Tabular Abstraction, Editing, and Formatting

by

Xinxin Wang

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 1996

Abstract

This dissertation investigates the composition of high-quality tables with the use of electronic tools. A generic model is designed to support the different stages of tabular composition, including the editing of logical structure, the specification of layout structure, and the formatting of concrete tables. The model separates table's logical structure from its layout structure, which consists of tabular topology and typographic style. The notion of an abstract table, which describes the logical relationships among tabular items, is formally defined and a set of logical operations is proposed to manipulate tables based on these logical relationships. An abstract table can be visualized through a layout structure specified by a set of topological rules, which determine the relative placement of tabular items in two dimensions, and a set of style rules, which determine the final appearance of different items. The absolute placement of a concrete table can be automatically generated by applying a layout specification to an abstract table. An NP-complete problem arises in the formatting process that uses automatic line breaking and determines the physical dimension of a table to satisfy user-specified size constraints. An algorithm has been designed to solve the formatting problem in polynomial time for typical tables. Based on the tabular model, a prototype tabular composition system has been implemented in a UNIX, X Windows environment. This prototype provides an interactive interface to edit the logical structure, the topology and the styles of tables. It allows us to manipulate tables based on the logical relationships of tabular items, regardless of where the items are placed in the layout structure, and is capable of presenting a table in different topologies and styles so that we can select a high-quality layout structure.

Acknowledgements

I would like to express my gratitude to my supervisor, Professor Derick Wood, for his support, advice and encouragement. Without his tireless reading and constructive criticism, there would have been no dissertation.

I would also like to acknowledge my external examiner, Professor Richard Furuta from the Texas A & M University, and the remaining members of my thesis committee, Professors Donald Cowan, Frank Tompa, and David Matthews for their contribution to the improvement of this dissertation.

My thanks to Darrell Raymond for his help in providing information about tabular typesetting.

Professor Ming Li made suggestions for Chapter 5 (Formatting) and proofread the chapter. I also benefited much from discussions with Dr. John Tromp on the definition of the tabular formatting problem and with Professor Qiang Yang on the final algorithm.

The Department of Computer Science, University of Waterloo, provided me with an excellent study and work environment.

I am grateful for the financial support I received from the Information Technology Research Center of Ontario and the Natural Sciences and Engineering Research Council of Canada.

To my friends, Roger Skubowius and Jane Liang, thank you for your friendship, help and encouragement along the way.

Finally, to my parents, who provide endless love and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis investigates different issues of tabular composition: abstract model, layout specification, editing and formatting. We consider the composition of tables to be one of the most challenging aspects of document typesetting. Tables may contain different kinds of objects, such as text, graphics, mathematical formulas, and so on, which display distinct characteristics and need different treatment. From the logical point of view, tables are multi-dimensional objects. They are, however, usually presented in two dimensions. Tabular typesetting needs to solve the same problem that we need to solve to typeset text and tabular typesetting raises additional problems that need to be solved. Moreover, we often need to explore different layouts and styles of the same tables, so that we can choose one layout and style that presents the table's data in a convincing way. We do not know of any generally available tabular composition system that satisfies these requirements. We introduce various aspects of tabular composition in this chapter. We first describe the characteristics of tables. We then discuss the different stages that are involved in tabular composition. Next, we review the development of tabular composition systems and give our research objectives. Finally, we state the major research contributions of the thesis.

Table 1.1: 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

1.1 Definition and characteristics

It may be easy to point out a table in a book, but a precise definition of a table is elusive. The *Oxford English Dictionary* defines a table as: "An arrangement of numbers, words or items of any kind, in a definite and compact form, so as to exhibit some set of facts or relations in a distinct and comprehensive way, for convenience of study, reference, or calculation." This definition summarizes the characteristics of a table using three different aspects: content, form and function.

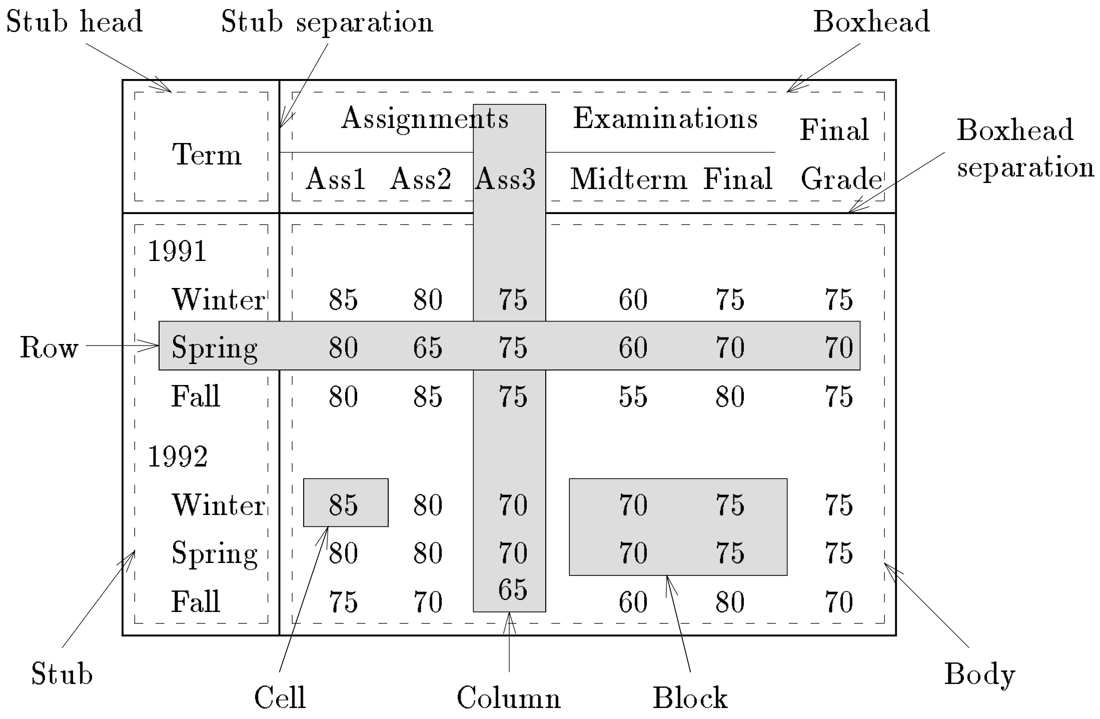
1.1.1 The content of a table

The content of a table is a collection of interrelated items, which may be numbers, text, symbols, figures, mathematical equations, or even other tables. Some of the items are the basic data a table displays, and the others are the auxiliary data that are used to locate the basic data. We use the term *entries* to denote the former kind of data and the term *labels* to denote the latter kind. Labels are further classified into *categories* that are organized hierarchically. For example, Table 1.1 presents the average marks of the assignments and examinations for a course offered in the three trimesters of 1991 and 1992. The marks are entries and the strings that denote the years, the terms, and the kinds of marks are labels. Furthermore, **Year** is a category that consists of the labels **1991** and **1992. Term** is another category that consists of the labels **Winter, Spring,** and **Fall. Mark** is a category that consists of the subcategories **Assignments** and **Examinations** and the label **Final Grade**. There are logical relationships between the entries and the labels. Each entry is associated with one label from each of the categories. For example, the entry **85** at the top-left corner is associated with the labels **1991, Winter,** and **Ass1,** and the entry **70** at the bottom-right corner is associated with the labels **1992, Fall,** and **Final Grade.** The tabular items and their logical relationships provide the *logical structure* of the table and the number of categories defines the logical dimension of the table. Table 1.1 has three categories; thus, it is a three-dimensional table.

1.1.2 The presentational form of a table

The content of a table must be presented in some form and on some medium. Usually, tables are presented as row–column structures on a planar medium, such as paper or screen. Fig. 1.1 defines the terminology for the parts of a table represented as a row–column structure. We inherit the most terminology from *The Chicago Manual of Style* [Chi93] except that we define the concepts of 'stub head' and 'block' for our convenience. 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 categories in the stub, and the *body* is the region to the right of the stub and below the boxhead that contains the entries. The intersection of a row and a column is called a *cell* and a rectangular collection of cells is called a *block.*

In traditional tabular presentation, the entries are usually put in the body of a table and the labels are placed in the stub or in the boxhead. To present multidimensional tables in two dimensions, we have to associate more than one category with the stub or with the boxhead. In this case, some labels appear more than once. For example, in the stub of Table 1.1, the labels of category **Term** appear twice so as to present the asso-



Term	Assignments			Examinations		Final
	Ass1	Ass2	Ass3	Midterm	Final	Grade
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

Figure 1.1: The terminology for the row–column presentational structure of table.

ciations between **Year** and **Term.** The arrangement of labels decides the arrangement of entries. Each entry is usually put in a cell such that it is to the right of its associated labels in the stub and beneath its associated labels in the boxhead. Different types of typographic cues can be used to help readers search for information in a table. We can use rules or white space to separate tabular items, distinct type faces or point sizes to distinguish different types of items, and background colors or patterns to highlight important information.

Although the row–column structure is a familiar and natural form for tabular presentation, tabular data can also be presented in other forms. For example, Fig. 1.2 is a pictorial form of Table 1.2. The combination of pictorial form and the row–column structure can increase the accuracy of obtaining tabular information [PLSS84]. The reasons are that the row–column structure provides precise information for a particular question, while the pictorial form provides general information for browsing and comparison. Various graphical techniques have been investigated to reveal tabular information with visual presentation [Bri14, Tuf90, Tuf83, Zei85]. In this thesis, we focus on presenting tables only in the row–column structure. We use *layout structure* to denote the presentational form of a table.

1.1.3 The function of a table

The main function of tables is to present detailed information in a compact way such that the ability to search and compare the information is enhanced. Since tabular information is conveyed by its presentational form, one critical factor in determining how easily tables can be read depends on the presentational forms selected by the designer. This selection is motivated, in part, by an understanding of how users interact with tables [Wri82]. At least three cognitive processes are involved in users' interaction with a table [Wri80a, Wri80b]:

1. A comprehension process, needed for understanding the principle on which the table is organized to grasp the underlying logical structure of the table.

2. A search process, needed for locating the relevant information within the table.

3. An interpretive and comparative process, needed to answer specific questions after the relevant information has been obtained.

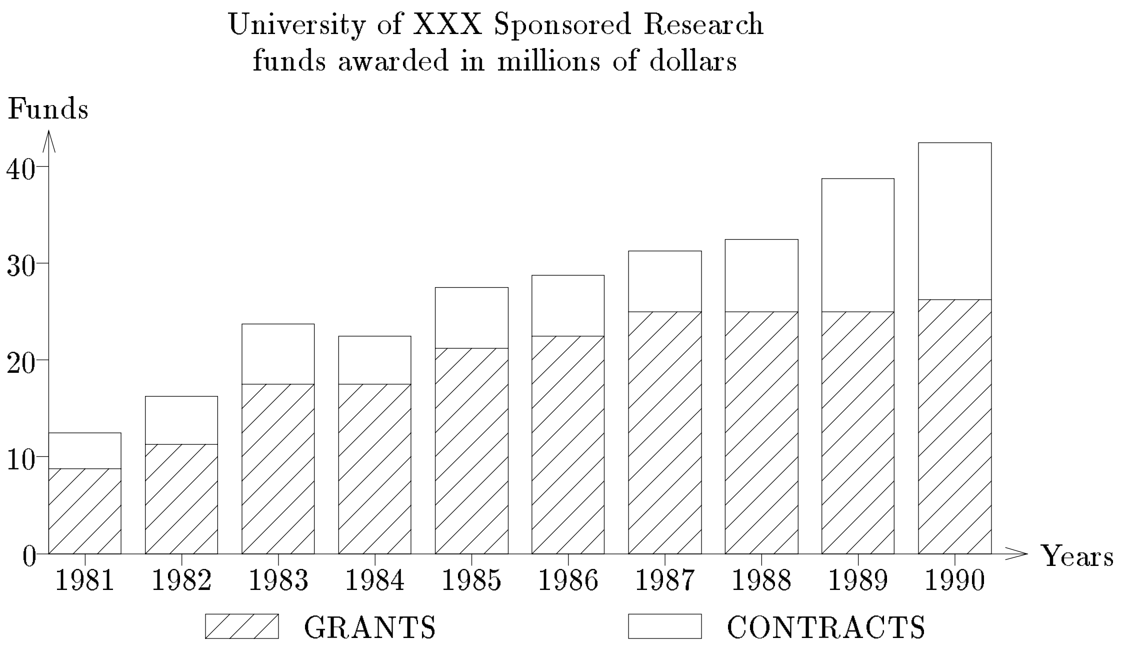


Figure 1.2: A pictorial form of Table 1.2.

Table 1.2: University of XXX sponsored research funds (in millions of dollars).

Year	Fund	
	Grant	Contract
1981	8.8	4.0
1982	11.6	5.1
1983	18.0	6.4
1984	18.0	5.1
1985	21.6	6.8
1986	22.8	6.4
1987	25.1	6.4
1988	25.1	8.0
1989	25.1	14.4
1990	26.8	16.8

Based on these three cognitive processes, a well-designed table should be organized in such a way that the underlying logical structure is made obvious and tabular items are located and interpreted easily.

1.2 Tabular composition

In traditional typesetting, tables were always treated separately from the main body of text. They involve the most frequent change in typographic style within the text and require the skill of a compositor or typesetter to handle them [Wil83]. Two kinds of people are involved in the composition of tables. The author is mainly responsible for the design of the logical structure and the topological arrangement whereas the graphic designer is concerned with the presentational style. Since authors know their subject material and graphic designers are familiar with aesthetic principles and publishers' styles, their cooperation guarantees the production of high-quality tables. Nowadays however, anybody can produce tables with the help of a tabular editor and formatter. Although users may know their subject material very well and be quite familiar with statistical principles, they still may not know how to design high-quality tables, or even realize the shortcomings of a given table.

The criteria for a well-designed table may differ for different people and different purposes. The most important criteria that most people agree with are legibility and accuracy. A well-designed table should enable readers to obtain information rapidly and make few errors. With respect to the three cognitive processes we mentioned in Section 1.1.3, a well-designed table should enable readers to learn to use the table quickly with little or no instruction in the comprehension process, to locate information easily and accurately in the search process, and to avoid the time-consuming calculations that provide opportunities to make mistakes in the interpretive and comparative process. The other criteria that affect the design of tables are the style of a publisher, the space constraints of the medium on which a table is to be displayed, and the purpose for which a table is to be used.

Researchers from psychology, statistics, and typography have suggested possible guidelines for the design of high-quality tables. Hall [Hal43] discusses the principles and technical details involved in the design of statistical tables and the improvement of the tables. He gives rules to guide the design of tables and gives examples that illustrate how badly-designed tables can be improved by applying these rules. In a series of papers [WF70, Wri68, Wri77], Wright provides guidelines for the design of tables based on previous cognitive research and experiments conducted by herself and others. Her research focuses on improving the comprehension of the underlying logical structure of a table and on improving the effectiveness of obtaining tabular information. Ehrenberg [Ehr77] provides basic precepts for the presentation of numerical data which have largely been ignored in statistical practice. These precepts can be used to address the criterion that the underlying logical structure of a table should be obvious at a glance with little or no instruction. Norrish [Nor89] gives the conventions for tabular presentation in the traditional publishing industry. These conventions not only address the issues of how to present the table body, but also how tables are related to the surrounding text. *The Chicago Manual of Style* [Chi93] is the standard style guide for authors and editors involved in publishing. It devotes one section to the conventions and techniques for tabular typesetting, including the arrangement of elements, the selection of styles for different components, and the handling of different sizes and shapes. Zeisel [Zei85] draws attention to the difficulties in the presentation of statistical tables and presents some solutions to these difficulties. He also discusses analytical techniques for the refinement of statistical tables to meet readers requirements. The emphasis of his book is on the relationships among data that describe what is and what happens, rather than on issues of presentational form.

Based on these studies, we have abstracted guidelines for the design of high-quality tables at different stages of composition. These guidelines are summarized.

1.2.1 Logical structure design

At this stage, we decide the content of a table by taking into account the readers' requirements and convenience. There are three guidelines for this stage:

1. Contain only necessary information [Hal43, Zei85]

   Suppose a course instructor needs to design a table to show students their final marks. Students are concerned not only with their marks but also how their marks compare with those of other students. Thus, the table should list not only the marks for each student, but should also give the average, minimum and maximum marks. On the other hand, a large table with complex structure needs more time to comprehend. If a table contains more information than readers need, it is better to simplify the table by combining items and removing redundant and unrelated items. For example, if a department chair wants to examine the marks for a course, he or she is probably interested only in the average, minimum, and maximum marks, and how many students have failed the course. Thus, a table for the chair need not display the marks of every student. We can consider such a summary table a view, in the database sense, of the original table.

2. Present a table as an explicit structure [WF70]

   A table in which all the information is given explicitly such that a reader needs only to locate the required item, is called an *explicit structure*. A table that contain all necessary information, but requires readers to do some calculation after locating an item is called an *implicit structure*. For example, if we design a table conversion from pounds to kilograms in the range 0 through 99 pounds, an explicit structure will list all the conversion values for 0, 1, ..., 99 pounds, whereas an implicit structure may list only the conversion values for 0, 1, ..., 9 pounds and 10, 20, ... 90 pounds. The implicit structure requires readers to do an addition if they want to know how many kilograms are equivalent to 55 pounds. Obviously, an explicit structure is more efficient for the reader and an implicit structure's presentation normally uses less space.

3. Reduce the number of categories and subcategories as appropriate [Wri77, Zei85]

   Experiments carried out by Wright have shown that increasing the number of decisions to be made is a handicap in reading tabular information. The number of decisions is proportional to the number of categories and the number of subcategories in each category. We can combine categories to reduce the logical dimension or merge two levels of labels to lower the depth of a category. For example, we can combine the categories **Year** and **Term** in Table 1.1 to form a new category that has labels **W91**, **S91**, **F91**, **W92**, **S92** and **F92.** One advantage of the reduction of the logical dimension or the depth of category is that it can save space in the presentation of a table.

1.2.2 Tabular arrangement

After we decide on the content of a table, we need to arrange the items in two dimensions so that the logical structure of the table is clearly seen. Some guidelines for this stage are:

1. Place related items close together [WF70, Ehr77]

   Placing related items close together helps readers locate and compare information. For example, university terms are normally used in connection with years. It would be unwise to change the topological arrangement of Table 1.1 by placing category **Term** in the boxhead and category **Year** in the stub. Similarly, **Midterm** and **Final** are closely related in that they are both examinations. It is inappropriate to separate them with other items.

