McMaster University Software Engineering Research Group, the implementation of table inversion algorithms is based on the Table Holder module [6]. This hides the internal (storage) format of tabular expressions and simplifies the implementation, since we need concern ourselves only with the realization of the algorithms instead of with both the algorithms and the complicated data structures for expressions.

The prototype of the Table Inversion Tool provides users with an interface to select tables, display the selected table or a slice of it if the dimension of the table is higher than 2, edit header alias names for further references, apply transformations to the selected table and save the transformed tables. The main menu of the user interface looks like this:

```
1. Load Table
2. Alias Header Name
3. Show a Slice of the Table
4. Apply a Transformation Algorithm
5. Save Table
6. Exit

Select
```

After applying the algorithm in §3.1.3 to Table 1, we get an inverted table with the same headers as Table 2, but the conditions on the main grids are much more complicated. For example, the condition in the cell (1,1) is

(/FLYTOTW/=0 $\wedge$!Desig! $\wedge$!Before slewing! $\wedge$!Range!>30) $\vee$

(/FLYTOTW/=0 $\wedge$!Desig! $\wedge$!Before slewing! $\wedge$ $\neg$!Range!>30) $\vee$

(/FLYTOTW/=0 $\wedge$!Desig! $\wedge$ $\neg$!Before slewing! $\wedge$!Range!>30) $\vee$

(/FLYTOTW/=0 $\wedge$!Desig! $\wedge$ $\neg$!Before slewing! $\wedge$ $\neg$!Range!>30) $\vee$

(/FLYTOTW/=0 $\wedge$ $\neg$!Desig! $\wedge$!Before slewing! $\wedge$!Range!>30) $\vee$

(/FLYTOTW/=0 $\wedge$ $\neg$!Desig! $\wedge$!Before slewing! $\wedge$ $\neg$!Range!>30) $\vee$

(/FLYTOTW/=0 $\wedge$ $\neg$!Desig! $\wedge$ $\neg$!Before slewing! $\wedge$!Range!>30) $\vee$

(/FLYTOTW/=0 $\wedge$ $\neg$!Desig! $\wedge$ $\neg$!Before slewing! $\wedge$ $\neg$!Range!>30)

After simplifying this formula by De Morgan's law and $C \vee \neg C$=***True***, we can reduce this to */*FLYTOTW*/=0*, which is the same as cell (1,1) in Table 2. When the conditions on all main grids are simplified, we get an inverted table exactly the same as Table 2. To make the Table Inversion Tool of more useful, we need a tool to simplify such formulas mechanically.

3.3 A relation between table transformations

We have shown that by applying the algorithm in §3.1.3 to Table 1 and then simplifying the conditions in the main grids, we obtain Table 2. Conversely, we can recover Table 1 from Table 2 by applying the algorithm in §3.1.5. This suggests that the composition of the algorithms in §3.1.3 and §3.1.5 together with some simplifications of conditions produces the identity transformation. A proof of this can be found in [14].

4 Conclusions

4.1 How a table transformation tool would be used in industry

The software industry wants better documentation, but does not feel that it can afford the time and expense for methods that require extensive careful mathematical work. It also wants the documentation to be compact. The tools that we are developing can reduce the cost/benefit ratio by making it easier to produce more accurate compact documentation. Other tools reduce this ratio by increasing the benefit [11,12].

4.2 Future work arisen from table inversion:

In this paper we have illustrated the usefulness of transformations from normal to inverted tables, and conversely. Other transformations should also be mechanized, *e.g.* it is sometimes useful, especially with large tables, to raise the table's dimensionality, not in the trivial way considered in [17], but so as to produce a series of cross–sections which are simpler to comprehend. Also, if we find that several tables have the same headers, we may want to combine them into a single "vector function table" or "mixed vector table" (see [9]). Each of these tasks can be done without mechanical support, but good tools would make it possible to perform the task much quicker and eliminate errors.

