HKUST Theoretical Computer Science Center Research Report HKUST-TCSC-98-05

A Conceptual Model for Tables*

Xinxin Wang[1] and Derick Wood[2]

[1] AT&T Labs, 200 Laurel Avenue, Middletown, NJ 07748, USA.
xinxin@puma.att.com.

[2] Department of Computer Science, Hong Kong University of Science & Technology, Clear Water Bay, Kowloon, Hong Kong.
dwood@cs.ust.hk.
WWW: http://www.cs.ust.hk/~dwood.

Abstract. We describe a new, simple conceptual model for tables. The conceptual model treats a table as a map that has a domain which is a product of categories or index sets and a codomain which is a set of entry values. We demonstrate how we can use the model to specify the semantics of some tabular editing operations.

1 Introduction

Tables have been and are primarily a presentational technique, but with the surge of use of the Internet and computers, we expect that tables will be used even more. Moreover, they will be used in new ways. For example, tables are already being produced that are much too large to be displayed on a single page or in a single window. In this scenario, we need to have a sound conceptual model of tables to provide a foundation for the design of tabular browsers, the production of tabular views, and the design of tabular query systems.

The ideas we discuss are presented in a more rudimentary form in Wang's thesis [21] and in even more rudimentary form in an early paper by Wang and Wood [22]. Vanoirbeek [19, 20] appears to have been the first researcher to identify the multidimensional nature of tables and the hierarchical structure of rubrics, which we call categories or index sets depending on the context. Tables present a challenging problem for structured-document modelers as they do not fit comfortably in a hierarchical model as Furuta [10] observed in his thesis. Models that are more presentation oriented have been discussed by others [2, 3, 5, 7, 16, 17, 15] as well by the SGML community [12, 13, 14]. We could, of course, use a more complex apparatus for modeling tables, such as type theory or algebraic specifications, but our stance and philosophy is to develop small models using simple notions, rather than base models on more complex notions. This approach is similar to the "small languages" approach of Jon Bentley, Brian

* This paper will appear in the Principles of Digital Document Processing (PODDP 98) to be published in the Springer-Verlag Lecture Notes in Computer Science series. This work was supported under a grant from the Research Grants Council of Hong Kong.

Kernighan and others [4], and to the “KIS” philosophy of system and program design. Brooks [6] explores what may and will happen when systems are large and are complex.

The **content** of a table is a collection of interrelated items that may be numbers, text, symbols, figures, mathematical equations, or even other tables. There are two kinds of items: the basic data displayed by a table, the **entries,** and the auxiliary data used to locate the entries, the **labels.** For example, Table 1 presents the average marks for the assignments and examinations of a course offered in the three terms of 1991 and 1992. The marks are the entries (for example, 75 and 80) and the strings that denote the years, the terms, and the kinds of marks are the labels (for example, 1991, Winter, and Midterm). We cannot always determine what are labels and what are entries so easily; however, we shall assume we can always do so and that they are disjoint from each other.

Table 1. The average marks for 1991–1992.

	Assignments			Examinations		Grade
	Ass1	Ass2	Ass3	Midterm	Final	
1991						
Winter	85	80		60	75	75
Spring	80	65	75	60	70	70
Fall	80	85		55	80	75
1992						
Winter	85	80	70		75	75
Spring	80	80			75	75
Fall	75	70	65	60	80	70

A table is divided into four main regions by **stub separation** and **boxhead separation.** The **stub** is the lower left region that contains the row headings, the **boxhead** is the upper right region that contains the column headings, the **stub head** is the upper left region that contains the index sets in the stub, and the **body** is the region to the right of the stub and below the boxhead that contains the entries.

We present a conceptual model for tables that is based on simple mathematical notions. It is presentation independent (it does not depend on any specific display of a table), representation independent (we do not provide any specific representation of the model), and it is system independent (it is isolated from any tabular management system). As an application of the model, we demonstrate how it can be used to specify the semantics of some tabular editing operations. a tabular editor. We consider our tabular model to be a first step in addressing the problem: “What is a table?” It bears a similar relationship to tables that a context-free grammar model has to programming languages—there is much left unsaid. Despite this frugality, the model provides a firm basis for further work.