2. Avoid using two dimensions whenever possible [WF70, Wri68]

   Although presenting a table in two dimensions (using both row and column headings) saves space, a two-dimensional structure is more difficult to comprehend than a one-dimensional structure because readers need to integrate a row heading and a column heading simultaneously to locate a cell. This guideline is, however, appropriate for tables that have only one or two categories, even though we can always reduce the dimension of a table to one. When a table has more than two categories, it is better to present it in two dimensions. A one-dimensional presentation of a table with three or more categories is hard to read, and it is aesthetically displeasing.

3. Place the most frequently referenced items to the left or at the top of a table [Wri68]

   Westerners are used to reading information from left to right and from top to bottom. These reading habits greatly affect the way we read tables. Previous studies provide evidence that searching from left to right takes less time than searching from right to left. That is why in traditional tabular presentation labels are usually put in the stub and boxhead and entries in the body.

4. Vertically arrange items to be compared [WF70, Ehr77]

   It is easier to search and compare items reading down a column rather than reading across a row, especially for a large number of items [Ehr77]. For example, it is easier to compare a group of decimal numbers that are aligned vertically on their decimal points.

5. Arrange items in some meaningful order [Hal43, Ehr77]

   Arranging the rows and columns in some meaningful order often enables readers to see the overall distribution of the data. It also helps readers to compare a particular entry with others. For example, if we want to generate a table to show students the marks in a course, we may want to sort the students' names in decreasing order of their marks.

1.2.3 Presentational style

Finally, we are at the stage of selecting a presentational style for a table. In the world of publishing, various publishers have their own styles for tabular presentation [Chi93, AAU78]. These styles control the general appearance of tables throughout a publication, although for some particular tables, we may need to specify specific styles. Some guidelines for the selection of presentational style are:

1. Use type sizes between 8 and 12 point [WF70, Chi93]

   Researchers have found that an 8-point typeface is more legible than a 6-point typeface in mathematical tables [Tin30], and for non-numerical material a type size larger than 12 point can reduce reading efficiency [Spe68]. Type sizes between 8 point and 12 point are the best choices.

2. Separate and group items by spaces or rules [Hal43, WF70, Ehr77, Nor89]

   Occasionally using spaces and rules to separate or group items can help the readers' eyes to align the items across a row and down a column. Wright [WF70] has observed that it is better to leave less space between related columns than between unrelated ones [Wri73]. Widely spaced items require the readers' eyes to travel too far and slow down the searching process. Tinker [Tin60] has found that grouping rows is much more helpful than having all rows equally spaced, and grouping rows into blocks of approximately five rows is the best solution. With the advent of mechanization in typesetting, such as the use of linotype machines, it became difficult and expensive to typeset vertical rules. Consequently, there was a universal trend by publishers to give up the use of vertical rules. Although there is no longer a problem in generating vertical rules with computer-aided typesetting, many publishers still maintain this style and many style manuals, such as *The Chicago Manual of Style* [Chi93], still do not advocate the use of vertical rules.

3. Use typographic cues to distinguish different kinds of items [WF70]

   Previous studies indicate that distinguishing different kinds of items by typographic cues, such as type faces, type sizes, foreground and background colors, and patterns, can significantly reduce errors when reading a table. Typographic cues can help readers scan selectively and locate the appropriate answers more easily.

4. Flush left and indent the row headings in the stub [Chi93, Nor89]

   Many publishers prefer to left justify the row headings left in the stub. If there are two or more levels of subheadings, successive levels are indented at least two quads from the previous levels. Tables presented in this way not only clearly display the logical structure but also use less space.

5. Align the items as appropriate for different classes of items [Chi93]

   For example, numbers should be aligned vertically on decimal points, dollar symbols, pound symbols, or percentage symbols. Mathematical formulae are aligned on operators (such as $+, -, <, =$, and so on). For columns that contain text, if all entries are short, then they may be centered in the column. Long segments of text and mixed-length segments of text are normally left justified.

6. Round numbers to just two or three significant digits [Ehr77]

   It is difficult to compare a pair of numbers and calculate their difference mentally if the numbers are too long. Rounding numbers to two or three significant digits makes comparison easier.

7. Span the items that contain the same value [Chi93]

   If adjacent entries contain the same values, we can present the common value once and place it in the center of the area occupied by these entries. An item that occupies more than one table cell is *spanned*. Spanning items enable us to easily comprehend which entries share the same value and may reduce the presented table size if the common items occur very frequently.

1.2.4 Dealing with size and shape

At all stages of tabular composition, we should take into account the space limitations of the medium on which a table is presented and the proportion between tabular width and height [Chi93]. No publisher is happy to see a tall, thin table or a short, fat one that must be printed broadside. Also, the variation among column widths and among row heights affects the appearance of a table. We would prefer not to have a table that contains one column that is a centimeter wide and another one that is 10 centimeters wide.

If a table has unsatisfactory size or shape, we can improve it by changing the content, the topological arrangement, or the typographic style of the table. To change the content of a table, we can remove unnecessary information and use shorter text to make a large table smaller, or replace abbreviations with their complete forms to make a narrow column wider. To change the topological arrangement, we can transpose a fat and short table, or move some categories from the stub to the boxhead for a tall and thin table. To change the typographic style, we can select smaller type sizes and reduce the white space between columns and rows of a large table, or change the sizes of columns and rows to correct unpleasant proportions between them. When we change the width of a column that contains long text, the line-breaking points have to be adjusted to fit the new width.

If we cannot place a large table on one page, then we have to use other typesetting techniques. We can break a table that is too tall, but is not too fat, into multiple pages by duplicating the column headings for each page. For a table that is too fat for one page, we can print it broadside or print it on facing pages. If a table is still too fat, then we have to print it on a larger sheet of paper and fold it, an expense that no publisher likes to incur, except for important tables in profitable books.

1.3 Review of previous work

We first briefly describe the development of computer-aided document typesetting and how it has affected the evolution of electronic tabular composition. We then describe several tabular composition systems in some detail. Finally, we evaluate these systems according to criteria that evaluate their functionality and ability to support the different stages of tabular composition.

1.3.1 The development of electronic tabular composition

Tables are indispensable objects and the evolution of electronic tabular formatting is closely associated with the development of computer-aided typesetting [Fur82]. The use of the computer for document typesetting began in the 1960's. The earliest discussions of computer-composition systems are Barnett's *Computer Typesetting* [Bar65], Stevens's *Automatic Typographic-Quality Typesetting Techniques* [SL67], and Phillips's *Computer Peripherals and Typesetting* [Phi68]. All of the early document formatting systems accepted a stream of text characters interleaved with action codes and produced very simple layouts. Some of them did not even deal with page layout but only produced typeset galleys to be pasted-up manually in the traditional way. Early efforts in tabular typesetting used special programs that performed calculations over numerical data and generated tables in a single format. The pioneering system in style specification was TABPRINT [Bar65] developed by Barnett at MIT in the early 1960's. Typographic styles for each table preceded the data and provided basic formatting choices.

A significant evolution of document formatters occurred when formatting commands were embedded in documents to govern the presentation of the logical content of the documents. The document formatting systems at this stage, such as `troff` with `me` and `ms` macros [Oss76], Scribe [Rei80] and TeX with LaTeX macros [Knu84, Lam85], compile a document with embedded formatting tags and generate formatted documents, possibly accompanied by some error and warning messages. These systems separate the document structure from the document style and enable users who lack the skills of document design to produce high-quality documents in multiple presentational layouts. They describe tables as row structures and provide more styles for tabular formatting, including vertical and horizontal alignment options for text, different types of rules and spanning specification. These systems do not capture the logical structure of tables and they treat rows and columns differently. Moreover, the available document styles provide little support for the achievement of a consistent appearance for tables.

NLS [EE68], the first interactive document composition systems, introduced the notion of WYSIWYG (what you see is what you get) during the late 1960s. Subsequently, a number of integrated document composition systems, including Etude [Ils80], Janus [Cea82], Tioga [Tei84], Furuta's system [Fur86], and Grif [QV86], were developed to provide a WYSIWYG environment for editing and formatting structured documents. These systems allow users to view and manipulate documents through a visual interface and integrate multiple objects into a uniform representation. A WYSIWYG environment is especially suitable for tabular editing and formatting because tabular items are organized simultaneously in two dimensions (rows and columns) and the logical relationships among the items are presented through their relative positions in two dimensions. By modeling tables as two-dimensional row–column structures, these interactive composition systems can manipulate rows and columns equally well and select styles in a more direct way for both rows and columns. Yet, these systems still do not capture the logical structure of tables.

Although the separation of the logical and layout structures of documents has been widely used, there was no distinction between the logical and layout structures of tables until Improv [Imp91], a commercial spread sheet system, was introduced. At about the same time, Vanoirbeek adopted a new tabular model, in Grif [QV86], that specifies the logical structure of tables [Van92]. Both systems maintain the logical relationships among tabular items, provide the ability to arrange these interrelated items easily in two dimensions, and allow users to manipulate tables based on their logical structure. These two systems, however, are weak in the manipulation of tabular logical structure and provide insufficient styles to govern the presentation of logical components. More recently, a tabular formatting system called TAFEL MUSIK [SKS94, SSK94] was designed to specify the logical structure and typographic styles using database schemas and techniques. This system does not appear to support tabular editing.

1.3.2 Some tabular composition systems

We introduce only systems that are representative of the different approaches to tabular processing at different development periods.

TABPRINT

TABPRINT [Bar65] was developed at MIT in the early 1960s. It dealt with numerical data punched on cards or written on magnetic tape in a fixed format and generated formatted output. The input consisted of three parts: the typographic specification, the heading section, and the data section. The typographic specification gave the general style for the whole table, including type face, point size, and line spacing. The heading section described the column headings and their alignment options. The data section specifies row by row.

Tbl

Tbl [Les79] is a preprocessor for the batch-oriented document formatting system `troff` [Oss76]. It processes the table definitions and generates the formatting commands for `troff`. Tables are defined in three sections: options, format, and data. The option section gives the global parameters for the whole table, such as the rule types for the table frame, the alignment options for the whole table, and the delimiters for data items. The format section specifies the formatting attributes for each column, including type faces and sizes, column widths, column separation space, vertical rule types, alignment options, and horizontal spanning headings. The data section specifies the entries row by row. The entries can be strings of characters, `troff` commands, horizontal rule types, vertical spanning headings, and text blocks. Tbl is capable of determining the heights of rows and widths of columns based on the text placed in them, but users have to give either the line-breaking points or the width of text for `troff` to do the line breaking.

LaTeX

LaTeX [Lam85] is a document preparation system based on TeX, a procedural formatting system [Knu84]. The system is based on the concept of structured document design. Users specify documents by their logical components, which are actually TeXmacro definitions. Tables are specified with the tabular environment and the array environment. The first environment is designed for common text tables, and the second one is for tables that contain mathematical equations. These environments allow users to specify the border line style, the justification of each column, and the data as rows that consist of a list of entries mixed with additional formatting information. Like Tbl, LaTeX can also determine the heights of rows and widths of columns based on the text, provided that either the line-breaking points or the width of text are given in advance.

Tabular mark up in SGML

SGML [Int86], the Standard Generalized Markup Language, is an ISO standard that provides a syntactic meta-language for the definition of textual markup systems, which are then used to indicate the logical structures of documents. Each markup system, specified by a context-free grammar, defines the structure and rules for marking up the document instances. The marked-up document instances can be formatted by compiling the mark up into the mark up for a formatting system such as LaTeX, can be interchanged across a heterogeneous network, or can be added to a database system. When translating a marked-up document for a formatting system, the typographic description of how to present documents is usually supplied in a style sheet, which is a collection of styles that may be attached to part or all elements of a document. A specific tabular markup method has been designed as an application of SGML [Int88]. Using this method, a table is specified by four components: a heading, a body, a caption, and an optional description. The table heading specifies only the hierarchical structure of the column headings, which can be divided into four levels of subheadings. The table body is a list of rows, and each row is a list of entries for the columns. The table caption and description are text. With a second SGML tabular markup method [Int92], a table is modeled by a four-level hierarchy: the first level is the whole table; the second level may contain a head that specifies the column headings, a foot that specifies the footnotes, and a body that specifies the entries; the third level consists of rows, and the fourth level consists of cells. The formatting attributes can be specified with different levels of objects, including type sizes, size constraints, cell arrangement, alignment options, and rule types.

TABLE

TABLE [BEF84] is a prototype interactive editor that provides a uniform editing environment and true integration for a variety of dissimilar objects (specifically text objects and table objects) in a WYSIWYG environment. All document objects are represented as an object-oriented architecture and the operations upon different objects are determined by the nature of the objects. Each object in the hierarchy has its own variables and operations. Subobjects can inherit variables and operations from their superobjects. TABLE describes a table with a dual-hierarchical structure (row hierarchy and column hierarchy). Tables are manipulated using an object-oriented mode as follows. An object can be activated, the levels of granularity can be changed, from the granularity of a whole table to the granularity of a single character, and a different logical object can be selected in the current granularity. The currently active operations are determined by the nature of the active object.

Spreadsheet systems

Lotus 1-2-3 [Lot84] and Microsoft Excel [MS-90] are sophisticated spreadsheet systems, that provide automated business tools for the manipulation, computation, and analysis of data as well as providing presentational tools for reporting results in different formats. Tabular data is put in a worksheet, a two-dimensional lattice that can be addressed by row and column indices. Formatting attributes can be assigned to any data cell. A part from lattice formats, tabular data can be presented in different forms, such as bar graphs, pie graphs, and line graphs.

Beach's system

Richard Beach [Bea85] presented a framework for formatting tables that is suitable for use in interactive editors and formatters. A central idea in his approach is the separation of the table arrangement from the table layout. The table arrangement, or *table topology*, is expressed by a grid structure. Geometric constraints are expressed as linear inequalities in which the independent variables are the positions of the grid lines and the alignment points of table entries. The table layout, or *table geometry*, is computed from both the table topology and the physical dimensions of the table entries. A linear-inequality-constraint solver is used to compute the table geometry. He implemented sophisticated algorithms to manipulate and render tables based on the grid structure. All editing objects, including the whole table, a row, a column, an entry, and a rule, are organized with an object-oriented architecture, and style attributes can be specified for each of them. The style options include alignment options, rule parameters, or bearoff distances. A subobject may inherit the style attributes of a superobject. For example, an entry may either have its own style attributes or inherit the style attributes of its row and column, or of the whole table.

Furuta's prototype

Richard Furuta [Fur86] developed an integrated editor-formatter that merges the flexibility of document representation using an the abstract object-oriented approach with the naturalness of document manipulation using the exact-representation editor-formatters. Documents are represented by a heterogeneous structure: *tnt* (strict tree — not strict tree). The top level of a *tnt* is a strict tree, and the leaves of the strict tree are tree blocks with arbitrary structure, which are used to represent nonhierarchical objects (for example, tables and mathematical equations). Tree blocks can contain objects that are *tnt* structures. A table block is modeled with a variety of dual-hierarchy structures. The *tnt* structure allows table entries to be different kinds of objects such as text, equations, and subtables. Furuta's prototype provides operations to manipulate a table's row–column structure, to edit the contents of entries, and to span entries horizontally and vertically.

Cameron's system

John P. Cameron [Cam89] presented a cognitive model for tabular editing. The model is an extension of the model presented by Beach. The goal of the model is to propose a group of functions which allow table designers to manipulate both the topological structure and the content of a table in a natural manner to give a visual, interactive environment. To provide the operations that are involved in the mental process of making a table, Cameron introduced two distinguishing concepts: *region* and *section*. A region of a table is an area of the table that is obtained by slicing completely through the table with either two parallel horizontal lines or two parallel vertical lines. A section is any group of cells in a rectangular box. Cameron's system breaks down the mental process underlying tabular construction into three steps: structure editing, content editing, and visual editing. Structure editing consists of the creation or modification of the topological structure of a table. The operations in this process are: splitting and joining cells, and inserting, deleting, duplicating and moving a region. Cameron also mentions that more complex operations such as rearranging the label region (reversing the hierarchy of index items and their subindex items in a label region) and transposing a table can also be added to his system, but it would increase the complexity of the system. Content editing consists of the activities involved in entering, deleting, and modifying the individual entries in a table. Since the entries can be of various types, such as numeric, textual, mathematical, graphical, or even tabular, different operations are required to support each of these activities. Visual editing consists of modifying the visual format of the entries in a section of a table. The allowable modifications are: type faces, alignment options, background shading and colors, and the types of border rules.

Improv

Improv [Imp91] is an improved version of Lotus 1-2-3. It is an interactive commercial system for the editing and formatting of tabular data for finance and business. Tables are defined by specifying multiple categories in both the horizontal and vertical dimensions of a spreadsheet. The labels of these categories are placed at the top or on the left side of the spreadsheet. Entries are placed in cells that are addressed by the labels of different categories. Besides inheriting the functions provided by Lotus 1-2-3, Improv also provides some operations to manipulate tables logically. For example, tables can be topologically rearranged by moving a category from the horizontal dimension to the vertical dimension, and conversely.

Vanoirbeek's system

In Vanoirbeek's system [Van92], a table is specified as a collection of entries that are semantically connected to multiple labels of different categories. The logical structure of a table is modeled by a tree with additional edges: a table consists of a set of logical dimensions (categories) and a set of items (entries); the logical dimensions include rubrics (labels) which may themselves contain subrubrics; additional edges are used to represent the connections between items and rubrics. The main reason for this representation mechanism is to comply with the hierarchical document representation used in the host system Grif [QV86]. Vanoirbeek breaks table creation into two processes: editing and formatting. Editing includes structure editing and content editing. When editing the structure, one can add or suppress dimensions, rubrics, and subrubrics, and also merge items. When editing the content, one can use classical text editing functions to edit the names of dimensions, rubrics, subrubrics, and the content of items. Formatting associates the values of typographic attributes with the tabular components. The typographic attributes include the presentational options that control the geometric arrangement of the table and formatting options for data, rules, and decoration. The typographic attributes can be specified in a generic way by a set of presentational rules. Each presentational rule is related to an attribute, and it specifies how the value must be calculated during formatting. Presentational rules allow the propagation and synthesis of attribute values in a tree structure to achieve consistent typographic choices throughout the document.