From §3.2, we see that simplification is necessary to make the Table Inversion Tool more useful. Such simplification may be performed automatically by use of term rewriting techniques[2]. Purely automated simplification is however, unlikely to be adequate for all applications. Future tools must allow manual intervention.

Another problem concerning table inversion worth further investigation is term comparison. If we examine the algorithm in §3.1.3, we find that we have to decide whether two terms in the main grid of a normal table (such as $x+y$ and $y+x$) are equivalent. It is not always simple to compare such terms, since for many structures, equality between terms is undecidable. A tool to assist such decisions would be very helpful. It could incorporate packages such as Mathematica or Maple.

References

[1] 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, August, 1992.

[2] R. Boyer and J. Moore, "A computational Logical Handbook", Academic Press, Boston, 1988.

[3] K.L. Heninger, "Specifying Software Requirements for Complex Systems: New Techniques and their Application", *IEEE Transactions Software Engineering*, Vol. SE-6, No. 1, January 1980, pp. 2-13.

[4] S.D. Hester, D.L. Parnas, D.F. Utter, "Using Documentation as a Software Design Medium", *Bell System Technical Journal*, Vol. 60, No. 8, October 1981, pp. 1941-1977.

[5] R. Janicki, “Towards a Formal Semantics of Parnas Tables”, *Proceedings of the 17th International Conference on Software Engineering*, Seattle WA, 1995, pp. 231-240.

[6] C. Krasnor, and D.L. Parnas, “The Table Tool System: The Table Holder”, *CRL Report No. 300*, McMaster University, TRIO (Telecommunications Research Institute of Ontario), 1995. *in preparation*

[7] D.L. Parnas, G.J.K. Asmis, J. Madey, “Assessment of Safety-Critical Software in Nuclear Power Plants”, *Nuclear Safety*, Vol. 32, No. 2, 1991, pp. 189-198.

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

[9] D.L. Parnas, “Tabular Representation of Relations”, *CRL Report 260*, McMaster University, Communications Research Laboratory, TRIO (Telecommunications Research Institute of Ontario), October 1992, 17 pgs.

[10] D.L. Parnas, et al, “The Table–Tool Project Design Issues”, *Technical Report*, CRL, McMaster University, Hamilton, Ontario, Canada; *in preparation*.

[11] Peters, D., “Generating a Test Oracle from Program Documentation”, *M.Eng. Thesis*, Department of Electrical and Computer Engineering, McMaster University, Hamilton, ON, Canada, April 1995, 97 pgs. *CRL Report No. 302*, McMaster University, Communications Research Laboratory, TRIO (Telecommunications Research Institute of Ontario), April 1995.

[12] D. Peters, D. L. Parnas, “Generating a Test Oracle from Program Documentation”, *Proceedings of the 1994 International Symposium on Software Testing and Analysis (ISSTA),* August 17-19, 1994, pp. 58 - 65.

[13] M. Phister, “Logical Design of digital computer”, John Wiley & Sons, Inc. June, 1968.

[14] H. Shen, “Implementation of Table Inversion Algorithms”, *M. Eng. Thesis*, Department of Electrical and Computer Engineering, McMaster University, October 1995, 96 pgs. *CRL Report No. 315*, McMaster University, Communications Research Laboratory, TRIO (Telecommunications Research Institute of Ontario), December 1995.

[15] A.J. van Schouwen, D.L. Parnas, J. Madey, “Documentation of Requirements for Computer Systems”, Proceedings of *’93 IEEE International Symposium on Requirements Engineering*, San Diego, CA, 4 – 6 January, 1993, pp. 198 – 207.

[16] A.J. Wilder, J.V. Tucker, “System Documentation Using Tables – a short course”, CRL Report 306, McMaster University, Communications Research Laboratory, TRIO (Telecommunications Research Institute of Ontario), May 1992, 110 pgs. Also published as Report CSR 11–95, Computer Science Department, University of Wales, Swansea, 1995.

[17] J. I. Zucker, “Transformation of Normal and Inverted Function Tables”, *Formal Aspects of Computing*, To appear.