2 Some preliminary remarks

We present a conceptual model for tables that captures the underlying syntactic relationships of a table. The conceptual model treats a table as a map that has an unordered Cartesian product of categories or index sets as its domain and some universe of entries as its codomain. The index sets are restricted partial orders; indeed, they are trees. The number of index sets determines the dimension of the table and, as with programming language arrays, each entry in a d-dimensional table is determined by d indices. The method of indexing tables is what makes tables different from arrays and spreadsheets. We base the tabular model on a formalism for categories and index sets that is appropriate for tables. Our position is that the manipulation of categories is the primary aspect of tabular manipulation whether it is for editing, querying, or formatting. In contrast, index sets for arrays and spreadsheets do not use and do not require such a rich repertoire of index-set operations.

One way we use a table such as Table 1 is that we have a specific year, term, and kind of mark in mind and we want to retrieve the corresponding mark. For example, the mark corresponding to 1992, Spring, Examinations, and Final is 75. We use the labels to index a unique entry. Observe that although the label Spring occurs twice in the table, we use only one of its appearances; namely, the one within 1992. There is a hierarchical arrangement of years and terms in this table; in general, tabular labels are arranged hierarchically. We use dot notation to indicate the hierarchical dependence by writing 1992·Spring as the stub index (Dewey decimal classification schemes, email addresses, and C++ structures use a similar notation). Similarly, Midterm and Final both depend on Examinations, so we write Examinations·Final as the box-head index. Dependent labels define **indices** and sets of indices form **index sets**, which are hierarchical. Table 1 has six stub indices and six box-head indices, so the table has 36 entries (although some identical entries are presented only once) and its **size** is, therefore, 36.

Presentationally, a table is two-dimensional; hence, two indices are necessary and sufficient to determine a unique entry. Conceptually, however, Table 1 may be viewed as a three-dimensional table since Year and Term may be treated as separate index sets. (Observe that we cannot break Marks into two or more index sets.) In this case, we need three indices to determine an entry uniquely just as we do with a three-dimensional array. Observe that there are two Year, three Term, and six Marks indices, so the conceptual table still has size $2 \times 3 \times 6 = 36$. As a reader, we use a Cartesian product (even when we do not think of it this way) of the three index sets

$$Year = \{1991, 1992\},$$
$$Term = \{Winter, Spring, Fall\},$$

and

$$Marks = \{Assignments \cdot Ass1, Assignments \cdot Ass2, Assignments \cdot Ass3,$$

$$Examinations \cdot Midterm, Examinations \cdot Final, Grade\}$$

to specify the entries. There are six possible products, the one in Table 1 is the product $Year \times Term \times Marks$. The other five different products are $Year \times Marks \times Term$, $Term \times Year \times Marks$, $Term \times Marks \times Year$, $Marks \times Term \times Year$, and $Marks \times Year \times Term$. The different products correspond to the same conceptual table presented in different ways. In addition, for each product of the three indices, we can assign the index sets to the stub and boxhead in four different ways. For example, the partition given in Table 1 is $Year \times Term \| \times Marks$, where we use '$\|$' to specify the partition (this notation is similar to, but different from, the use of parentheses suggested by Darrell Raymond [18]). We could use one of the other three partitions $\|Year \times Term \times Marks$, $Year\| \times Term \times Marks$, or $Year \times Term \times Marks\|$.

An array is a random-access data structure for efficient storage and retrieval. For example, the array $R[0..1, 0..2, 0..3]$ of real numbers is three-dimensional and it has $2 \times 3 \times 4 = 24$ elements whose values are real numbers. The array R can be modeled as a map from a Cartesian product of the three index sets to the reals.

Tables and arrays (and spreadsheets) have, not surprisingly, some similarities: Both can be multidimensional and both are indexed by products of index sets. But at this point the similarities end and we find only differences.

Product order: Each Cartesian product of the index sets of an array determines a different array. For example, for the array R, we have the product order $0..1 \times 0..2 \times 0..3$. If we change the product order to $0..2 \times 0..1 \times 0..3$, say, it is a different array.
For tables, each different product order determines the same table with a different presentation.

Index sets: The ordering of the indices in an index set is total for arrays, but for tables it may be a total order, a partial order, a preorder, or no order at all. In addition, array index sets are not only a total order but also they are a subrange of a total order (usually the set of integers). For example, for the array R, $0..3$ is a subrange of the integers, but the index set $\{Winter, Spring, Fall\}$ is not a subrange of the understood total order $\{Winter, Spring, Summer, Fall\}$. The index set for Marks is a partial order but is not a total order as $Midterm$ has no predefined relationship with assignments $Ass1$, $Ass2$, and $Ass3$.