TAFEL MUSIK

TAFEL MUSIK [SKS94, SSK94] borrows database techniques to handle various aspects of tabular processing. It provides a data model to represent a homogeneous class of tabular logical structures and supports a *tabular style description language* (TSDL) to specify styles for tabular logical structures. A TSDL interpreter applies the styles to a tabular logical structure retrieved from the database and generates the final tabular layout. The details of the data model and TSDL are not yet known since the paper [SSK94] about them is still in preparation. The authors do, however, describe an algorithm that attempts automatic formatting and high-quality layout has been described [SKS94]. The algorithm automatically determines the physical dimensions of the rows and columns and breaks text into lines according to the widths of the columns. Moreover, the algorithm generates a layout that satisfies some objective function (for example, minimal area, minimal diameter, and minimal white space) and satisfies all the user-specified size constraints expressed as linear inequalities. The algorithm divides the entire optimization problem into a number of subproblems, and it uses accelerating techniques to increase efficiency.

1.3.3 Evaluation of prior work

We evaluate tabular composition systems based on the following criteria, which we believe can be used to indicate whether they provide sufficient functionality to support the different stages of tabular composition:

1. Does it specify the multi-dimensional logical structure of tables and provide sufficient functionality to manipulate the logical structure?

2. Does it specify the topological arrangement of tables and provide the ability to arrange tabular items flexibly in both the horizontal and vertical dimensions?

3. Does it specify sufficient styles for different kinds of tabular components to achieve high-quality layout?

4. Does it help users to deal with different table sizes and shapes?

The early system, TABPRINT, provides little functionality to support tabular composition. It can generate tables according to only limited formatting styles that control the presentation of the whole table. Variant styles for different items are not allowed.

Using Tbl and LaTeX, users specify tables explicitly based on the topological arrangement; thus, there is no clear separation between the logical structure and the topology. Tbl and LaTeX are specification languages that rely on the underlying formatting systems `troff` and TeX, respectively. These formatting systems do not provide true two-dimensional formatting. Table specifications are precompiled and tabular items are broken down into two separate formatting processes: a horizontal formatting process followed by a vertical formatting process. Because Tbl and LaTeX do not provide editing facilities, users have to respecify tables if they want to change the topological arrangement of these tables. Tbl provides many typographic styles, but the styles for columns and rows are treated differently. LaTeX provides styles for only columns. It is difficult for users to specify tables that require complex layouts, such as the cut-in style and grouping items in a number of rows with white space and rules. Tbl and LaTeX can break text into lines if the text width is given and the tabular markup mechanism specifies size constraints for both rows and columns. Such functionality can help users to control table size and shape to a limited extent.

The tabular markup methods using SGML are not tabular composition systems. They specify explicitly the topological arrangement and do not separate the logical structure from the topology. Like Tbl and LaTeX, they require the respecification of a table if its topological arrangement needs to be changed. The latest tabular markup method in SGML provides many styles for the whole table, the column headings, the rows, and the cells, but it provides no styles for the columns except an alignment option. It also allows size constraints for both rows and columns.

TABLE, spreadsheet systems, Beach's system, Furuta's prototype, and Cameron's system, which are all interactive, also describe tables based on their topological arrangement and do not separate the logical structure from the topology. These systems provide true two-dimensional formatting and treat rows and columns equally. Although these systems provide a WYSIWYG environment for editing the topological arrangement of a table, users may need many editing operations to rearrange tabular items. For example, if we want to change the topological arrangement of a table by exchanging the labels in the stub and the boxhead, we have to use many moving operations. These systems are able to specify typographic styles interactively for the whole table, columns, rows, blocks, and cells. Beach's system can automatically calculate the heights of rows and widths of columns that satisfy a set of size constraints expressed as linear inequalities and achieve the minimal value for the sum of the tabular width and height. It assumes, however, that the text is broken into lines in advance, which enables the system to find a layout in polynomial time.

Improv and Vanoirbeek's system are able to specify the multi-dimensional logical structures of tables. Neither of them, however, provides sufficient ability to modify the logical structure of a table. Both systems offer only the basic functions to create a new logical structure interactively. Some changes to an existing structure, such as the change from an implicit structure to an explicit one, may require the user to abandon the old structure and create a new one. Both systems provide the ability to edit the topological arrangement by changing the position of a category inside the stub or the boxhead and by moving a category from the stub to the boxhead, and conversely. This ability also helps users to control tabular size and shape. Although Improv captures the tabular logical structure, it provides few typographic styles to control the presentation of logical components. Vanoirbeek's system does provide some basic typographic styles for categories, labels and entries, but it provides no typographic styles for complex tabular layouts. Both Improv and Vanoirbeek's system can control the sizes and shapes by specifying the widths and heights of tables or the sizes of columns and rows. Yet they do not deal with automatic line-breaking and size constraints during the calculation of the physical dimensions of a table.

TAFEL MUSIK is a tabular typesetting system that is currently under development and as such little is known about it. It also describes tables based on their multi-dimensional logical structure. It is, however, a batch-oriented formatting system and provides no editing ability to update the logical structures of tables. TAFEL MUSIK provides a powerful ability to control tabular size and shape by allowing users to specify size constraints as linear inequalities, automatically breaking text into lines, and calculating the optimal layouts for a small number of objective functions.

1.4 Research Objectives

The general goal of our research is to create a tabular model for the design of high-quality tables in two dimensions. It should support the different stages of tabular composition, including the design of the logical structure, the arrangement of tabular items, the specification of typographic styles, and the formatting of concrete tables. This model should be based only on the nature of tables and should be independent of any existing formatting and editing systems. The specific objectives are:

1. To propose an abstract model to specify tabular logical structure

   Like Vanoirbeek's system and TAFEL MUSIK, our abstract model should also describe the multi-dimensional logical structure of tables. The major difference between our abstract model and theirs should be the representation used to specify the logical structure. Vanoirbeek's system abstracts tables as a tree with additional edges to comply with the hierarchical structure used in its host system Grif. TAFEL MUSIK uses a two-dimensional database model to specify multi-dimensional tables. Neither the hierarchical structure nor the database model naturally describe the characteristics of multi-dimensional tables. Our abstract model should use well-understood mathematical notation to abstract tables and should hide the representation and implementation.

2. To investigate what operations are needed for the manipulation of abstract tables.

   The editing operations for row-column structures have been investigated. The operations for multi-dimensional logical structures, however, have been little investigated. The editing model should manipulate tables at an abstract level. The operations in the editing model should be topology independent.

3. To explore what topological and style rules are necessary to specify a tabular layout structure.

   We divide layout specification into two parts: topological specification and style specification. In topological specification, we should focus on the rules needed to specify the relative placement of tabular items in two dimensions. In style specification, we should provide style rules that govern the presentation of the whole table, the main regions (the stub, boxhead, stub head, and body), the logical components (categories, labels, and entries), and the layout components (rows, columns, and blocks). These style rules should include not only basic formatting attributes, such as the type face and point size, spacing and rule type, horizontal and vertical alignment options, and size constraints, but also formatting attributes that enable us to easily specify complex layout structures, such as grouping items, cut-in headings, and spanning options for entries.

4. To solve the formatting problem arising in the generation of a concrete table when applying layout specifications to an abstract table.

   The most difficult problem arising in tabular formatting is how to determine efficiently the physical dimensions of a table that satisfies user-specified size constraints. Beach [Bea85] has given a polynomial-time algorithm that requires users to indicate the line breaks in advance. TAFEL MUSIK's developers have designed an exponential-time algorithm that achieves automatic line-breaking and satisfies one of a small number of objective functions. Automatic line-breaking and size constraints are important features that can help users to deal with table size and shape. Our objectives are to analyze the computational complexity of tabular formatting with respect to different restrictions and to design an algorithm that supports automatic line-breaking and size constraints expressed as linear inequalities and finds the physical dimensions in polynomial time for many tables.

5. To demonstrate that our model is feasible by implementing a prototype tabular composition system that helps users to efficiently design high-quality tables.

Based on our tabular model, we should implement a prototype tabular editor and formatter. This prototype should provide an interactive interface to help users easily specify and manipulate logical structure, topological arrangement, and typographic styles. It should also generate formatted tabular outputs for different typesetting systems.

1.5 Contributions

The contributions of this thesis can be summarized from five aspects. First, we propose an abstract model to specify the multi-dimensional logical structure of tables. This model uses well-understood mathematical notation, such as sets and functions, to abstract tables, which distinguishes our model from other models. This model not only precisely abstracts the category structure and the logical associations between labels and entries but also allows us to determine what operations are necessary for the manipulation of the multi-dimensional logical structure.

Second, we present an editing model for the manipulation of the multi-dimensional logical structure of tables. The editing model enables us to edit tables independently of their topology. We no longer need to perform a transformation between logical components and layout components when editing tabular logical structure. As far as we know, no one has explored what operations are needed for multi-dimensional tables at such an abstract level.

Third, we give a presentational model for the specification of tabular layout structure. We adopt a similar arrangement of the categories to those used in Improv and in Vanoirbeek's system, but offer more options to arrange labels. The style rules provided in the presentational model can be used to specify style from different viewpoints. In addition to the traditional style for the row-column structure, we provide style rules to specify the style for the logical components of an abstract table and for the four major regions of a table: stub, boxhead, stub head and the body. To our knowledge, no current system offers such an abundance of style rules for tabular presentation. Moreover, we also propose an approach to solve style conflicts when applying a set of styles rules to a table.

Fourth, we are the first to prove that the tabular formatting is NP-complete with respect to two useful features: automatic line breaking and size constraints expressed as linear inequalities. We also design a polynomial-time greedy algorithm that can partially solve the tabular formatting problem for many tables.

Lastly, we implemented a prototype to validate our ideas and demonstrate that we can integrate our models in an interactive tabular editor and formatter. This prototype not only helps users to easily design high-quality tables in two dimensions, but also offers users a tool to analyze and explore tabular data efficiently.

This thesis describes our proposals, discussion and investigations, and it presents possible future work based on our current achievements. In Chapter 2, we present a formal model for the abstraction of tabular logical structure. In Chapter 3, we discuss the operations for the manipulation of tabular logical structure. In Chapter 4, we describe what presentational rules are necessary for topological specification and style specification. In Chapter 5, we formally define the tabular formatting problem and prove its NP-completeness. We also give an algorithm that solves the problem in polynomial time for many common cases. In Chapter 6, we introduce a prototype interactive tabular editor and formatter which is based on the tabular model and describes how we solve some key problems arising during the implementation of the prototype. In the last chapter, we draw some conclusions about current achievements and discuss what remains to be done.

Chapter 2

Abstraction

Tabular abstraction plays an important role in tabular composition because it determines the capabilities of editing and presentation. When we design a table, we usually decide on the logical structure before we select a presentational form. Thus, we should deal with the logical structure and the layout structure separately. There are at least two advantages with the separation the logical structure and the layout structure. First, tables can be manipulated independently of their layout structure. For example, to remove a label from a category, we no longer have to determine which rows or columns should be removed from the layout structure. Second, by associating different topologies and styles with the logical structure, we easily can obtain various layout structures for a table. For example, to obtain the transposition of a table, we need to respecify only the topology of a table. We now present an abstract model that specifies only the logical structure of tables and ignores their layout structures. This model is based on our preliminary model [WW93]. We describe an editing model and a presentational model, which are based on the abstract model, in Chapters 3 and 4, respectively.

2.1 Guidelines for tabular abstraction

We propose three guidelines for the design of a tabular abstract model. We base the model on observation of tables in the literature [CRC88, Sta86, BR74, Rit86] and on the discussions of tables from the perspectives of typography [Chi93, Rub88, Wil83], psychology [WF70, Wri68, Wri77, Tin60, SW84], and statistics [Zei85, Ehr77, Hal43].

First, the model should capture a wide range of tables. We do not expect to provide an approach that models the logical structures of all tables. There are some tables that have a complex logical structure (see Section 2.4). We focus our efforts on the most common kinds of tables, with one simplifying assumption: we ignore footnotes. We examined tables in books from various sources, including typography, statistics, sociology, science, and business, and found that majority of tables can be specified with a multi-dimensional logical structure (see Table A.1 in Appendix A). A table with multi-dimensional logical structure consists of a number of categories and a set of entries. The labels in each category are organized hierarchically and each entry is logically associated with exactly one label from each category.

Second, the model should not include any characteristic that is related to the presentational form of a table. Any concept that is associated with tabular topology (such as row or column) or typography (such as typeface or rule type) should not appear in the model.

Third, the model should abstract tabular logical structure with well-understood mathematical notions, rather than with a specific representational scheme. In this way, we can view an abstract table as an abstract data type that hides its representation and implementation. We can also use this model to define the semantics of the editing model described in Chapter 3.

2.2 Terminology

We specify the logical structure of a table as an *abstract table*, which describes the hierarchical label structure of categories and the logical relationships between labels and entries. We first define some necessary terminology and notation before we define an abstract table.

Labels

A *label* can be any string of characters and symbols, including the empty string.

Labeled sets

A *labeled set* is a set together with a label. We specify a labeled set as an ordered pair $(label, set)$. For example, $(1991, \emptyset)$ and $(Grade, \{50, 60\})$ are labeled sets.

Labeled domains

A *labeled domain* is defined inductively as follows:

1. A labeled empty set $(L, \emptyset)$ is a labeled domain.
2. A labeled set of labeled domains such that the labels of the labeled domains are pairwise distinct is a labeled domain.
3. Only labeled domains that are obtained from rules 1 and 2 are legal.

For a labeled domain $D = (l, s)$, we use $lbl(D)$ to denote the label l and $set(D)$ to denote the set s. A labeled domain can be represented by a labeled tree in which the children of a node are unordered. Fig. 2.1 presents the relationship between a labeled domain and its labeled tree. Each node in the tree represents a labeled domain and each external node represents a labeled empty set. For convenience, we will use the tree of a labeled domain to explain some concepts and operations that are related to labeled domains. It should be clear that we can use labeled domains to describe the hierarchical label structures of categories.

Label sequences

We use *label sequences* to uniquely identify the labeled subdomains in a labeled domain. For a labeled domain $D = (l, s)$, the label l is a label sequence that identifies D. We extend this notion inductively for the labeled subdomains in D as follows. If

```
(D1, { (d11, { (d111, phi),
               (d112, phi),
             } ),
       (d12, phi),
       (d13, phi),
     } )
```

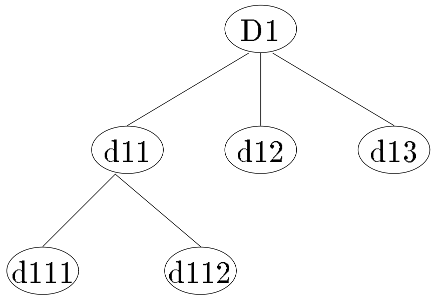


Figure 2.1: The relationship between a labeled domain and a corresponding labeled tree.

a labeled sequence l identifies a labeled domain which contains a set of labeled domains $\{(l_1, s_1), \ldots, (l_r, s_r)\}$, then $l.l_i$ is a label sequence that identifies labeled subdomain (l_i, s_i). For example, the label sequence $D1.d11$ identifies the labeled subdomain $(d11, \{d111, \emptyset\}, \{d112, \emptyset\})$ in the labeled domain in Fig. 2.1. The dot notation that we use is well known in library classification systems and it is often called Dewey notation. An explicit dot is used to separate the labels in a label sequence to avoid ambiguity. Given a label sequence l, we use $\Lambda(l)$ to denote the labeled domain identified by l. We also use $lbl(l)$ and $set(l)$ to denote the label and the set of $\Lambda(l)$.

Frontier label sequences

A label sequence that identifies a labeled domain with an empty set is a *frontier label sequence*. In the associated labeled tree, such a label sequence corresponds to a root-to-frontier path. The frontier $fr(D)$ of a labeled domain D is the set of all frontier label sequences of D and for a set C of labeled domains, $fr(C) = \{fr(D) | D \in C\}$. Given a labeled domain D, $fr(D)$ is unique and, moreover, given $fr(D)$, we can reconstruct a unique D. Thus, given a set S of label sequences that satisfies the following two conditions:

1. All label sequences in S have a common first label,
2. S is prefix-free; that is, whenever a label sequence $x.y$ is in S, for some label sequences x and y, the label sequence x is not in S,

we can construct a labeled domain D such that $fr(D) = S$. If we can construct a labeled domain D from a set S of label sequences such that $S = fr(D)$, then S is *consistent.* For example, $S = \{D1.d11.d111, D1.d11.d112, D1.d12, D1.d13\}$ is consistent because S is the frontier of the labeled domain in Fig. 2.1. $S = \{D1.d11, D1.d11.d111\}$ is not consistent because we cannot construct a labeled domain such that S is the frontier of the labeled domain.

Unordered Cartesian product

Given $n \geq 1$ disjoint sets $A_1, A_2, \ldots, A_n$ their unordered Cartesian product $A_1 \otimes \ldots \otimes A_n$ is a set A such that each element of A is a set that contains exactly one element from each of the sets $A_i (1 \leq i \leq n)$. We use the unordered Cartesian product to associate frontier label sequences with entries in an abstract table. When we have n disjoint labeled domains $D_1, D_2, \ldots, D_n$, we need the unordered Cartesian product of their sets of frontier label sequences, namely $fr(D_1) \otimes \ldots \otimes fr(D_n)$. For a set C of labeled domains $D_1, \ldots, D_n$, we use $\otimes fr(C)$ to denote $fr(D_1) \otimes \ldots \otimes fr(D_n)$.

2.3 The definition of an abstract table

Now we are ready to define an abstract table. An abstract table is specified by an ordered pair (C, δ), where

1. C is a finite set of labeled domains.
2. δ is a map from $\otimes fr(C)$ to the universe of possible value.

We have introduced labeled domains to model the informal notion of a category; thus, we now treat a category as a labeled domain and C as a finite set of categories. We use $\otimes fr(C)$ to model the entry set of a table. Each entry is identified by a $|C|$-element set in $\otimes fr(C)$ and is assigned a value by δ. If δ assigns no value for an entry $\{f_1, f_2, \ldots, f_{|C|}\}$, we say that entry $\{f_1, f_2, \ldots, f_{|C|}\}$ is *undefined.* We use the word "frame" to denote a table in which the map δ is empty, or is considered to be empty; thus, $\otimes fr(C)$ is also called the *frame* of a table. There are two important quantitative measures of a table:

Table 2.1: The average marks for 1991–1992.

Year	Term	Mark					
		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

its dimension and size. The dimension *dim(T)* of an abstract table $T = (C, \delta)$ is the size of C, the number of categories in C. On the other hand, the size *size(T)* of an abstract table $T = (C, \delta)$ is the size of $\otimes fr(C)$, the number of entries in T. Using this model, we can specify the logical structure of Table 2.1 with the abstract table $T = (C, \delta)$ in which C consists of following three categories:

$(Year, \{(1991, \emptyset), (1992, \emptyset)\})$,
$(Term, \{(Winter, \emptyset), (Spring, \emptyset), (Fall, \emptyset)\})$, and
$(Mark,\ \{(Assignments,\ \{(Ass1, \emptyset),\ (Ass2, \emptyset),\ (Ass3, \emptyset)\}),$
$\quad (Examinations,\ \{(Midterm, \emptyset),\ (Final, \emptyset)\}),$
$\quad (Grade,\ \emptyset)\})$.

and δ is defined by:

$\delta(\{Year.1991,\ Term.Winter,\ Mark.Assignments.Ass1\}) = 85;$
$\delta(\{Year.1991,\ Term.Winter,\ Mark.Assignments.Ass2\}) = 80;$
$\delta(\{Year.1991,\ Term.Winter,\ Mark.Assignments.Ass3\}) = 75;$

$\delta(\{Year.1991,\ Term.Winter,\ Mark.Examinations.Midterm\}) = 60;$
$\delta(\{Year.1991,\ Term.Winter,\ Mark.Examinations.Final\}) = 75;$
$\delta(\{Year.1991,\ Term.Winter,\ Mark.Grade\}) = 75;$
$\delta(\{Year.1991,\ Term.Spring,\ Mark.Assignments.Ass1\}) = 80;$
$\delta(\{Year.1991,\ Term.Spring,\ Mark.Assignments.Ass2\}) = 65;$
$\delta(\{Year.1991,\ Term.Spring,\ Mark.Assignments.Ass3\}) = 75;$
$\delta(\{Year.1991,\ Term.Spring,\ Mark.Examinations.Midterm\}) = 60;$
$\delta(\{Year.1991,\ Term.Spring,\ Mark.Examinations.Final\}) = 70;$
$\delta(\{Year.1991,\ Term.Spring,\ Mark.Grade\}) = 70;$
$\delta(\{Year.1991,\ Term.Fall,\ Mark.Assignments.Ass1\}) = 80;$
$\delta(\{Year.1991,\ Term.Fall,\ Mark.Assignments.Ass2\}) = 85;$
$\delta(\{Year.1991,\ Term.Fall,\ Mark.Assignments.Ass3\}) = 75;$
$\delta(\{Year.1991,\ Term.Fall,\ Mark.Examinations.Midterm\}) = 55;$
$\delta(\{Year.1991,\ Term.Fall,\ Mark.Examinations.Final\}) = 80;$
$\delta(\{Year.1991,\ Term.Fall,\ Mark.Grade\}) = 75;$
$\delta(\{Year.1992,\ Term.Winter,\ Mark.Assignments.Ass1\}) = 85;$
$\delta(\{Year.1992,\ Term.Winter,\ Mark.Assignments.Ass2\}) = 80;$
$\delta(\{Year.1992,\ Term.Winter,\ Mark.Assignments.Ass3\}) = 70;$
$\delta(\{Year.1992,\ Term.Winter,\ Mark.Examinations.Midterm\}) = 70;$
$\delta(\{Year.1992,\ Term.Winter,\ Mark.Examinations.Final\}) = 75;$
$\delta(\{Year.1992,\ Term.Winter,\ Mark.Grade\}) = 75;$
$\delta(\{Year.1992,\ Term.Spring,\ Mark.Assignments.Ass1\}) = 80;$
$\delta(\{Year.1992,\ Term.Spring,\ Mark.Assignments.Ass2\}) = 80;$
$\delta(\{Year.1992,\ Term.Spring,\ Mark.Assignments.Ass3\}) = 70;$
$\delta(\{Year.1992,\ Term.Spring,\ Mark.Examinations.Midterm\}) = 70;$
$\delta(\{Year.1992,\ Term.Spring,\ Mark.Examinations.Final\}) = 75;$
$\delta(\{Year.1992,\ Term.Spring,\ Mark.Grade\}) = 75;$
$\delta(\{Year.1992,\ Term.Fall,\ Mark.Assignments.Ass1\}) = 75;$
$\delta(\{Year.1992,\ Term.Fall,\ Mark.Assignments.Ass2\}) = 70;$
$\delta(\{Year.1992,\ Term.Fall,\ Mark.Assignments.Ass3\}) = 65;$
$\delta(\{Year.1992,\ Term.Fall,\ Mark.Examinations.Midterm\}) = 60;$
$\delta(\{Year.1992,\ Term.Fall,\ Mark.Examinations.Final\}) = 80;$
$\delta(\{Year.1992,\ Term.Fall,\ Mark.Grade\}) = 70.$

Since we use sets to specify the category structure of a table, the categories are unordered and the labels in a category or a subcategory are also unordered. Ordering is an issue of topology, and we do not include it in the abstract model. We will deal with category ordering and label ordering in Chapter 4.

The definition of an abstract table fulfills our three guidelines; that is, it can be used to specify the logical structures of commonly used tables, it is independent of tabular topology and typography, and it uses sets and mappings, which are well-understood mathematical notions. In the next chapter, we will use this model to specify the semantics of the tabular editing operations.

2.4 Expressiveness of the abstract model

We have made the simplying assumption that we do not model footnotes in the abstract model. Clearly, footnotes play an important role in tables. See the examples in the book *Human Activity and Environment* [Sta86]. In this book, 148 of the 172 tables have footnotes (see Table A.1, Appendix A). Although we do not model footnotes, a user can still use footnotes with any tabular entry. The limitation is that they are dealt with by the target typesetting system, they are not manipulable as abstract objects within our model.

Second, the abstract model does not capture all tables even when we ignore footnotes. The model can be used to specify tables that have only a multi-dimensional logical structure. Not all tables have such a nice structure however. Some tables are a combination of several tables as a multi-dimensional structure. For example, Table 2.2 is a combination of two tables as a multi-dimensional structure. There are two categories: **Barometer reading** and **Temp. alt. factor** in this table. The entries of the table are divided into two groups. One group, including all the entries above the double line, are associated with only partial labels in the category **Temp. alt. factor** and the partial labels in the **Barometer reading**. The other group, including the entries below the double line, are also associated with partial labels in the two categories. We can break the category **Barometer reading** into two categories in this way: **Barometer reading 1** includes the labels above the double line and **Barometer reading 2** includes the labels below the double line. Similarly, we can also break the category **Temp. alt. factor** into two categories: **Temp. alt. factor 1** and **Temp. alt. factor 2**. Then, we obtain two tables that can be specified as a multi-dimensional structure. Another example, Table 2.3, is a combination of three tables in multi-dimensional structure. There are three categories: **X**, **Y**, and **Type of calculations** (the category in the stub head) in this table. The first subtable, whose entries are associated with the categories **X** and **Type of calculations**, is placed in the boxhead. The second subtable, whose entries are associated with the categories **Y** and **Type of calculations**, is placed in the stub. The third subtable, whose entries are associated with categories **X** and **Y**, is placed in the body. To specify these tables, we should be able to specify multiple mappings that can share some categories in an abstract table. This is a topic of future investigation. We also need to investigate how to present these kinds of abstract tables in two dimensions.

We carried out an experiment to measure how well our abstract model specifies tables in the real world. We counted tables in books from various sources, including statistics, sociology, science, and business. The results of the experiment, given in Table A.1, Appendix A, reveals that the abstract 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 a multi-dimensional logical structure.

Table 2.2: Metric units.

Temperature-altitude factor	Barometer reading					
	780 mm	760 mm	740 mm	720 mm	700 mm	
1	0.9	0.9	0.9	0.8	0.8	
5	4.5	4.4	4.3	4.2	4.0	
10	9.0	8.8	8.6	8.3	8.1	
15	13.0	13.2	12.9	12.5	12.2	
20	18.2	17.7	17.2	16.8	16.3	
25	22.8	22.2	21.6	21.0	20.4	
30	27.4	26.7	26.0	25.3	24.6	
35	......	31.2	30.4	29.6	28.8	
	760 mm	740 mm	720 mm	700 mm	680 mm	660 mm
40	35.8	34.9	33.9	33.0	32.0	31.1
45	40.4	39.3	38.3	37.2	36.2	35.1
50	45.0	43.8	42.7	41.5	40.3	39.1
55	49.7	48.4	47.1	45.8	44.5	43.1
60	......	52.9	51.5	50.1	48.6	47.2
65	......	57.5	55.9	54.4	52.8	51.2
70	......	62.1	60.4	58.7	57.1	55.4
75	......	66.7	64.9	63.1	61.3	59.5

Table 2.3: 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
Y	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

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

Chapter 3

Editing

Modeling a table as a row–column structure requires users to perform a transformation from logical components to layout components when editing the logical structure of the table. For example, if we want to delete a label from a category, we need to determine the rows or the columns that contain this label and remove these rows or columns. Modeling tables with their logical structure, however, makes editing independent of their topological arrangement. We can manipulate tables at a logical level without worrying about their layout structure. We present an editing model that proposes a set of editing operations for tables. We use the abstract model described in Chapter 2 to specify the logical structure of tables and the semantics of these operations. As we will see in Chapter 6, we use these operations to implement the editor in a prototype tabular composition system.

3.1 What operations are necessary?

We need to be able to create a new table and to manipulate and modify an existing table. The operations should include: changing logical dimension, reorganizing the label structure of categories and updating the entry values and labels. Thus, we divide the operations into three groups.

Table 3.1: The average marks for 1991–1992.

Year	Term	Mark					
		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

3.1.1 Table operations

Table operations may change the dimensions of tables and, therefore, change their frames. The size of a table, however, may or may not be changed by these operations. The basic operations for this group include the creation of an empty table, the addition of a new category, and the deletion of an existing category. More complex operations may be necessary when we take into account some special requirements of editing. Sometimes we need to generate new categories that are based on existing ones. For example, if we want to design a table that shows the flight schedules between the major cities of Canada for an airline company, we can use a two-dimensional table with two categories that consist of the same labels—the cities. It is easier to design such a table if we create one category first and then copy it to make the second category. Thus, we may need an operation to duplicate a category. Other examples of additional operations are: reducing the logical dimension of Table 3.1 by combining categories **Year** and **Term**, or undoing the combination by splitting the combined category into two categories. Thus, we need operations to combine two categories and split one category into two categories.

3.1.2 Category operations

Category operations change the label structure of a category; thus, they preserve the dimension of a table and may change the size of a table. The basic operations for this group include inserting a subcategory into a category, deleting a subcategory from a category, moving a subcategory to a new place within a category, and duplicating a subcategory. Now suppose that we want to design a conversion table from pounds to kilograms for the range of 0 to 99 pounds. We may present the table as an implicit structure shown in Table 3.2, in which the labels of the category **Pounds** are organized as shown in Fig. 3.1(a). Assume that we want to change it to the explicit structure shown in Table 3.3, in which the labels of the category **Pounds** are organized as shown in Fig. 3.1(d). Fig. 3.1 shows an approach to transforming the category structure from an implicit to an explicit structure. It is helpful if we have an operation that can combine two subcategories by appending all the children of a subcategory to the frontier nodes of the other categories and a reverse operation that splits a subcategory into two categories. We also need an operation that promotes a set of subcategories up one level and an operation that demotes a set of subcategories down one level to change the depth of a category.

3.1.3 Label and entry operations

Label and entry operations change only the labels and entry values. The operations in this group are simple but are frequently used. These operations do not change the frame of an abstract table, but affect the content of items in the frame. They preserve both the dimension and the size of a table. The operations include changing a label and assigning a new value for an entry. If we want to support a searching ability, we also need operations that read the entry values. Sometimes we need to compute the value of an entry based on its old value. For example, if the value of an entry is a set of numbers, we may need to change the entry value to be the sum of the numbers. Thus, we need an operation that performs a calculation over an entry value.

Table 3.2: An implicit conversion table from pounds to kilograms.

Pounds	Kilograms
One digit	
0	0.00
1	0.45
2	0.90
3	1.36
4	1.81
5	2.26
6	2.72
7	3.17
8	3.63
9	4.08
Two digits	
00	0.00
10	4.54
20	9.07
30	13.60
40	18.14
50	22.68
60	27.22
70	31.75
80	36.29
90	40.82

Table 3.3: An explicit conversion table from pounds to kilograms.

Pounds	Kilograms
0	0.00
1	0.45
2	0.90
.	.
.	.
.	.
8	3.63
9	4.08
10	4.54
11	4.99
12	5.44
.	.
.	.
.	.
18	8.17
19	8.62
.	.
.	.
.	.
90	40.82
91	41.27
92	41.72
.	.
.	.
.	.
98	44.45
99	44.90

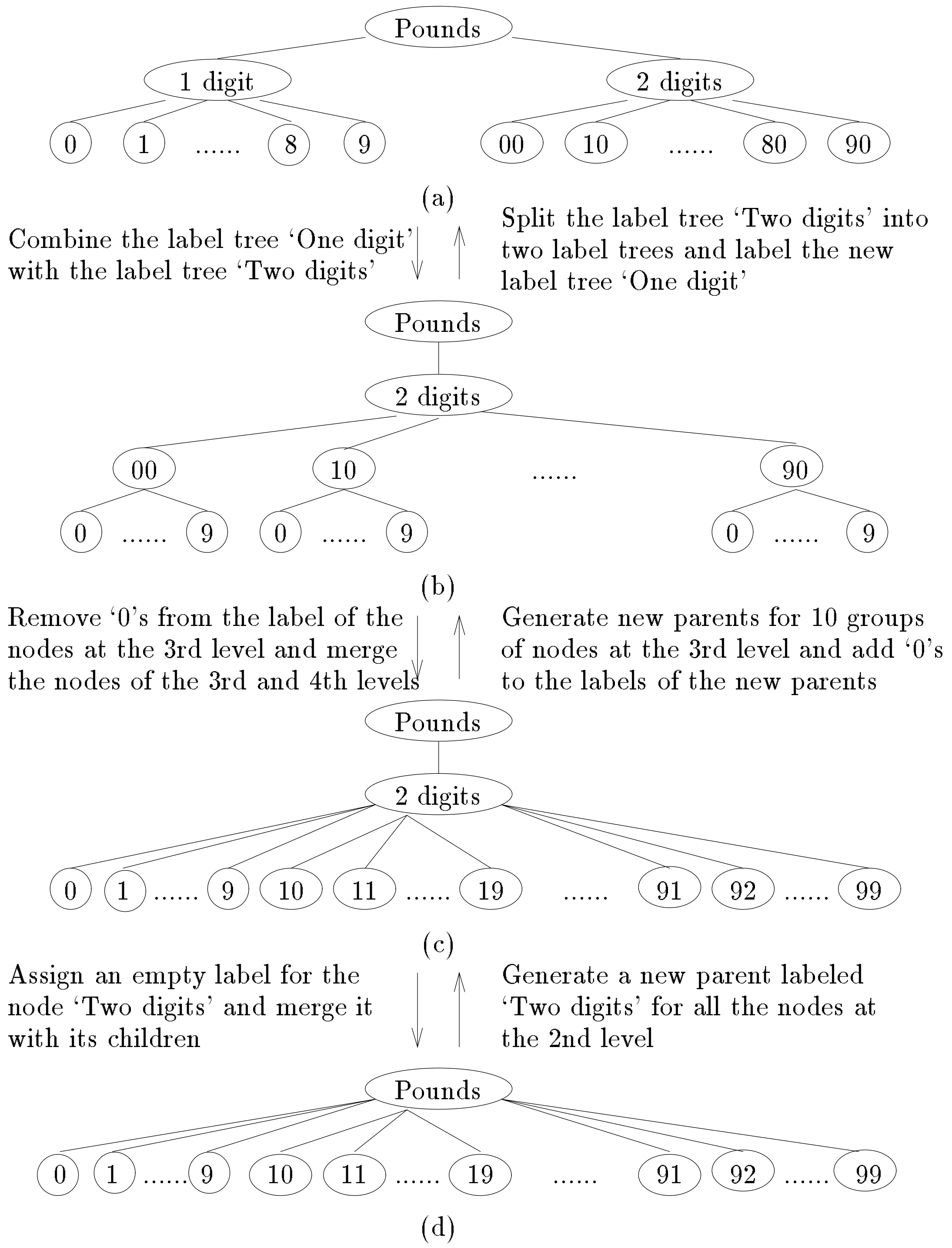

Pounds
1 digit
2 digits
0
1
......
8
9
00
10
80
90
(a)
Combine the label tree 'One digit' with the label tree 'Two digits'
Split the label tree 'Two digits' into two label trees and label the new label tree 'One digit'
(b)
Remove '0's from the label of the nodes at the 3rd level and merge the nodes of the 3rd and 4th levels
Generate new parents for 10 groups of nodes at the 3rd level and add '0's to the labels of the new parents
11
19
91
92
99
(c)
Assign an empty label for the node 'Two digits' and merge it with its children
Generate a new parent labeled 'Two digits' for all the nodes at the 2nd level
(d)

Figure 3.1: The transformations between implicit and explicit structures.

3.2 Applying an operation

After applying an operation to a table $T = (C, \delta)$, we obtain a new table $T' = (C', \delta')$. An operation may change the category structure C and the mapping δ. If C is changed, the domain $\otimes fr(C)$ of δ is also changed, which causes a change in the associations between labels and entries. Mathematically, δ' defines new values for the entries of T'. In the editing model, we could assign no values to δ'; however, in most cases, the values of δ' depend on the values of δ. Thus, we generate new entry values from old entry values according to the requirements of the different operations.