Use: Arrays are a data structure for the efficient storage and retrieval of similar kinds of values. They provide random access to the stored values. Tables, on the other hand, are a structure for the presentation and effective retrieval of data; presentation is their primary use, whereas presentation is at most a secondary use for arrays.

The manipulation of tables is, therefore, necessarily more complex than is the manipulation of arrays. We want to be able to change the product ordering, the partition of the index sets between the stub and the box head, and the ordering within index sets to achieve the purpose of a specific table—*to communicate information and its relationships.*

3 The conceptual model

We use strings and dotted strings to define index sets for tables. We have chosen dotted strings simply because we want to incorporate the categorial labels directly in the dotted strings. We then use unordered Cartesian products of index sets and maps to obtain the tabular model.

Let Σ be an alphabet of symbols and $A \subseteq \Sigma^*$ be a finite set of strings over Σ. Then, a **dotted string** x over A is the **null dotted string** that we denote by Λ, is a string in A, or $x = y \cdot z$, where y is in A and z is a dotted string over A. For example, $Winter \cdot 1994$ is a dotted string over the set $\{1993, 1994, Fall, Winter, Spring\}$. The dotted string Λ satisfies the usual properties of a nullity: $x \cdot \Lambda = \Lambda \cdot x = x$, for all dotted strings x. Indices are dotted strings and index sets are sets of dotted strings that satisfy a simple, yet important, property. Given a dotted string x, a dotted string y is a **dotted prefix** of x if $x = y \cdot z$ for some dotted string z. Clearly, a dotted string is always a dotted prefix of itself and Λ is a dotted prefix of every dotted string.

A set X of dotted strings is **prefix free** if, for each dotted string x in X, there is no dotted string $y \in X$ such that $x \neq y$ and $x = y \cdot z$, for some dotted string z. The crucial point about a prefix-free set X of dotted strings is that it can be represented by a tree in which the root is labeled with Λ, each nonroot node is labeled with a string from A, and each root-to-frontier path spells out an index string in X. (We may alternatively choose to label each edge with a string from A and the conceptual incoming edge of the root with Λ.) We define an **index set** to be a prefix-free dotted-string set.

We are now in a position to define our tabular model. A **table** is defined by three items: A finite collection $\mathcal{I} = \{I_1, \ldots\}$ of index sets, a universe $\mathcal{E}$ of entry values, and a map τ from the unordered Cartesian product of the index sets to the universe of entry values. In other words, we have $\tau : \bigotimes \mathcal{I} \longrightarrow \mathcal{E}$. Thus, we use a tuple $(\mathcal{I}, \mathcal{E}, \tau)$ to denote a table, where we assume that the underlying alphabet is denoted by Σ and the underlying set of strings is denoted by A.

The map τ can be partial. It is of course, possible to allow τ to be a relation, but this generality is unnecessary in the majority of cases.

We now return to the discussion of prefix freeness. Although index sets such as $M = \{Assigns \cdot A1, Assigns \cdot A2, Assigns \cdot A3, Grade\}$ are prefix free, we can express them as a disjoint union of more than one index set. For example, we can partition M into M_1 and M_2, where $M_1 = \{Assigns \cdot A1, Assigns \cdot A2, Assigns \cdot A3\}$ and $M_2 = \{Grade\}$. Note that M_1 cannot be further partitioned without destroying its hierarchical nature. We can capture the partitioning of a prefix-free set as follows. First, we define a function $first$ that extracts the first A-string of a non-null dotted string. For a dotted string x, define $first(x)$ to be undefined if x is the null dotted string; otherwise, it is u, where $u \in A$ and $x = u \cdot v$, for some dotted string v. We can extend $first$ to apply to dotted-string sets as follows: For a dotted-string set Y, $first(Y) = \{u : u \in A \text{ and } u \cdot v \in Y, \text{ for some dotted string } v\}$. For example, $first(Assigns \cdot A2) = Assigns$. A dotted-string set X is **prime** if all dotted strings in X begin with the same string. In other words, X is prefix free and $\# first(X) = 1$.