The table operations that change the dimension of a table generate a new table in which $\otimes fr(C')$ is different from $\otimes fr(C)$; thus, δ' associates new values for all the entries in $\otimes fr(C')$. Suppose we insert a new category, which contains l frontier label sequences, into an n-dimensional table T. Before the insertion, each entry is associated with n frontier label sequences from n different categories. After the insertion, the number of entries increases by a factor l and each entry is now associated with $n + 1$ frontier label sequences. For this operation, we assign the value of an old entry to the l new entries that are also associated with the frontier label sequences of the old entry. Removing a category with k frontier label sequences from a table T is more complex. In contrast with insertion, the number of entries in the new table is smaller and each entry is now associated with $n - 1$ frontier label sequences. Each new entry corresponds to the k old entries that were associated with the common frontier label sequences of the new entry. There are many possible ways to assign values for the new entries. For example, we may choose one of the values from the k old entries or use the average of these values. It is impossible to make an appropriate choice unless we know the motivation for removing the category. We should provide a method that allows users to make the decision. Our strategy is to assign a multiset of the k old entry values to the new entry so that we can generate a new value from it with subsequent operations.

When we consider the category operations that change the label structure of a category, only some of the entries in $\otimes fr(C')$ are affected. δ' needs to assign new values for only the affected entries and should not change the other associations. For example, consider the operation that relocates a subcategory within a category. The modified category loses some frontier label sequences and gains some new ones. The value of an old entry that was associated with a lost frontier label sequence is assigned to the new entry that is associated with the corresponding new frontier label sequence. Another example is the deletion of a subcategory from a category. After the deletion, the modified category loses some frontier label sequences and may also gain one new one. If the deletion of a subcategory results in a new frontier label sequence being added to the category, we also assign a multiset of the old entry values associated with the lost frontier label sequences to the new entry associated with the new frontier label sequence.

The label and entry operations that change only the entry values and labels are much easier to handle. These operations affect only one label or one entry.

3.3 Labeled-domain operations

Since we use labeled domains to model the category structure, we define some basic operations for labeled domains before we define editing operations. We also need to define some operations for labels and label sequences.

Label operations

Given two labels x and y, xy is the catenation of x and y and $x\backslash y$ is the left quotient of x and y. For example, if $x =$ "lab" and $y =$ "labeled", then $xy =$ "lablabeled" and $x\backslash y =$ "eled". We define $strip(x, y)$ to be $x\backslash y$ if x is a prefix of y and to be y, otherwise.

Label-Sequence operations

Given a label sequence l, the *first label* in l, denoted by $first(l)$, is undefined if l is the empty label sequence; otherwise, it is the label l_1 such that $l = l_1.l_2$ and l_2 is a label sequence. The *last label* in l, denoted by $last(l)$, is undefined if l is the empty label sequence; otherwise, it is the label l_2 such that $l = l_1.l_2$ and l_1 is a label sequence. The *front* of a label sequence l, denoted by $front(l)$, is undefined if l is the empty label sequence; otherwise, it is the label sequence l' such that $l = l'.last(l)$. The *back* of a label sequence l, denoted by $back(l)$, is undefined if l is the empty label sequence; otherwise, it is the label sequence l' such that $l = first(l).l'$.

Given a labeled domain d, a label sequence l of d determines a labeled subdomain d' of d. The subdomain d' satisfies $lbl(d') = last(l)$. Observe that $fr(d')$ satisfies the relation

$$\{l.back(h) : h \in fr(d')\} \subseteq fr(d).$$

Expansion

Given a labeled domain d and a label sequence l of d such that $first(l) = lbl(d)$, the expansion of d with l, denoted by $d + l$, is the labeled domain d' such that

$$fr(d') = (fr(d) \cup \{l\}) - \{x : \exists y \text{ a non-empty label sequence and } xy = l\}.$$

From the viewpoint of a labeled tree, $d+l$ adds one or more nodes to d to ensure that there is a path from the root to a frontier node identified by l. To maintain the consistency of $fr(d')$, we need to remove all prefix label sequences of l that were frontier label sequences of d. We generalize this operation for a set L of label sequences in the obvious way; we denote it by $d + L$. Fig. 3.2(a) illustrates the expansion of a labeled domain d with two label sequences $d.d1.d4$ and $d.a1.a2$.

Contraction

Given a labeled domain d and a label sequence l of d, the contraction of d with l, denoted by $d - l$, is the labeled domain d' such that

$$fr(d') = (fr(d) - \{l\}) \cup \{front(l) : front(l).y \text{ is not in } fr(d), \text{ for any } y\}.$$

From the viewpoint of a labeled tree, $d - l$ removes the subtree whose root is the node identified by l. If the node identified by $front(l)$ becomes a frontier node after removing the node identified by l, $front(l)$ will be added to $fr(d')$. Obviously, if $fr(d)$ is consistent, $fr(d')$ is also consistent. We generalize this operation for a set L of label sequences in the obvious way; we denote it by $d - L$. Fig. 3.2(b) illustrates the contraction of a labeled domain d with label sequences $d.d1$ and $d.d2.d5$.

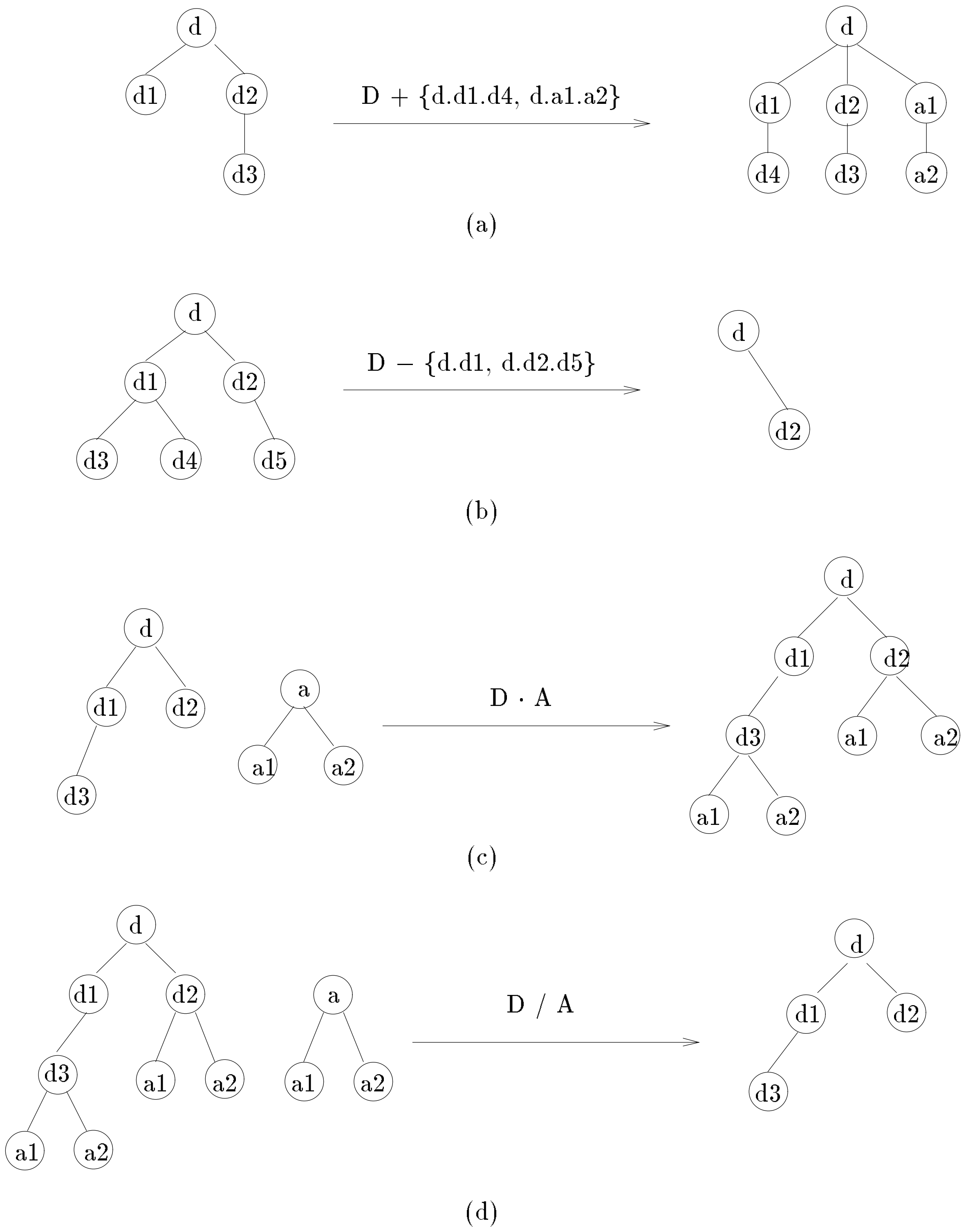


Figure 3.2: Examples of the labeled-domain operations.

Product

Given two labeled domains d_1 and d_2, the product of d_1 and d_2, denoted by $d_1 \cdot d_2$, is the labeled domain d that satisfies

$$fr(d) = \{l_1.back(l_2) : l_1 \in fr(d_1) \wedge l_2 \in fr(d_2)\}.$$

That is, l is in $fr(d)$ if and only if there are unique l_1 and l_2 such that l_1 is in $fr(d_1)$, l_2 is in $fr(d_2)$, and $l = l_1.back(l_2)$. Fig. 3.2(c) illustrates the product of two labeled domains D and A.

Quotient

Given two labeled domains d_1 and d_2, the quotient of d_1 and d_2, denoted by d_1/d_2, is the labeled domain d such that

$$d_1 = d \cdot d_2.$$

If there is no d such that $d_1 = d \cdot d_2$, then d_1/d_2 is undefined. Fig. 3.2(d) illustrates the quotient of two labeled domains D and A.

3.4 Editing operations for abstract tables

We propose 18 editing operations for the manipulation of abstract tables. We describe the syntax of these operations in a functional form by giving the names of the operations and the types of their operands and results. We have used extra white space to divide them into three groups, namely, tabular operations, category operations and label and entry operations:

Empty :		$\rightarrow$ *table*
Insert_Category :	*table* $\times$ *labeled domain*	$\rightarrow$ *table*
Delete_Category :	*table* $\times$ *label seq.*	$\rightarrow$ *table*
Duplicate_Category :	*table* $\times$ *label seq.* $\times$ *label*	$\rightarrow$ *table*
Combine_Categories :	*table* $\times$ *label seq.* $\times$ *label seq.*	$\rightarrow$ *table*
Split_Category :	*table* $\times$ *label seq.* $\times$ *label seq.* $\times$ *label*	$\rightarrow$ *table*

$Insert\_Subcategory:$	$table \times label\ seq. \times label\ seq. \times labeled\ domain$	$\rightarrow table$
$Delete\_Subcategory:$	$table \times label\ seq. \times label\ seq.$	$\rightarrow table$
$Move\_Subcategory:$	$table \times label\ seq. \times label\ seq. \times label\ seq.$	$\rightarrow table$
$Duplicate\_Subcategory:$	$table \times label\ seq. \times label\ seq. \times label\ seq. \times label$	$\rightarrow table$
$Combine\_Subcategories:$	$table \times label\ seq. \times label\ seq. \times label\ seq.$	$\rightarrow table$
$Split\_Subcategory:$	$table \times label\ seq. \times label\ seq. \times label\ seq. \times label$	$\rightarrow table$
$Promote\_Subcategories:$	$table \times label\ seq. \times label\ seq. \times label\ set$	$\rightarrow table$
$Demote\_Subcategories:$	$table \times label\ seq. \times label\ seq. \times label\ set \times label$	$\rightarrow table$

$Change\_Label:$	$table \times label\ seq. \times label\ seq. \times label$	$\rightarrow table$
$Change\_Entry\_Value:$	$table \times entry \times entry\ value$	$\rightarrow table$
$Compute\_Entry\_Value:$	$table \times entry \times operator$	$\rightarrow table$
$Get\_Entry\_Value:$	$table \times entry$	$\rightarrow entry\ value$

The semantics of an operation can be specified by giving the change in an abstract table as a result of applying the operation; thus, the operations are independent of a table's presentational form. We define the semantics of these operations using the labeled-domain operations defined in the previous section. To make them easier to understand, we use concrete tables rather than abstract tables to present examples of the operations; thus, we have to specify the label orders for the categories and the placement of categories in the stub and boxhead for these concrete tables. All the operations, however, are ordering independent. We use the notation $\{\ddagger \cdots \ddagger\}$ to represent a multiset.

Empty

This operation generates an empty table (C, δ), where $C = \emptyset$ and $\delta = \emptyset$.

Insert_Category

This operation adds a new category to a table. Given a table $T = (C, \delta)$ and a category d, we obtain a new table $T' = (C', \delta')$, where $C' = C \cup \{d\}$ and δ' is defined as follows. For each $f' \in \otimes fr(C')$, there is a unique $f \in \otimes fr(C)$ and a frontier label sequence $l \in fr(d)$ such that $f' = f \cup \{l\}$. We define $\delta'(f') = \delta(f)$. For example, Table 3.4 is generated by

Table 3.4: The average marks for 1991–1992.

Year	Term	Section	Mark					
			Assignments			Examinations		Grade
			Ass1	Ass2	Ass3	Midterm	Final	
1991	Winter	Section1	85	80	75	60	75	75
		Section2	85	80	75	60	75	75
	Spring	Section1	80	65	75	60	70	70
		Section2	80	65	75	60	70	70
	Fall	Section1	80	85	75	55	80	75
		Section2	80	85	75	55	80	75
1992	Winter	Section1	85	80	70	70	75	75
		Section2	85	80	70	70	75	75
	Spring	Section1	80	80	70	70	75	75
		Section2	80	80	70	70	75	75
	Fall	Section1	75	70	65	60	80	70
		Section2	75	70	65	60	80	70

adding a new category

$$(Section, \{(section1, \emptyset), (section2, \emptyset)\})$$

to Table 3.1.

Delete_Category

This operation removes a category from a table. Given a table $T = (C, \delta)$ and a category d in C, we obtain a new table $T' = (C', \delta')$, where $C' = C - \{d\}$ and δ' is defined as follows. For each $f \in fr(C')$, there are $|fr(d)|$ frontier label sequences $l_1, l_2, \ldots, l_{|fr(d)|}$ such that $f \cup \{l_i\} \in fr(C)$. We define

$$\delta'(f) = \{\ddagger\ \delta(f \cup \{l\})\ :\ l \in fr(d)\ \ddagger\}.$$

Table 3.5: The average marks for 1991–1992.

Year	Mark					
	Assignments			Examinations		Grade
	Ass1	Ass2	Ass3	Midterm	Final	
1991	85	80	75	60	75	75
	80	65	75	60	70	70
	80	85	75	55	80	75
1992	85	80	70	70	75	75
	80	80	70	70	75	75
	75	70	65	60	80	70

For example, if we remove the category **Term** from Table 3.1, we get Table 3.5, in which each entry is a multiset of marks that were associated with the three removed terms. If we do not keep the repeated values, we may not get the appropriate result. We can also supply an operator to Compute_Entry_Value to remove the repeated elements.

Duplicate_Category

This operation duplicates a category for a table. Given a table $T = (C, \delta)$, a category d in C and a label l which should be different from the labels of categories in C, we obtain a new table $T' = (C', \delta')$, where

$$C' = C \cup \{(l, set(d))\}$$

and δ' is defined as follows. For each $f' \in \otimes fr(C')$, there is an $f \in fr(C)$ and a frontier label sequence $s \in fr(d)$ such that $f' = f \cup \{s\}$. We define $\delta'(f') = \delta(f)$. For example, suppose d is the category:

$$(From, \{(Toronto, \emptyset), (Vancouver, \emptyset), (Montreal, \emptyset), (Ottawa, \emptyset),$$
$$(Edmonton, \emptyset), (Calgary, \emptyset)\}),$$

Table 3.6: The frame of a flight schedule between major cities of Canada.

From	To					
	Toronto	Vancouver	Montreal	Ottawa	Edmonton	Calgary
Toronto						
Vancouver						
Montreal						
Ottawa						
Edmonton						
Calgary						

then the following operations

$$\begin{aligned}
T_1 &:= Empty \\
T_2 &:= Insert\_Category(T_1, From) \\
T_3 &:= Copy\_Category(T_2, From, To)
\end{aligned}$$

generate the frame of a flight schedule between major cities of Canada as shown in Table 3.6.

Combine_Categories

This operation combines two categories of a table using the product of labeled domains. Given a table $T = (C, \delta)$ and two categories c_1 and c_2 in C, we obtain a new table $T' = (C', \delta')$, where

$$C' = (C - \{c_1, c_2\}) \cup \{c_1 \cdot c_2\}$$

and δ' is defined as follows. First, observe that $size(T') = size(T)$; therefore, T' and T have the same number of entries. For each $f \in \otimes fr(C)$, there are $l_i \in fr(c_i)$, for $i = 1, 2$, such that $\{l_1, l_2\} \subseteq f$. There is a unique corresponding $f' \in \otimes fr(C')$ such that

$$f' = (f - \{l_1, l_2\}) \cup \{l_1.back(l_2)\}.$$

Table 3.7: The average marks for 1991–1993.

Year		Mark					
		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

We define $\delta(f') = \delta(f)$. For example, after combining categories **Year** and **Term** in Table 3.1, the new table contains only the two categories **Year** and **Mark. Mark** keeps the same label structure as before and **Year** has the new label structure:

$$\begin{aligned}(Year,\{&(1991, \{(Winter, \emptyset), (Spring, \emptyset), (Fall, \emptyset)\}),\\ &(1992, \{(Winter, \emptyset), (Spring, \emptyset), (Fall, \emptyset)\})\\ &\}).\end{aligned}$$

The new table, which is shown in Table 3.7, looks similar to Table 3.1 except that the stub head contains only the name of the category **Year**.