Given a prefix-free set X, we can partition it into prime subsets in a unique way, if the prime subsets satisfy an additional condition. A subset Y of a dotted-string set X is a **maximal prime set with respect to X** if all the dotted strings that are in X but are not in Y begin with a different string from the one in $first(Y)$; that is, $first(Y) \cap first(X - Y) = \emptyset$. For example, M_1 and M_2 are maximal with respect to M, but M_1 is not maximal with respect to $M \cup \{Assigns \cdot A4\}$. We obtain the following characterization of partitionability.

Proposition 1. *Let X be a dotted-string set. Then, X can be partitioned into a finite number of prime sets, maximal with respect to X, if and only if X is prefix free. Moreover, this partition is unique.*

In practice, we often use index sets that are not prime; for example, the index set Marks of Table 2 is one such example. Usually, however, we prefer to

Table 2. The average marks for 1991–1992.

	Assignments			Examinations		Grade
	Ass1	Ass2	Ass3	Midterm	Final	
1991						
Winter	85	80	**75**	60	75	75
Spring	80	65	**75**	60	70	70
Fall	80	85	**75**	55	80	75
1992						
Winter	85	80	70	70	75	75
Spring	80	80	70	70	75	75
Fall	75	70	65	60	80	70

premultiply all dotted strings in the index set with one string to ensure that the index set is prime. For example, we can use Marks itself as such a string to obtain $Marks \cdot Assignments \cdot A1$, and so on. This transformation is similar to that used in relational databases when we introduce a universal relation or in an object-oriented environment when we introduce a superclass of all classes. If we do not modify a nonprime index set in this way, then we may not only partition the index set, but also we may partition the corresponding table. For example, we can partition Table 2 into three tables corresponding to the three prime sets obtained from Marks; see Tables 3, 4, and 5.

Prefix-free sets of strings are well known, they are used to define prefix codes. Thus, we can view index sets as codes over an alphabet A. It is well known that prefix codes define labeled trees whose root-to-frontier paths spell out the codewords.

We now introduce a number of useful notions for dotted-string sets. A **common dotted prefix** w of a set X of dotted strings satisfies the condition: *For*

Table 3. The average assignment marks for 1991–1992.

	Assignments		
	Ass1	Ass2	Ass3
1991			
Winter	85	80	75
Spring	80	65	75
Fall	80	85	75
1992			
Winter	85	80	70
Spring	80	80	70
Fall	75	70	65

Table 4. The average examination marks for 1991–1992.

	Examinations	
	Midterm	Final
1991		
Winter	60	75
Spring	60	70
Fall	55	80
1992		
Winter	70	75
Spring	70	75
Fall	60	80

Table 5. The average grades for 1991–1992.

	Grade
1991	
Winter	75
Spring	70
Fall	75
1992	
Winter	75
Spring	75
Fall	70

all dotted strings x in X, $x = w \cdot y$, for some dotted string y.

The **dotted length** $\|x\|$ of a dotted string x is 0, if $x = \Lambda$, and is $\|y\| + 1$, otherwise, where $x = y \cdot z$ and $z \in A$. If two dotted strings u and v are common dotted prefixes of a dotted-string set X and $u \neq v$, then either $\|u\| < \|v\|$ or $\|v\| < \|u\|$. Thus, we have the notion of a **longest common dotted prefix** of a set X of dotted strings.

Given two dotted-string sets X and Y, their **dotted product** $X \cdot Y$ is the dotted-string set $\{x \cdot y : x \in X \text{ and } y \in Y\}$. Corresponding to product we have quotient, which comes in two varieties. Given two dotted-string sets X and Y, we can divide Y on the left or on the right with X. We define the **left quotient** $X \backslash Y$ to be the set $\{w : x \in X \text{ and } x \cdot w \in Y\}$ of dotted strings. We define the **right quotient** Y/X to be the set $\{w : x \in X \text{ and } w \cdot x \in Y\}$ of dotted strings.

4 Dotted-string operations

An effective tabular editor must provide operations that change index sets as well as operations that act on entries and labels. The reason is simple. A user may wish to change the structure of an index set by: adding new labels and indices, removing labels and indices, or modifying indices. These index-set operations involve the addition, removal, and rearrangement of entries in a displayed table. As index sets can have a rich structure, such operations are crucial during tabular design when index sets are more fluid that they will be once the design is frozen (if that ever happens).

Within the dotted-string model, we should have dotted-string set operations that give appropriate operations for index sets. Thus, we are interested in what are the primitive and meaningful operations on index sets that would form the basis of tabular editing operations. We identify two simple operations: adding and removing an index string from an index set. As we shall demonstrate we can specify the semantics of other useful index-set operations in terms of these two.

Addition: $[X' = X + x$ and $X + Y]$. We want to be able to add index strings to an index set, essentially, to form $X \cup \{x\}$, for an index set X and a dotted string x. The set-theoretic union does not always preserve prefix freeness however, so we have to define addition carefully. For an index set X and an index string x, X' is defined as follows:

1. If x is a dotted prefix of some dotted string in X, then $X' = X$.
2. If there is some dotted string $y \neq x$ in X such that y is a dotted prefix of x, then $X' = (X - \{y\}) \cup \{x\}$. Note that there can be at most one such string y in X.
3. Otherwise, $X' = X \cup \{x\}$ and $\#X' = \#X + 1$.

We generalize Addition to allow the second operand to also be an index set. In this case, $X' = X + Y$, where X and Y are both index sets, is defined to be: X, if $Y = \emptyset$; otherwise, it is $(X + y) + (Y - \{y\})$, where $y \in Y$.

Removal: $[X' = X - x$ and $X - Y]$. If we remove a dotted string from an index set, we still have a prefix-free set and an index set. Thus, $X - \{x\}$ is well

defined. We define a more general notion of removal, however, that is more appropriate and useful. It removes any dotted string from X that has x as a dotted prefix. It corresponds to the set-theoretic difference when $x \in X$ since, in this case, x is the only dotted string in X having x as a dotted prefix. We define X' as follows: $X' = \{y : y \in X$ and x is a not a dotted prefix of $y\}$.
We generalize Removal to allow the second operand to also be a dotted-string set. In this case, $X' = X - Y$, where X is an index set and Y is a dotted-string set, is defined to be: X, if $Y = \emptyset$; otherwise, it is $(X - y) - (Y - \{y\})$, where $y \in Y$. Observe that although Y need not be an index set, the result $X - Y$ is still an index set.

5 Edit operations

We demonstrate the effectiveness of the conceptual model by showing how we can apply it to specify the semantics of some example edit operations for tables. One initial comment. We define an **empty table** $(\mathcal{I}, \mathcal{E}, \tau)$ to satisfy the conditions: $\mathcal{I} = \emptyset$, $\mathcal{E}$ to be any set of values, and $\tau(())$ to be undefined. We define an **empty category** to be the set $\{\Lambda\}$, rather than the set $\emptyset$ to ensure that the domain of τ is nonempty. We can add an empty category to a table to increase its dimension, yet not increase its size! We model or represent categories with dotted-string sets and, in addition, we use the concept of a **subcategory;** for example, *Examinations* is a subcategory of *Marks*. We define a subcategory formally using dotted strings. For a dotted-string set X that represents a category C, a dotted-string set Y represents a subcategory S of C if and only if there is a dotted string u such that $\{u\}\backslash X = Y$.

Add-Subcategory: We want to add a category S as a subcategory in a category C of a table $T = (\mathcal{I}, \mathcal{E}, \tau)$. We have to specify where it should be inserted; that is, we have to provide a dotted string x that is a dotted prefix of some of the dotted strings in C. For example, we want to add a new category Special as a subcategory of Term, where $Special \cdot June$ and $Special \cdot July$ correspond to two extremely short special terms. Thus, we should obtain the new indices $Term \cdot Special \cdot June$ and $Term \cdot Special \cdot July$.
Now, Add-Subcategory(S,C,x,T) is defined by: $T' = (\mathcal{I}', \mathcal{E}, \tau')$, where $\mathcal{I}' = (\mathcal{I} - \{C\}) \cup C'$ and τ' is identical to τ over their common domain and is undefined for the new domain values. C' is modeled by an index set; thus, we obtain: If $\{x\}\backslash C \neq \emptyset$ (there are indices with x as a dotted prefix), then

$$C' = C + (\{x\} \cdot S);$$

otherwise, the operation is undefined.