Split_Category

This operation splits a category of a table into two categories using the quotient of labeled domains. Given a table $T = (C, \delta)$, a category c in C, a label sequence s of c such that $set(s) \neq \emptyset$ and $c/\Lambda(s)$ is not undefined, and a label l which is different from the labels of the categories in C, we obtain two categories c_1 and c_2 in this way: $c_1 = c/\Lambda(s)$, the quotient of c and $\Lambda(s)$, and c_2 is $(l, set(s))$, the labeled domain obtained after assigning a new label l for $\Lambda(s)$. We obtain a new table $T' = (C', \delta')$, where

$$C' = (C - \{c\}) \cup \{c_1, c_2\}$$

and δ' is defined as follows. First, observe that $size(T') = size(T)$; therefore, T' and T have the same number of entries. For each $f \in \otimes fr(C)$, there is an $d \in f$ such that $d = l_1.l_2 \in fr(c)$, where $l_1 \in fr(c_1)$ and l_2 is the back of a frontier label sequence in $fr(\Lambda(s))$. There is a unique corresponding $f' \in \otimes fr(C')$ such that

$$f' = (f - \{l_1.l_2\}) \cup \{l_1, l.l_2\}.$$

We define $\delta(f') = \delta(f)$. For example, suppose T specifies the logical structure of Table 3.7, which contains only two categories, **Year** and **Mark.** We can split category **Year** into two categories, **Year** and **Term**, by performing the operation

$$Split\_Category(T, Year, Year.1991, Term)$$

to change Table 3.7 back to Table 3.1.

Insert_Subcategory

This operation expands a category by inserting a new subcategory into it. Given a table $T = (C, \delta)$, a category c in C, a labeled domain d, and a label sequence s of c, the insertion of d into c with respect to s is a category c' that satisfies

$$c' = c + \{s.x : x \in fr(d)\}.$$

We obtain a new table $T' = (C', \delta')$, where

$$C' = (C - \{c\}) \cup \{c'\}$$

and δ' is defined as follows. For each $f \in \otimes fr(C')$, if $f \in \otimes fr(C)$, we define $\delta'(f) = \delta(f)$. If $f \in \otimes fr(C') - \otimes fr(C)$, there must be a $t \in fr(d)$ such that $s.t \in f$; thus, we define $\delta'(f) = \delta((f - \{s.t\}) \cup \{s\})$ if $s \in fr(c)$; otherwise, it is undefined. For example, Table 3.8 is the result of inserting $(Summer, \emptyset)$ into the category **Term** with respect to **Term** and $(0.3A + 0.3M + 0.4F, \emptyset)$ into the category **Mark** with respect to **Grade** in Table 3.1.

Table 3.8: The average marks for 1991–1992.

Year	Term	Mark					
		Assignments			Examinations		Grade
		Ass1	Ass2	Ass3	Midterm	Final	0.3A+0.3M+0.4F
1991	Winter	85	80	75	60	75	75
	Spring	80	65	75	60	70	70
	Summer						
	Fall	80	85	75	55	80	75
1992	Winter	85	80	70	70	75	75
	Spring	80	80	70	70	75	75
	Summer						
	Fall	75	70	65	60	80	70

Delete_Subcategory

This operation removes a subcategory from a category of a table. Given a table $T = (C, \delta)$, a category c in C, and a label sequence s of c, the deletion of c with respect to s is a category c' that satisfies

$$c' = c - s.$$

We obtain a new table $T' = (C', \delta')$, where

$$C' = (C - \{c\}) \cup \{c'\}$$

and δ' is defined as follows. For each $f \in \otimes fr(C')$, if $f \in \otimes fr(C)$, we define $\delta'(f) = \delta(f)$. If $f \in \otimes fr(C') - \otimes fr(C)$, f must contain $front(s)$ which becomes a frontier label sequence of d' after removing s; thus, we define

$$\delta'(f) = \{\ddagger\ \delta((f - \{front(s)\}) \cup \{front(s).k\}) : k \in fr(\Lambda(s))\ \ddagger\}.$$

For example, after deleting the labeled domains $(Summer, \emptyset)$ and $(0.3A + 0.3M + 0.4F, \emptyset)$ from Table 3.8, we obtain Table 3.1.

Move_Subcategory

This operation moves a subcategory inside a category of a table. Given a table $T = (C, \delta)$, a category c in C, and two label sequences s and p of c, we obtain a new category c' by making labeled domain $\Lambda(s)$ a labeled subdomain of $\Lambda(p)$:

$$c' = (c - s) + \{p.x : x \in fr(s)\}.$$

We obtain a new table $T' = (C', \delta')$, where

$$C' = (C - \{c\}) \cup \{c'\},$$

and δ' is defined as follows. For each $f \in \otimes fr(C')$, if $f \in \otimes fr(C)$, we define $\delta'(f) = \delta(f)$. If $f \in \otimes fr(C') - \otimes fr(C)$, there are two cases:

1. f contains $front(s)$ and $front(s)$ is a frontier label sequence of c', in which case $\delta'(f)$ is undefined.
2. f contains a label sequence $p.t$, where $t \in fr(s)$, in which case

$$\delta'(f) = \delta((f - \{p.t\}) \cup \{front(s).t\}).$$

For example, suppose that **Ass3** is a quiz and we want to reclassify it as an examination. We can move subcategory **Ass3** under **Examinations** with this operation and change its label to **Quiz** to obtain Table 3.9.

Duplicate_Subcategory

This operation duplicates a subcategory inside a category of a table. Given a table $T = (C, \delta)$, a category c in C, two label sequences s and p of c, and a label l which is different from the labels of the labeled domains in $set(p)$, we obtain a new category c' by adding a copy of $\Lambda(s)$ to $set(p)$ after labeling the new labeled domain as l:

$$c' = c + \{l.x : x \in fr(set(s))\}.$$

We obtain a new table $T' = (C', \delta')$, where

$$C' = (C - \{c\}) \cup \{c'\}$$

Table 3.9: The average marks for 1991–1992.

Year	Term	Mark					
		Assignments		Examinations			Grade
		Ass1	Ass2	Midterm	Final	Quiz	
1991	Winter	85	80	60	75	75	75
	Spring	80	65	60	70	75	70
	Fall	80	85	55	80	75	75
1992	Winter	85	80	70	75	70	75
	Spring	80	80	70	75	70	75
	Fall	75	70	60	80	65	70

and δ' is defined as follows. For each $f \in \otimes fr(C')$, if $f \in \otimes fr(C)$, we define $\delta'(f) = \delta(f)$. If $f \in \otimes fr(C') - \otimes fr(C)$, f must contain a label sequence $p.l.t$, where $t \in fr(set(s))$; thus, we define

$$\delta'(f) = \delta((f - \{p.l.t\}) \cup \{s.t\}).$$

For example, if we want to add one more assignment **Ass4** under **Assignments** to Table 3.1 and the marks for assignment 4 are almost the same as for assignment 3, we can use this operation to duplicate subcategory **Ass3** and its associated entries to obtain Table 3.10.

Combine_Subcategories

This operation combines two subcategories in a category of a table using the product of labeled domains. It is similar to Combine_Categories except that the operation is applied to subcategories. Given a table $T = (C, \delta)$, a category c in C, and two label sequences s_1 and s_2 of c such that s_2 is not a prefix of s_1, we obtain a new category c' by removing labeled domains $\Lambda(s_1)$ and $\Lambda(s_2)$ from c and adding a new labeled domain $\Lambda(s_1) \cdot \Lambda(s_2)$

Table 3.10: The average marks for 1991–1992.

Year	Term	Mark						
		Assignments				Examinations		Grade
		Ass1	Ass2	Ass3	Ass4	Midterm	Final	
1991	Winter	85	80	75	75	60	75	75
	Spring	80	65	75	75	60	70	70
	Fall	80	85	75	75	55	80	75
1992	Winter	85	80	70	70	70	75	75
	Spring	80	80	70	70	70	75	75
	Fall	75	70	65	65	60	80	70

to $set(front(s_1))$:

$$c' = ((c - s_1) - s_2) + \{front(s_1).x : x \in fr(\Lambda(s_1) \cdot \Lambda(s_2))\}.$$

We obtain a new table $T' = (C', \delta')$, where

$$C' = (C - \{c\}) \cup \{c'\}$$

and δ' is defined as follows. For each $f \in \otimes fr(C')$, if $f \in \otimes fr(C)$, we define $\delta'(f) = \delta(f)$. If $f \in \otimes fr(C') - \otimes fr(C)$, there are two cases:

1. f contains $front(s_2)$ and $front(s_2)$ is a frontier label sequence of c', in which case $\delta'(f)$ is undefined.

2. f contains a label sequence $s_1.u.v$, where $u \in fr(set(s_1))$ and $v \in fr(set(s_2))$, in which case

$$\delta'(f) = \{\ddagger\ \delta((f - \{s_1.u.v\}) \cup \{s_1.u\}), \delta((f - \{s_1.u.v\}) \cup \{s_2.v\})\ \ddagger\}.$$

For example, suppose T is the conversion table from pounds to kilograms in the range of 0 to 19 pounds shown in Table 3.11. We can change the label structure of the category

Table 3.11: A conversion table from pounds to kilograms.

Pounds		Kilograms
One digit	0	0.00
	1	0.45
	2	0.90
	3	1.36
	4	1.81
	5	2.26
	6	2.72
	7	3.17
	8	3.63
	9	4.08
Two digits	00	0.00
	10	4.54

Pounds into the structure of Table 3.12 by performing the operation

$$Combine\_Subcategory(T, Pounds, Pounds.two\ digits, Pounds.one\ digit).$$

To convert Table 3.12 into a conversion table we need to add the values in each entry multiset using the operation

$$Compute\_Entry\_Value(T', Pounds.two\ digits.i.j, Sum)$$

where $i = 00$ or 10 and $j = 0, \ldots, 9$.

Split_Subcategory

This operation splits a subcategory in a category of a table into two subcategories using the quotient of labeled domains. It is similar to Split_Category except that the operation is applied to subcategories. Given a table $T = (C, \delta)$, a category c in C, two label sequences s_1 and s_2 of c such that s_1 is a prefix of s_2, and a label l which is different

Table 3.12: After combining two subcategories in Table 3.11.

Pounds			Kilograms
Two digits	00	0	0.00 0.00
		1	0.00 0.45
		2	0.00 0.90
		3	0.00 1.36
		4	0.00 1.81
		5	0.00 2.26
		6	0.00 2.72
		7	0.00 3.17
		8	0.00 3.63
		9	0.00 4.08
	10	0	4.54 0.00
		1	4.54 0.45
		2	4.54 0.90
		3	4.54 1.36
		4	4.54 1.81
		5	4.54 2.26
		6	4.54 2.72
		7	4.54 3.17
		8	4.54 3.63
		9	4.54 4.08

from the labels of the labeled domains in $set(front(s_1))$, we obtain a new category c' by removing labeled domain $\Lambda(s_1)$ from c and adding two new labeled domains $\Lambda(s_1)/\Lambda(s_2)$ and $(l, set(s_2))$ to $set(front(s_1))$:

$$c' = ((c - s_1) + \{front(s_1).x : x \in fr(\Lambda(s_1)/\Lambda(s_2))\}) + \{front(s_1).l.x : x \in fr(set(s_2))\}.$$

We obtain a new table $T' = (C', \delta')$, where

$$C' = (C - \{c\}) \cup \{c'\}$$

and δ' is defined as follows. For each $f \in \otimes fr(C')$, if $f \in \otimes fr(C)$, we define $\delta'(f) = \delta(f)$. If $f \in (\otimes fr(C') - \otimes fr(C))$, there are two cases:

1. f contains a label sequence $front(s_1).t$, where $t \in fr(\Lambda(s_1)/\Lambda(s_2))$, in which case

$$\delta'(f) = \{\ddagger\ \delta((f - \{front(s_1).t\}) \cup \{front(s_1).t.u\}) : u \in fr(set(s_2))\ \ddagger\}.$$

2. f contains $front(s_1).l.t$, where $t \in fr(set(s_2))$, in which case

$$\delta'(f) = \{\ddagger\ \delta((f - \{front(s_1).l.t\}) \cup \{s_1.u.t\}) : s_1.u.t \in fr(c)\ \ddagger\}.$$

For example, suppose T is the conversion table of Table 3.13. We can change the label structure of T into Table 3.14 by performing the operation

$$Split\_Subcategory(T, Pounds, Pounds.two\ digits, Pounds.two\ digits.00, one\ digit).$$

Promote_Subcategories

This operation promotes a set of subcategories up one level in a category. Given a table $T = (C, \delta)$, a category c in C, a label sequence s of c, and a set L of labels of the labeled domains in $set(s)$, we obtain a new category c' by moving labeled domains $\Lambda(s.x), x \in L$, to $set(front(s))$ and assign $last(s)x$ as the labels of the corresponding promoted labeled

Table 3.13: A conversion table from pounds to kilograms.

Pounds			Kilograms
Two digits	00	0	0.00
		1	0.45
		2	0.90
		3	1.36
		4	1.81
		5	2.26
		6	2.72
		7	3.17
		8	3.63
		9	4.08
	10	0	4.54
		1	4.99
		2	5.44
		3	5.90
		4	6.35
		5	6.80
		6	7.26
		7	7.71
		8	8.17
		9	8.62

Table 3.14: After splitting a subcategory in Table 3.13.

Pounds		Kilograms	
One digit	0	0.00	4.54
	1	0.45	4.99
	2	0.90	5.44
	3	1.36	5.90
	4	1.81	6.35
	5	2.26	6.80
	6	2.72	7.26
	7	3.17	7.71
	8	3.63	8.17
	9	4.08	8.62
Two digits	00	0.00	0.45
		0.90	1.36
		1.81	2.26
		2.72	3.17
		3.63	4.08
	10	4.54	4.99
		5.44	5.90
		6.35	6.80
		7.26	7.71
		8.17	8.62

subdomains. If $set(s)$ is empty after the promotion, the labeled domain $\Lambda(s)$ is also removed from the category. We can define c' as:

$$\begin{aligned} c' &= ((c - \{s.x : x \in L\}) - \{s : L = \{lbl(x) : x \in set(s)\}\}) \\ &\quad + \cup_{x \in L}\{front(s).last(s)x.t : t \in fr(set(s.x))\}. \end{aligned}$$

We obtain a new table $T' = (C', \delta')$, where

$$C' = (C - \{c\}) \cup \{c'\}$$

and δ' is defined as follows. For each $f \in \otimes fr(C')$, if $f \in \otimes fr(C)$, we define $\delta'(f) = \delta(f)$. If $f \in (\otimes fr(C') - \otimes fr(C))$, there is a label sequence $u \in f$ such that $u = front(s).last(s)x.t$, where $x \in L$ and $t \in fr(set(s.x))$; thus, we define

$$\delta'(f) = \delta((f - \{u\}) \cup \{s.x.t\}).$$

For example, suppose T identifies the logical structure of Table 3.1; then we can generate Table 3.15 by performing following operations:

$T_1 := Combine\_Categories(T, Year, Term)$
$T_2 := Promote\_Subcategories(T_1, Year, Year.1991, \{Winter, Spring, Fall\})$
$T_3 := Promote\_Subcategories(T_2, Year, Year.1992, \{Winter\})$.

Demote_Subcategories

This operation demotes a set of subcategories down one level in a category. Given a table $T = (C, \delta)$, a category c in C, a label sequence s of c, a set L of labels of the labeled domains in $set(s)$, and a label l that is different from the labels of the remaining labeled domains in $set(s)$, we obtain a new category c' by replacing all the labeled domains in $set(s)$ whose labels are in L with a new labeled domain

$$(l, \{(strip(l, lbl(x)), set(x)) : x \in set(s) \wedge lbl(x) \in L\}).$$

For each demoted labeled domain, if the old label contains l as a prefix, the new label is the remaining part of the old label after removing the prefix l; otherwise, the label is unchanged. We can define c' as:

$$c' = (c - \{s.x \in L\}) + \bigcup_{x \in L}\{s.strip(l, x).t : t \in fr(set(s.x))\}.$$

Table 3.15: The average marks for 1991–1992.

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

We obtain a new table $T' = (C', \delta')$, where

$$C' = (C - \{c\}) \cup \{c'\}$$

and δ' is defined as follows. For each $f \in \otimes fr(C')$, if $f \in \otimes fr(C)$, we define $\delta'(f) = \delta(f)$. If $f \in (\otimes fr(C') - \otimes fr(C))$, there is a label sequence $u \in f$ such that $u = s.strip(l, x).t$, where $x \in L$ and $t \in fr(set(s.x))$; thus, we define

$$\delta'(f) = \delta((f - \{u\}) \cup \{s.x.t\}).$$

For example, suppose T identifies the logical structure of Table 3.15; then we can generate Table 3.16 by performing the operation

$$\begin{aligned} &Demote\_Subcategories(T, Year, Year,\\ &\qquad \{1991Winter, 1991Spring, 1991Fall\},\\ &\qquad 1991). \end{aligned}$$

Table 3.16: The average marks for 1991–1992.

Year		Mark					
		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
1992Winter		85	80	70	70	75	75
1992	Spring	80	80	70	70	75	75
	Fall	75	70	65	60	80	70

Change_Label

This operation changes the label of a labeled domain in a category. Given a table $T = (C, \delta)$, a category c in C, a label sequence s of c, and a label l that is different from the labels of the labeled domains in $set(front(s))$, we obtain a new category c' by replacing the old label of $\Lambda(s)$ with l. We obtain a new table $T' = (C', \delta')$, where

$$C' = (C - \{c\}) \cup \{c'\}$$

and δ' is defined as follows. For each $f \in \otimes fr(C')$, if $f \in \otimes fr(C)$, we define $\delta'(f) = \delta(f)$. If $f \in (\otimes fr(C') - \otimes fr(C))$, there is a label sequence $u \in f$ such that $u = front(s).l.t$, where $t \in fr(set(s))$; thus, we define $\delta'(f) = \delta((f - \{u\}) \cup \{s.t\})$.

Change_Entry_Value

This operation assigns a new value for an entry in a table. Given a table $T = (C, \delta)$, an entry e in $\otimes fr(C)$, and a value v of any kind, we obtain a new table $T' = (C', \delta')$, where $C' = C$ and δ' is defined as follows. For each $f \in \otimes fr(C')$, we define $\delta'(f) = v$ if $f = e$, and $\delta'(f) = \delta(f)$, otherwise.

Compute_Entry_Value

This operation computes a new value based on the old value of an entry in a table. Given a table $T = (C, \delta)$, an entry e in $\otimes fr(C)$, and a user-defined operation op which takes an entry value as an operand, we obtain a new table $T' = (C', \delta')$, where $C' = C$, and δ' is defined as follows. For each $f \in \otimes fr(C')$, we define $\delta'(f) = op(\delta(f))$ if $f = e$, and $\delta'(f) = \delta(f)$, otherwise. Given an entry value v, suppose we define an operation *Sum* that returns the sum of the numbers in v if v is a multiset, and returns v, otherwise. We can use Compute_Entry_Value with *Sum* to generate Table 3.13 from Table 3.12.