Remove-Subcategory: We want to remove a subcategory S in a category C of a table $T = (\mathcal{I}, \mathcal{E}, \tau)$. We have to specify from where S should be deleted; that is, we have to provide a dotted string x that is a dotted prefix of some of the dotted strings in C. For example, we want to delete the subcategory

$Special = \{Special \cdot June, Special \cdot July\}$ from the category Term$'$ used in the previous definition.

Now, Remove-Subcategory(S,C,x,T) is defined by: $T' = (\mathcal{I}', \mathcal{E}, \tau')$, where $\mathcal{I}' = (\mathcal{I} - \{C\}) \cup C'$ and τ' is identical to τ over their common domain. C' is modeled by an index set; thus, we obtain: If $\{x\} \backslash C \neq \emptyset$ (there are indices with x as a dotted prefix), then

$$C' = C - (\{x\} \cdot S);$$

otherwise, the operation is undefined. Now, C' may be the empty set, in which case we redefine it to be $\{\Lambda\}$ for consistency.

Move-Subcategory: We want to move a subcategory S of a category C within C. We specify S with a dotted string s and its new position with a dotted string t. For example, we want to move the subcategory $A3$ of $Marks$ to be a subcategory of $Examinations$ as a first step in renaming $A3$ as $Quiz$ and treating it as an examination. It is crucial that s is not a dotted prefix of t, although the converse is acceptable.

Now, Move-Subcategory(S,C,s,t,T) is defined by: $T' = (\mathcal{I}', \mathcal{E}, \tau')$, where $\mathcal{I}' = (\mathcal{I} - \{C\}) \cup C'$ and τ' is identical to τ over their common domain. We have to specify the value of τ over its new domain values. We may assume that C is the first category in the domain of τ, as it is an unordered product of $\mathcal{I}$. Now, the only change for the values of τ are that a first index of the form $s \cdot z$, where $z \in S$, no longer occurs. It has been replaced in τ' by an index of the form $t \cdot z$. Thus, for all $z \in S$, we define $\tau'(t \cdot z, i_2, \ldots, i_d) = \tau(s \cdot z, I_2, \ldots, i_d)$.

As C' is modeled by an index set, we obtain: If $\{s\} \backslash C \neq \emptyset$ (there are indices with s as a dotted prefix), $\{t\} \backslash C \neq \emptyset$ (there are indices with t as a dotted prefix), and $\{s\} \backslash \{t\} = \emptyset$ (s is not a dotted prefix of t), then

$$C' = (C - (\{s\} \cdot S)) + (\{t\} \cdot S);$$

otherwise, the operation is undefined.

Combine-Categories: We want to combine two categories C and D of a table $T = (\mathcal{I}, \mathcal{E}, \tau)$ into one new category. The operation does not change the size of the table, but it changes its dimension. Normally, we want to remove any common prefix from the indices in D. For example, we can combine the categories $Year = \{Yr \cdot 91, Yr \cdot 92\}$ and $Term = \{Term \cdot W, Term \cdot Sp, Term \cdot F\}$ to give a new category

$$Year@Term = \{Yr{\cdot}91{\cdot}W, Yr{\cdot}91{\cdot}Sp, Yr{\cdot}91{\cdot}F, Yr{\cdot}92{\cdot}W, Yr{\cdot}92{\cdot}Sp, Yr{\cdot}92{\cdot}F\},$$

where we have used obvious abbreviations and @ is a new symbol. Note that we have removed the prefix $Term$ from the index strings in Term.

Combine-Categories(C,D,x,T) is defined by: $T' = (\mathcal{I}', \mathcal{E}, \tau')$, where $\mathcal{I}' = (\mathcal{I} - \{C\} - \{D\}) \cup C'$.

Clearly, we want to model this operation by a product of index sets; If $D = \{x\} \cdot (\{x\} \backslash D)$ (x is a common prefix of the index strings in D), then we define $C' = C \cdot (\{x\} \backslash D)$

We can assume that categories C and D occur as the first two categories in $\mathcal{I}$ and that C' occurs as the first category in $\mathcal{I}'$. Thus, τ' is defined by: $\tau'(c \cdot (x \backslash d), i_3, \ldots, i_d) = \tau(c, d, i_3, \ldots, i_d)$.
Using this operation we can convert any d-dimensional table into a one-dimensional one. Presentationally, such a table has only either a box head or a stub.