The frequently-used user-defined operations are for numerical calculations such as *Sum*, *Product*, *Average*, *Minimum*, *Maximum*, and so on. There are also many other useful operations; for example, transforming a multiset into a set or catenating all elements in a set. We can implement Compute_Entry_Value in a table editor in at least two ways. In the first approach, the system provides some frequently-used operations and users can choose only these operations for Compute_Entry_Value. In the second approach, the system provides a language to define user-defined operations and a mechanism to interpret the operations defined in that language. Our prototype adopts the first approach. How to implement Compute_Entry_Value using the second approach is left for future investigation.

Get_Entry_Value

This operation returns the value of an entry in a table. Given a table $T = (C, \delta)$ and an entry e in $\otimes fr(C)$, this operation returns $\delta(e)$.

3.5 Expressiveness of editing model

The editing model provides the basic operations that support the editing of tables as multi-dimensional logical structures. We can use these operations to compose tables step by step, from an empty table to a table with a complex structure. We can also construct complex operations from these operations for some special applications. We believe that we have provided complete operations for editing a single table as a multi-dimensional logical structure. A table that can be specified as a multi-dimensional logical structure consists of two parts: the categories which are hierarchical structures and the mapping from the categories to entries. The editing model provides sufficient operations to add and remove categories, to manipulate the category hierarchy, and to update the mapping from categories to entries. However, we do not provide operations that can be applied to more than one tables, for example, to combine or split tables. Suppose Table A contains categories X and Y, and Table B contains categories X and Z. We could combine Tables A and B to obtain Table C that contains the category X and a new category that is the conjunction of Y and Z.

We also believe that the operations in the editing model are non-redundant. One may argue that we need only the operations: *Empty*, *Insert_Category*, *Delete_Category*, *Insert_Subcategpry*, *Delete_Subcategory*, *Change_Label*, *Change_Entry_Value*, *Compute_Entry_Value*, and *Get_Entry_Value*, and that the other operations can be obtained from these operations. Suppose we decompose *Move_Subcategpry* into the two operations: *Delete_Subcategory* and *Insert_Subcategory*. After we delete a subcategory, all associated entries are also removed. When we insert a subcategory into a table, the associated entries are empty. Thus, the semantics of *Move_Subcategpry* is not preserved under decomposition. Similar problems occur when we decompose the other operations into sequences of more basic operations.

Chapter 4

Layout specification

The final purpose of tabular composition is to generate a concrete table in two dimensions such that it clearly exhibits its underlying logical structure. The layout of a table determines the efficiency of reading the table and the accuracy of obtaining pertinent information. There are two components that affect tabular layout. The *topology* of a table determines the arrangement of tabular items in two dimensions and the *style* governs the final appearance of different tabular components. We have discussed some guidelines for the specification of topology and styles in Sections 1.2.2 and 1.2.3. We now propose a presentational model to specify layouts for abstract tables. This model consists of a set of presentational rules for tabular topology and style. These presentational rules support the high-quality tabular layouts with respect to the topology and style guidelines.

4.1 Tabular Layouts

When we present a table as a row–column structure, we usually first arrange the labels in the stub and boxhead and then decide the positions of the entries according to the positions of their associated labels. Each entry is placed in a cell such that it is to the right of its associated labels in the stub and beneath its associated labels in the boxhead. In the abstract model, labels are grouped into categories; thus, the arrangement of labels can be determined by the arrangement of categories in the stub and the boxhead as well as by the label orderings of the categories. We use *topological specification* to describe the relative arrangement of tabular items in two dimensions.

The selection of style rules is the key to the design of high-quality layouts of tables. Most current tabular composition systems provide only style rules that govern the appearance of *layout objects*, such as rows, columns or blocks. In the traditional style sheets of tables, we usually need to specify only the style for the whole table and its major regions, including the stub, the boxhead, the stub head, and the body. Thus, it is useful to provide style rules for these *presentational objects*. In addition, we may need to specify styles that govern the appearance of *logical objects*, such as categories, labels and entries, no matter where these objects appear in a concrete table. The style rules for both the presentational objects and the logical objects enable us to control the appearance of a table independently of the tabular topology. In this way, we do not have to respecify style rules for a table after we change its topology. When we compose a document, we usually present all tables in a uniform style so as to achieve consistent appearance throughout the document. It is crucial that we can specify collective style rules to govern the general appearance of a collection of tables. We use *style specification* to describe the selection of style rules for a table or for a set of tables.

4.2 Topological specification

When a table contains more than two categories, multiple categories appear in the stub, in the boxhead, or in both although they are not orthogonal to each other. When this multiplicity occurs, the labels in these categories are either indented as shown in the stub of Table 4.1 or organized hierarchically as shown in the stub of Table 4.2. Different orderings of categories in the stub or in the boxhead give rise to different topological arrangements. By interchanging the order of **Year** and **Term**, we get the arrangement shown in Table 4.3. We use two topological rules to specify the category orderings, one for the stub and the other for the boxhead:

$$\begin{array}{ll} \text{STUB:} & C^s_1, C^s_2, \ldots, C^s_m \\ \text{BOXHEAD:} & C^b_1, C^b_2, \ldots, C^b_n, \end{array}$$

where C^s_i is the ith category in the stub and C^b_j is the jth category in the boxhead. For

Table 4.1: 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

Table 4.2: The average marks for 1991–1992.

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

Table 4.3: The average marks for 1991–1992.

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

example, the category orderings of Tables 4.1 and 4.2 can be specified by

STUB:	Year, Term
BOXHEAD:	Mark.

The label ordering within a category is another attribute that affects the topological arrangement. In Table 4.1, the labels in category **Term** are arranged in the order of **Winter**, **Spring**, **Fall**. If we reverse the order to give **Fall**, **Spring**, **Winter**, we get a different arrangement. Therefore, we need another topological rule to specify the label ordering within a category:

ORDER C: $L_1, L_2, \ldots, L_k$,

where C is a category and L_i is the ith label of C in the ordering. Sometimes, we do explicitly specify the label ordering for a category; instead, we implicitly specify the order using standard ordering, such as numerical order or lexicographic order. Thus, another form of topological rule for label ordering is:

ORDER C: <ordering option>,

where <ordering option> includes numerical order, reverse numerical order, lexicographic order, and reverse lexicographic order. For example, the label orderings for the categories in Table 4.1 can be specified as:

ORDER Year:	lexicographic order
ORDER Term:	Winter, Spring, Fall
ORDER Mark:	Assignments, Examinations, Grade.

This specification does not, however, completely describe the label orderings in Table 4.1 because it does not specify the orderings of labels **Ass1**, **Ass2** and **Ass3** for **Assignments**, and **Midterm** and **Final** for **Examinations**. We must use the topological rules for the subcategories. Thus, the complete ordering specification of the labels for Table 4.1 is

ORDER Year:	lexicographic order
ORDER Term:	Winter, Spring, Fall
ORDER Mark:	Assignments, Examinations, Grade
ORDER Mark.Assignments:	lexicographic order
ORDER Mark.Examinations:	Midterm, Final.

Sometimes, we need to order labels based on their associated entries. For example, in Table 4.4, the student IDs are ordered based on their grades (in the last column): the student IDs with associated higher grades appear earlier than student IDs with associated lower grades. To specify this kind of indirect ordering, we extend the topological rule for label ordering to:

ORDER C: <order option> [ON <label sequence set>].

If ON <label sequence set> is omitted, the labels are ordered with respect to their own values; otherwise, they are ordered with respect to the entries that are associated with the given label sequence set. We can specify the label orderings for Table 4.4 with:

ORDER Mark:	Midterm, Final, Grade
ORDER Student ID:	reverse numerical order ON {Mark.Grade}.

Table 4.4: The marks for CS340.

Student ID	Mark		
	Midterm	Final	Grade
90800108	90	96	93
90800103	92	88	90
90800112	82	84	83
90800102	73	85	79
90800100	82	68	75
90800111	54	86	70
90800114	70	64	67
90800101	64	68	66
90800104	50	68	59
90800110	45	61	53

Once we are given a topological specification, we can determine the topological positions of the labels and the entries of a table. The geometric positions, however, cannot be determined without a style specification.

4.3 Style specification

A style rule consists of a scope and a set of formatting attributes that are associated with the scope. For example, tables (scope) are displayed in Roman (formatting attribute) with horizontal rules only (formatting attribute). The style rules for tables fall into three classes: presentational-oriented style rules, content-oriented style rules, and layout-oriented style rules. A *presentational-oriented style rule* has a scope that is a major region of a table: the table itself, the stub, the boxhead, the stub head, and the body. It affects the cells and separations (rules and spacing) in the major regions. A *content-oriented style rule* has a scope that is a logical object or a set of logical objects of an abstract table, including a category, a subcategory, a label, an entry, an entry value, and an entry set. It affects only the cells in which the logical objects are located and the separations of these cells. A *layout-oriented style rule* has a scope that is a layout component of a concrete table, including a row, a column, and a block. It always affects the cells and separations in the layout component no matter what objects are put into it.

The presentational-oriented style rules are independent of both the logical structure and the topology of a table. These style rules determine the general appearance of a table, regardless of any change in the logical structure and topology. The content-oriented style rules are associated with the logical components of a table and are independent of the topology. These style rules are always applied to the items in their scopes, no matter where the items are placed. The layout-oriented style rules are independent of the logical structure and affect the appearance of a set of items that are dependent on the current topology. If we rearrange the tabular items, then the layout-oriented style rules may be applied to unexpected items and require adjustment. For example, we have specified the following style rules for Table 4.5:

TABLE:	Roman
	double line for the top and bottom edges of the frame
	single line for the stub and the boxhead separations only
STUB:	indented style
CATEGORY Year:	bold face
COLUMN 7:	grey background.

By applying these style rules to a new topology, the transposition of Table 4.5, we get Table 4.6. The general appearance of these two tables is similar because they have the same presentational-oriented style rules for the table and the stub. The labels of category **Year** are displayed in bold face for both tables, even though they are in different positions. Although the entries that are associated with label **Grade** are a logical unit, we intended to highlight these entries by specifying a layout-oriented style rule for column seven in the first topology. After the change of topology, this layout-oriented style rule is applied to the marks that are associated with 1992 Fall term. To highlight the correct items in the new topology, we have to remove the layout-oriented style rule for column seven and add a new rule for row ten. From this example, we see that presentational-oriented and content-oriented style rules enable us to specify styles for tables independently of their specific topologies. If we ignore the inconvenience caused by the layout-oriented

Table 4.5: 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

Table 4.6: The average marks for 1991–1992.

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

style rules when changing a table's topology, they have some advantages. First, since tables are presented as a row–column structure, we are accustomed to specifying style rules for rows and columns. Second, sometimes it is easier to specify layout-oriented style rules, than to specify content-oriented style rules to achieve the same effect. In the last example, we would need to use two content-oriented style rules to replace the layout-oriented style rule for column seven: one style rule for the label **Grade** and the other for the set of entries that are associated with label **Grade.**

In the remainder of this section, we first discuss the formatting attributes for different style rules and then we provide more details about the presentational-oriented style rules, the content-oriented style rules, and the layout-oriented style rules. We also introduce the concepts of collective style rules and specific style rules.

4.3.1 Formatting attributes

We provide eight types of formatting attributes for style rules:

- Cell style

  Thi allows us to control the appearance and the background of the items in cells. We can specify type faces and sizes, background colors, line spacing, leading spacing, horizontal and vertical alignment options, and so on.

- Separation style

  Appropriate separation of tabular items can assist readers to find information in table move easily. We should be able to select white space or different types of horizontal and vertical rules to separate different kinds of items.

- Frame style

  Sometimes we want to highlight the items in a particular rectangular area by placing rules or white space around the area. The frame style enables us to select white space or different types of rules to surround a rectangular area.

- Arrangement style

This style enables us to control the arrangement of labels in the stub, boxhead, and stub head. We can specify four different styles for the stub: hierarchical, indented, cut-in, and repeated. These styles are in common used. Since indented style and cut-in style are never applied to the boxhead, we can specify only repeated style and hierarchical style for the boxhead. We can fill the stub head with the headings of the categories in the stub or leave the stub head empty.

- Spanning style

  This allows us to span the entries that have the same value in a rectangular block. The spanning options are: no spanning, horizontal spanning only, vertical spanning only, horizontal spanning first, and vertical spanning first. These spanning options enable us to span entries in one dimension without spanning in the the other dimension, or to span the entries in two dimensions by giving priority to one dimension. Rectangular spanning is the most useful spanning shape for most tables. Other spanning shapes, such as an L shape, an ortho-convex shape, or even an arbitrary shape may be used in some tables, but it is unclear where to put the spanned value inside these shapes. Inappropriate placement of a spanned value may make the table less legible.

- Grouping style

  This groups items into blocks of a given number of rows by the use of either white space or rules. We can turn grouping on or off and specify how many rows are in a group. The grouping separation should be specified in the separation style. Grouping style is usually applied to tall tables to assisting searching for items. We do not provide vertical grouping since the grouping of columns is never observed.

- Category heading style

  This specifies the style of the category headings. For example, in Table 4.7 the category heading **Formatting attributes** is displayed above its labels, but the category heading **Scopes** is presented in the stub head. The display of category headings can help readers comprehend the logical structure of a table more easily. On the other hand, some category headings, such as **Year** or **Weekday**, are familiar to us and we can still interpret the logical structure even when these category headings are not displayed.

- Size constraints

  To present a table in limited space and also achieve an aesthetic layout, we may want to constrain the area, the column widths, or row heights. Size constraints enable us to restrict the size and the shape of tables.

Style rules may have different formatting attributes in different scopes. For example, the grouping style can be applied to the whole table only and the category heading style can be applied only to the scopes that are associated with categories. Table 4.7 shows the formatting attributes for different style rules. The same formatting attribute for different scopes may not allow the same choices. For example, the separation style for the whole table allows more separation specifications than the same style for the other scopes. We explain the differences in the following subsections.

4.3.2 Presentational-oriented style rules

Presentational-oriented style rules control the general appearance of a table and its four major regions. The scope of these style rules can be the whole table or one of its regions: the stub, the boxhead, the body, and the stub head.

A style rule for the whole table can specify the cell style, the separation style, the frame style, the grouping style, the category heading style, and the size constraints. The separation style includes the selections of rule types, rule widths, and white space for different kinds of separations in a table, including horizontal separation (which separates the rows), vertical separation (which separates the columns), grouping separation (which separates a group of rows), block separation (which horizontally and vertically separates the items that are associated with labels in different subhierarchies), stub separation (which vertically separates the stub and the stub head from the boxhead and the body), and boxhead separation (which horizontally separates the stub head and the boxhead from the stub and the body). For example, the separation styles:

Table 4.7: The formatting attributes for different style rules.

Scopes		Formatting attributes							
		Cell	Separ-ation	Frame	Arrange-ment	Spann-ing	Group-ing	Cate-gory	Size constr.
Present-ational-structure style rules	Table	√	√	√			√	√	√
	Stub	√	√		√			√	
	Boxhead	√	√		√			√	
	Stub head	√	√		√				
	Body	√	√			√			
Content-oriented style rules	Category	√	√					√	
	Subcategory	√	√	√					
	Label	√		√					
	Entry	√		√					
	Entry value	√		√					
	Entry set	√	√	√					
Layout-oriented style rules	Block	√	√	√		√			√
	Row	√	√	√		√			√
	Column	√	√	√		√			√

Table 4.8: The marks of CS340.

	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

Stub separation:	single line with 10pt white space
Boxhead separation:	single line with 10pt white space
Horizontal separation:	5pt white space
Vertical separation:	5pt white space
Block separation:	dashed line with 10pt white space

generate Table 4.8.

A frame style for the whole table includes the selection of rule types, rule widths, and white space for the left, right, top, and bottom edges of the table frame. The frame style for Table 4.8 is:

Left edge:	5pt white space
Right edge:	5pt white space
Top edge:	single lines with 5pt white space
Bottom edge:	single lines with 5pt white space

The grouping style includes the selection of grouping and the number of rows to be grouped. Table 4.9 is given by the following style rules:

Grouping style:	grouping is enabled with 5 rows in a group
Grouping separation:	dashed line with 10pt white space

The category-heading style controls the selection of the category headings for a table. The size constraints include lower and upper bounds for the table width and height and lower and upper bounds for the column widths and the row heights.

Style rules for the stub and the boxhead may specify the arrangement style, the cell style, the separation style, and the category heading style. The separation style is simpler than its counterpart for the whole table. It contains only the horizontal and vertical separation of items inside the stub or inside the boxhead. For the stub, the arrangement style can be indented style, hierarchical style, cut-in style (the labels in the first category cut into the body), and repeated style (the labels of a subcategory that has a subsubcategory is not spanned, but is repeated for each label in the subsubcategory). For the boxhead, the arrangement style can be either the hierarchical style or the repeated style. The style rules for the stub head may specify the arrangement style, the cell style, and the separation style. The arrangement style for the stub head can be empty or contain the category headings in the stub. The style rules for the body may specify the spanning style, the cell style, and the separation style. The spanning style is: no spanning, horizontal spanning only, vertical spanning only, horizontal spanning first, or vertical spanning first. The presentational-oriented style rules:

TABLE:	Roman
	double line for the stub and the boxhead separations, and
	thin single line for all other separation
	thick single line for the frame
STUB:	cut-in style
	bold face
BOXHEAD:	repeated style
	dashed lines for both horizontal and vertical separations
STUB HEAD:	empty
BODY:	no spanning

generates Table 4.10.

Table 4.9: The marks of CS340.

Student ID	Mark		
	Midterm	Final	Grade
90800100	82	68	75
90800101	64	68	66
90800102	73	85	79
90800103	92	88	90
90800104	50	68	59
90800108	90	96	93
90800110	45	61	53
90800111	54	86	70
90800112	82	84	83
90800114	70	64	67
90800115	60	70	75
90800116	88	70	79
90800117	65	71	68
90800118	94	80	87
90800119	72	72	72
90800201	85	75	80
90800202	46	60	53
90800203	98	90	94
90800204	74	84	89
90800205	88	60	70

Table 4.10: The average marks of some courses, 1991–1992.