6 Expressiveness of the conceptual model

We have made the simplying assumption that we do not model footnotes in the conceptual model. Clearly, footnotes play an important role in tables. For example, in the book *Human Activity and Environment* [1], 148 of the 172 tables have footnotes. The conceptual model does not capture all tables even when we ignore footnotes but in our view it is sufficiently powerful to be useful—it provides at a "70 percent solution" to use one of guidelines of software engineers. Footnotes can, however, be included but they are passed without interpretation to the user's text formatter. This approach is viable since the footnotes themselves do not appear within tables, they are placed outside the tables. Only the associated footnote marks are included in the tables.

The model can be used to specify only tables that have a structure that corresponds to the conceptual model. Some tables, however, are a combination of several tables. Some do not distinguish between a label and an entry and some do not even have a rectangular frame. For example, Table 6 is a combination of three tables. There are three index sets: X, Y, and *Type of calculations* (the index set in the stub head) in this table. The first subtable, whose entries are associated with the categories X and *Type of calculations*, has been placed in the boxhead. The second subtable, whose entries are associated with the index sets Y and *Type of calculations*, has been placed in the stub. The third subtable, whose entries are associated with index sets X and Y, has been placed in the body.

We examined tables in books from various sources, including statistics, sociology, science, and business. The results of the experiment reveal that the conceptual model can be used to specify 56 percent of the tables if we consider footnotes or 97 percent of the tables if we ignore footnotes. From this experiment, we see that the majority of the tables in traditional printed documents can be specified with our conceptual model.

7 Last words

We have left a number of issues unresolved, both implications for other tabular manipulations, such as display, and completeness of the edit operations. Wang [21], and Wang and Wood [23] have developed a style language for tables that allows a user to specify global, and local styles for tables with respect to a table's conceptual structure, topological structure, and presentational structure.

Although this work is based on an earlier abstract model for tables, there is little difficulty in adapting it for the conceptual model we have described.

All models have their up sides and down sides. On the up side, the conceptual model captures the hierarchical structure of index sets and their unordered combination—key properties in our opinion. The downside is that the frugality of the model moves the specification of orderings outside the model, although the user of a tabular system should be unaware of this fact. We chose this option since we wanted to separate structure from display; it is sufficient that we can hang display issues on top of the conceptual model.

Lastly, a number of readers of the draft of this paper commented that there is some overlap in ideas with the work on OLAP (On-Line Analytical Processing), a key high-level notion in data mining and knowledge discovery [8, 9, 11]. OLAP is especially concerned with the processing of very large collections of data that have high dimensionality. Although the viewpoint and concern of OLAP is different, there are indeed similarities in the models. We will address their similarities and differences in the full version of this paper.

References

1. *Human Activity and the Environment—A Statistical Compendium.* Statistics Canada, 1986.
2. M.P. Barnett. *Computer Typesetting: Experiments and Prospects.* MIT Press, 1965.
3. R.J. Beach. *Setting Tables and Illustrations with Style.* PhD thesis, Dept. of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, May 1985. Also issued as Technical Report CSL-85-3, Xerox Palo Alto Research Center, Palo Alto, CA.
4. J.L. Bentley. *More Programming Pearls: Confessions of a Coder.* Addison-Wesley, Reading, MA, 1988. Chapter 9 is an excellent apologia for little languages.
5. T.J. Biggerstaff, D.M. Endres, and I.R. Forman. TABLE: Object oriented editing of complex structures. In *Proceeding of the 7th International Conference on Software Engineering*, pages 334–345, 1984.
6. F.P. Brooks. *The Mythical Man-Month: Essays in Software Engineering.* Addison-Wesley, Reading, MA, second edition, 1975. Reprinted with corrections, January 1982.
7. J.P. Cameron. A cognitive model for tabular editing. Technical Report OSU-CISRC-6/89-TR 26, The Ohio State University, Columbus, OH, June 1989.
8. S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26(2):65–74, 1997.
9. E.F. Codd, S.B. Codd, and C.T. Shelley. Providing OLAP (On-line Analytical Processing to User-Analysts. Technical report, Codd & Date, Inc., 1993.
10. R. Furuta. *An Integrated but not Exact-Representation, Editor/Formatter.* PhD thesis, Dept. of Computer Science, University of Washington, Seattle, WA, September 1986. Also issued as Technical Report 86-09-08, University of Washington.
11. C.-T. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in OLAP data cubes. *ACM SIGMOD Record*, 26(2):73–78, 1997.

12. International Organization for Standardization. *ISO 8879, Information processing — Text and office systems — Standard Generalized Markup Language(SGML)*, October 1986.
13. International Organization for Standardization and International Electrotechnical Commission. *ISO/IEC TR 9573:1988(E), Information processing — SGML Support Facilities — Techniques for Using SGML*, 1988.
14. International Organization for Standardization and International Electrotechnical Commission. *ISO/IEC TR 9573-11:1992(E), Information processing — SGML Support Facilities — Techniques for Using SGML*, 1992.
15. L. Lamport. *LaTeX: A Document Preparation System.* Addison-Wesley, Reading, MA, 1985.
16. M.E. Lesk. Tbl—a program to format tables. In *UNIX Programmer's Manual*, volume 2A. Bell Telephone Laboratories, Murray Hill, NJ, 7th edition, January 1979.
17. V. Quint and I. Vatton. Grif: An interactive system for structured document manipulation. In *Text Processing and Document Manipulation, Proceedings of the International Conference*, pages 200–312, Cambridge, UK, 1986. Cambridge University Press.
18. D.R. Raymond. *Partial Order Databases.* PhD thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 1996.
19. C. Vanoirbeek. *Une Modelisation de Documents pour le Formatage.* PhD thesis, Departement d'Informatique, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 1988.
20. C. Vanoirbeek. Formatting structured tables. In C. Vanoirbeek & G. Coray, editor, *EP92 (Proceedings of Electronic Publishing, 1992)*, pages 291–309, Cambridge, UK, 1992. Cambridge University Press.
21. X. Wang. *Tabular Abstraction, Editing and Formatting.* PhD thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 1996. Available as Research Report CS-96-09, Department of Computer Science, University of Waterloo.
22. X. Wang and D. Wood. An abstract model for tables. TUGBOAT*, The Communications of the TeX Users Group*, 14(3):231–237, October 1993.
23. X. Wang and D. Wood. Xtable—a tabular editor and formatter. *Electronic Publishing: Origination, Dissemination and Design*, 8:167–179, 1995. Special issue for papers that appeared in Electronic Publishing '96.

This article was processed using the LaTeX macro package with LLNCS style

Table 6. Correlation table — wheat and flour prices by months, 1914–1933.

X

class interval						.40 -.59	.60 -.79	.80 -.99	1.00 -1.09	1.20 -1.39	1.40 -1.59	1.60 -1.79	1.80 -1.99	2.00 -2.19	2.20 -2.39	2.40 -2.59	2.60 -2.79	2.80 -2.99	Total	f(dxdy)
	mid-point					5	7	9	11	13	15	17	19	21	23	25	27	29		
		deviation d				0	1	2	3	4	5	6	7	8	9	10	11	12		
			frequency f			20	6	25	37	52	24	15	15	13	18	6	5	4	240	
				fd		0	6	50	111	208	120	90	105	104	162	60	55	48	1119	
					fd^2	0	6	100	333	832	600	540	735	832	1458	600	605	576	7217	
15.00 -15.99	15.5	12	1	12	144													1		144
14.00 -14.99	14.5	11	5	55	605											1	2	2		616
13.00 -13.99	13.5	10	5	50	500										1	2	1	1		520
12.00 -12.99	12.5	9	10	90	810										6	2	2			864
11.00 -11.99	11.5	8	5	40	320									1	3	1				360
10.00 -10.99	10.5	7	14	98	686									6	8					840
9.00 -9.99	9.5	6	17	102	612						1	1	10	5						726
8.00 -8.99	8.5	5	28	140	700					4	8	11	4	1						790
7.00 -7.99	7.5	4	46	184	736			2	7	22	12	3								764
6.00 -6.99	6.5	3	54	162	486			5	20	25	3		1							576
5.00 -5.99	5.5	2	16	32	64		1	4	10	1										86
4.00 -4.99	4.5	1	34	34	34	15	5	14												33
3.00 -3.99	3.5	0	5	0	0	5														0
Total			240	999	5697															6319

Y (rows)

X= Wheat price per bushel in dollars; Y= Flour price per barrel in dollars.