	1991	1991	1991	1992	1992	1992
	Winter	Spring	Fall	Winter	Spring	Fall
CS241						
Midterm	60	60	55	70	70	60
Final	75	70	80	75	75	80
Grade	75	70	75	75	75	70
CS242						
Midterm	90	84	55	70	70	60
Final	76	70	75	80	50	84
Grade	83	77	65	75	60	72
CS246						
Midterm	60	80	95	60	78	74
Final	40	70	83	72	70	80
Grade	50	75	89	66	74	77

Table 4.11: The average marks for 1991–1992.

	Mark					
	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

4.3.3 Content-Oriented style rules

The scope of a content-oriented style can be a category, a subcategory, a label, an entry, a set of entries with the same value, or a set of entries that are associated with a label set.

The style rule for a category may specify the category heading style, the cell style, and the separation style. For example, the style rules:

CATEGORY Term:	bold face
CATEGORY Mark:	heading is displayed
	single line for horizontal separation

generate Table 4.11.

The style rule for a label, an entry or a set of entries with the same value may specify the cell style and the frame style. The style rules:

LABEL Mark.Grade: underlined

Table 4.12: 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

ENTRY {Year.1991, Term.Spring, Mark.Grade}:
bold face
dotted line for the frame
ENTRY VALUE 85: grey background

generate Table 4.12.

The style rule for a subcategory or for an entry set that is associated with a label set may specify the cell style, the separation style, and the frame style. The frame style controls all the frames of the blocks occupied by a subcategory or an entry set. The style rules:

SUBCATEGORY Examinations: dotted line for the left and right edges of the frame
dashed lines for the horizontal and vertical separation
ENTRY SET {Term.Winter, Mark.Assignments}:
grey background

generate Table 4.13.

Table 4.13: 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

4.3.4 Layout-Oriented style rules

The scope for a layout-oriented style rule can be a row, a column, or a block. The possible style rules for these scopes are the spanning style, the cell style, the separation style, the frame style and the size constraints. The size constraints specify lower and upper bounds of the column widths and row heights within the scope and lower and upper bounds on the total width and height within the scope. For example, the style rules:

COLUMN 7:	single line for the left edge of the frame
ROW 6:	grey background
BLOCK (8, 2, 10, 4):	horizontal spanning first
	dashed line for all separations
	single line for the top and right edges of the frame

generate Table 4.14.

Table 4.14: 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			70	75	75
Fall	75	70	65	60	80	70

4.3.5 Collective and specific style rules

The appearance of a table can be governed by many style rules. Some style rules are given by a publisher or an editor of a book to achieve a uniform appearance of all tables in the same book. Some style rules are given by a table designer for the specific presentation of one table. We can classify the style rules for a table into two classes: collective style rules and specific style rules.

A *collective style rule* is a style rule for the presentation of a collection of tables. A collective style rule can be any style rule that we have discussed in Sections 4.3.2 through 4.3.4. If a collective style rule is a presentational-oriented style rule, it should be applied to all the tables. If a collective style rule is a content-oriented style rule or layout-oriented style rule, it is applicable to only the tables that contains the scope of the style rule. For example, a collective style rule for a category, say the category **Year,** is applicable only to tables that contain a category named **Year,** and a collective style rule for a particular column, say the fifth column, is applicable only to tables that contain at least five columns.

Sometimes we need to override some formatting attributes of the collective style rules to present a table differently for specific reasons. In these cases we use *specific style rules.* For example, if we want to highlight the highest grades, we can use a specific style rule to set a grey background for the entries with the highest grades.

There are a number of advantages in the separation of the collective style rules from specific style rules. First, the collective style rules need to be specified only once for a collection of tables. Second, if we want to change the appearance of a collection of tables, we need to change only the collective style rules. Third, authors do not need to know the details of the collective style rules and editors do not need to know the details of the tables when they design the collective style specification.

4.4 Problems

Applying a topological specification to a table is straightforward. However, applying a style specification is another story. Many problems arise when applying a group of style rules to a table. We discuss three key problems: style conflict, the side effects of layout-oriented style rules, and the dynamic change of spacing.

4.4.1 Style conflict

We do not have to specify style rules for all components of a table. A component can inherit the style rules of one of its super-components or the default style rules. For example, a cell that holds a label can inherit the style rules of the label's category and the cell's region (stub, boxhead, or stub head); a cell that holds an entry can inherit the style rules of any entry set that contains the entry or the style rules of the cell's row and column. Thus, we need to find approaches to solve style inheritance. If we were able to use a tree structure to describe the relationships among the tabular components, we would define a priority order for style inheritance based on single inheritance. There are, however, multiple inheritances in a table. For example, a cell belongs to its row and column, which do not contain each other completely; thus, it can inherit style rules from both the row and the column. Therefore, the approaches for style inheritance should handle multiple inheritance. Three approaches can be used to make the decision:

1. Combine the style rules of all super-objects

   In this approach, we attempt to find style rules that satisfy all the style rules from all the super-objects. For example, italic Roman is the result of combining Roman and italic. There may not be, however, such a simple solution for all the style rules. For example, there is no suitable font that is the result of combining Roman and Courier.

2. Use the style rules of the super-object with the highest priority

   In this approach, either the tabular system or the user determines which super-object has the highest priority. Although we may define a realizable solution without user intervention, there are always cases that cannot meet users' expectations. This approach does not allow an object to inherit the combination of the style rules of its super-objects, which is similar to the way that C^{++} handles multiple inheritance: a subobject can inherit a method from a specific super-object, but it cannot inherit the combination of the methods in all super-objects.

3. Combine the previous two approaches

   First, we try to combine the style rules of all super-objects. Whenever there is no satisfactory combination, we use the style rules of the super-object with the highest priority. This approach overcomes the shortcomings of the previous two approaches.

Beach's system provides style rules for columns and rows and allows a cell to inherit the style rules of its column and row. Therefore, his system also needs to handle multiple inheritance. Beach adopted the first approach to solve style conflicts. He didn't, however, discuss the case in which there is no solution for the combination of multiple style rules. Since Vanoirbeek's system models tables as a tree structure, it provides only style rules for the objects in a tree structure; thus, this system does not have the problem of multiple inheritance. Our system adopts the third approach to handle style conflicts. We describe our approach in more detail in Chapter 6.

4.4.2 Side effects of layout-oriented style rules

From previous examples, we have seen that layout-oriented style rules may be applied to unexpected items after changing the topology of a table. This may happen whenever we specify layout-oriented style rules for logical components. To avoid these unpleasant side effects, we encourage users to specify content-oriented style rules for logical components. We can use three methods to handle the problem of layout-oriented style rules:

1. We do not change the style rules, but provide commands to remove layout-oriented style rules. In this case, users are responsible for the removal of old layout-oriented style rules and for the specification of new rules.

2. We remove or automatically suppress all layout-oriented style rules once the topology is changed. Therefore, users have to specify new style rules for the new topology.

3. We attempt to adjust the style rules after a topological change. For some changes, such as transposition, we can easily adjust the style rules. For other changes, however, we cannot adjust the style rules so easily. Since items in a block may be separated in multiple blocks after changing topology, a style rule for a block in the old topology needs to be replaced by multiple style rules for different blocks in the new topology.

We adopt the first approach in our tabular editor since it gives users the power to remove or to keep the layout-oriented style rules after changing the topology.

4.4.3 Dynamic change of spacing

To achieve an aesthetic layout, line and separation spacing should depend on the font size used to present the items. If we use a larger (or smaller) font to present a table, this spacing should be larger (or smaller) as well. We can use one of the following three approaches to handle this problem:

1. Users must change the spacing whenever they change a font size. They may not know, however, how much spacing is appropriate for a well-designed presentation.

2. We change the spacing to the appropriate values for well-designed presentation whenever the font sizes are changed. If users really do not like the new spacing, they can change it.

3. We provide two kinds of spacing: relative and absolute. Relative spacing is proportional to the font size of an object and absolute spacing is fixed. Users can select either kind according to their requirements.

The third approach is the best solution because it does not require respecification of the spacing after changing font size. Our editing system, however, uses the second approach because we had not developed the third approach when implementing the system.

4.5 Expressiveness of the presentational model

The topological rules in the presentational model allow only the arrangement of labels in the stub and the boxhead. This approach forces users to follow the guideline we gave in Section 1.2.2; namely, place the most frequently referenced items to the left or top of a table. Experiments [Wri68] have proved that readers tend to ignore the labels that are put in the body and consider them as entries. Thus, the presentation model does not allow a user to specify a topology in which some labels are placed in the table body, such as in Table 4.15. In Table 4.15, the apartment numbers in boldface are labels and they are placed in the body to shorten the table length. In some large tables, the labels in the boxhead are replicated many times in the body to help users to locate items faster. Table 4.16 shows the phosphorus loadings to the Great Lakes, from 1976 to 1982. Notice that the labels in the boxhead are repeated three times for each lake in the stub. After users locate a lake, they can immediately search for a year in the same row. The replication saves eye-traveling time between the the boxhead and the located row. Our presentational model is also unable to specify this kind of topological arrangement.

The presentational model enables users to specify styles from different viewpoints: general, logical, and layout. The presentational-oriented style rules, which control the general appearance of tables, are usually specified as collective style rules for a set of tables. The content-oriented style rules, which specify style for the logical components,

Table 4.15: Apartments at 31 Eleanor Drive, Nepean.

Floor number	Number of rooms	Size (ft^2)	Exposure	Number of rooms	Size (ft^2)	Exposure
		Apt1			**Apt2**	
1	1	700	West	1	700	West
		Apt3			**Apt4**	
1	1	700	West	1	700	East
		Apt5			**Apt6**	
1	1	700	East	1	700	East
		Apt7			**Apt8**	
2	2	1050	West	2	1050	West
		Apt9			**Apt10**	
2	2	1050	East	2	1050	East
		Apt11			**Apt12**	
3	3	1550	West	3	1550	West

Table 4.16: Phosphorus loadings to the Great Lakes, 1976 to 1982.

Lake Erie	**1976**	**1977**	**1978**	**1979**	**1980**	**1981**	**1982**
Point source	6,006	5,832	4,631	2,890	2,452	1,898	1,455
Non-Point source							
Tributary	7,211	6,545	12,874	6,241	9,773	6,745	9,154
Atmospheric	1,119	1,119	879	1,550	1,550	729	660
Total load	14,336	13,496	18,384	10,681	13,773	9,372	11,269
Target load	14,606	14,606	11,000	11,000	11,000	11,000	11,000
Lake Ontario	**1976**	**1977**	**1978**	**1979**	**1980**	**1981**	**1982**
Point source	2,119	2,594	2,030	2,419	2,122	1,818	1,643
Non-Point source							
Tributary	4,490	2,970	2,899	3,200	3,069	2,435	3,318
Atmospheric	473	623	764	311	311	328	600
Total load	7,082	6,187	5,693	5,930	5,502	4,581	4,961
Target load	6,072	6,072	5,000	5,000	5,000	5,000	5,000
All takes	**1976**	**1977**	**1978**	**1979**	**1980**	**1981**	**1982**
Point source	9,595	9,802	7,739	6,252	5,568	4,536	3,893
Non-Point source							
Tributary	21,248	16,017	24,517	18,432	20,631	17,750	21,500
Atmospheric	5,433	5,583	7,284	11,157	11,157	2,481	3,393
Total load	36,276	31,402	39,540	35,841	37,356	24,767	28,786
Target load	32,562	36,562	31,360	31,360	31,360	31,360	31,360

enable users to specify style independently of a table's topology. The layout-oriented style rules, which specify style for the layout components, provide the traditional way to specify style based on the row-column structure. The formatting attributes for various style rules were carefully chosen according to the guidelines we gave in Section 1.2.3 and as a result of the examination of different kinds of tables. We offer the styles that are commonly provided by other systems, such as various typographic options for items, commonly-used styles for rules, sufficient alignment options, and different methods of spanning items. We also provides some styles that are seldom provided by other tabular composition systems, for example, grouping items with rules or white space and arranging items in the stub in cut-in or indented styles. However, the presentational model cannot specify all styles observed in all tables. For example, we do not handle oblique lines; thus, we are unable to specify a table in which the headings of the categories in both dimensions are put in the stub head, separated by an oblique line. We allow only horizontal typesetting of text, vertical typesetting is not provided. We are not able to use graphical elements to highlight visual presentations; for instance, using horizontal or vertical braces to group items, or using arrows to strengthen the effect of spanning items.

We also did experiments to measure how well the presentational model can be applied to tables in the real world. We classified the tables from the books used in the experiment for the abstract model described in Section 2.4. Table A.2 in Appendix A reveals that the model can be used to specify the topology of 94 percent of the tables in these books and to specify the style of 97 percent of the tables. From these experiments, we see that our presentational model matches the real-world situation quite well.

Chapter 5

Formatting

An abstract table specifies only the logical structure of a table, it ignores the topological and typographical attributes, whereas a concrete table is a visualization of an abstract table in two dimensions. After applying a topological specification and a style specification to an abstract table, we generate a *grid structure*, an intermediate form between an abstract table and a concrete table, and size constraints for the columns and rows of the grid structure. The *formatting process* determines the physical dimension of a grid structure that satisfies the size constraints. Many factors contribute to the complexity of the formatting process. We focus on tabular formatting that provides automatic line breaking and allows size constraints expressed as linear equalities or inequalities, but does not provide objective functions. We first prove that the complexity of tabular formatting is NP-complete, and then we present an exponential-time algorithm that can solve the formatting problem in polynomial time for many tables.

5.1 Complexity of tabular formatting

The following three factors contribute to the complexity of tabular formatting:

1. The method of handling the line breaking of text within a table cell: fixed or automatic.

Table 5.1: The complexity of tabular formatting.

Line breaks	Size constraints	Objective functions			
		None	Diameter	Area	White space
Fixed	None	P[1]	P?[3]	P?[3]	P?[3]
	Linear equality or inequality	P?[3]	P[1]	P?[3]	P?[3]
	Nonlinear expression	?	?	?	?
Automatic	None	P?[3]	?	?	?
	Linear equality or inequality	NPC[2]	NPC?[3]	NPC?[3]	NPC?[3]
	Nonlinear expression	?	?	?	?

[1] Proved by Richard Beach [Bea85].
[2] See Theorem 5.1.
[3] These results are conjectured; see the report of Wang and Wood [WW96] and the discussion in Chapter 7.

2. The kinds of size constraints for the columns and rows: none, linear equalities or inequalities, or non-linear expressions.

3. The objective function that evaluates the quality of a tabular layout: none, minimal diameter, minimal area, or minimal white space.

Based on previous research and our current work, we list the complexity of tabular formatting for different combinations of the restrictions in Table 5.1, where P denotes polynomial-time solvable and NPC denotes NP-complete.

As far as we know, Beach is the only person who has discussed the computational complexity of tabular formatting. In his PhD thesis [Bea85], Beach presented a tabular formatting problem, RANDOM PACK, that arranges a set of unordered table entries into minimum area and proved that RANDOM PACK is NP-complete. Because of the random positioning of the table entries, RANDOM PACK does not produce pleasing and readable tables that clearly convey the logical structure. Beach also presented another problem, GRID PACK, that formats a set of table entries assigned to lie between particular row and column grid coordinates within the table and proved that GRID PACK is polynomial-time solvable. GRID PACK, however, assumes that the width and the height of the table entries are fixed; thus only fixed line breaks are allowed. Although Beach also allowed size constraints expressed as linear equalities or inequalities in his table model, he did not include the size constraints in RANDOM PACK and GRID PACK. The designers of TAFEL MUSIK have designed an exponential-time algorithm for tabular formatting that provides automatic line breaking, allows size constraints expressed as linear equalities and inequalities, and considers objective functions. However, they have analyzed neither the complexity of tabular formatting nor the running time of their algorithm.

We present a tabular formatting problem with restrictions on the three factors listed above. Automatic line breaking is important and useful for tabular formatting. It is also important to allow users to control the selection of the dimensions of columns and rows for a table. We simplify tabular formatting without losing these features.

We first disregard objective functions. The size constraints, we believe, play a more important role than the objective function in the selection of the final layout for the following reasons:

1. A layout that is optimal with respect to an objective function does not always provide the most appropriate layout.

   An optimal solution may make one column too narrow and another too wide, or generate a table with an unacceptable aspect ratio. We need to specify size constraints to avoid such pathological cases.

2. Users are more concerned about size constraints than they are about objective functions.

   Users tend to care more about the sizes of tabular components, Such as whether a table can be placed inside a region of a given width and height, whether the proportions of the sizes among components in a table are appropriate, and whether the proportions between a table and the surrounding objects are appropriate. For example, the width of a table should not be wider than the page size, the widths of different columns should not differ too much, and the width of a table should not be too narrow if the table is placed between wide objects. Once these requirements are satisfied, it really does not matter too much whether a table occupies the smallest space or contains the least white space. Such requirements are specified by size constraints, rather than by objective functions.

3. We do not always need an optimal solution.

   In most cases, a solution that is close to optimal is good enough. Users can adjust the size constraints to approach a solution that is closer to the optimal solution for a particular table. For example, we can specify a thinner column to reduce the white space in a column or specify a thinner or shorter table to reduce the area or diameter of a table. .

When we do not use objective functions, we can select any layout that satisfies the size constraints. We call this strategy an *if-satisfied-then-taken strategy.*

We next simplify the size constraints. We consider only the size constraints that can be expressed as linear equalities or inequalities that contain only variables for column widths or variables for row heights, but not for both. Size constraints expressed with these kinds of linear equalities or inequalities are called *homogeneous.* For example, suppose we use w_j to denote the width of the jth column and h_i to denote the height of the ith row, then $w_1 + 2w_3 \leq 100$ and $h_2 - 3h_4 = 0$ are homogeneous size constraints, whereas $h_1 + w_2 \geq 500$ is not. If a size constraint contains only variables for column widths, it is called a *width constraint*, and if a size constraint contains only variables for row heights, it is called a *height constraint.*

Finally, we fix the direction of typesetting. We assume that the text is read row by row from top to bottom and either from left to right or from right to left within each row. Given a rectangular region, we first horizontally fill the region with text that is as wide as possible. If the region is not wide enough for all the text, we break the text into lines and vertically fill the region. We can easily extend our model to allow text that is read column by column, but our assumption simplifies the presentation.

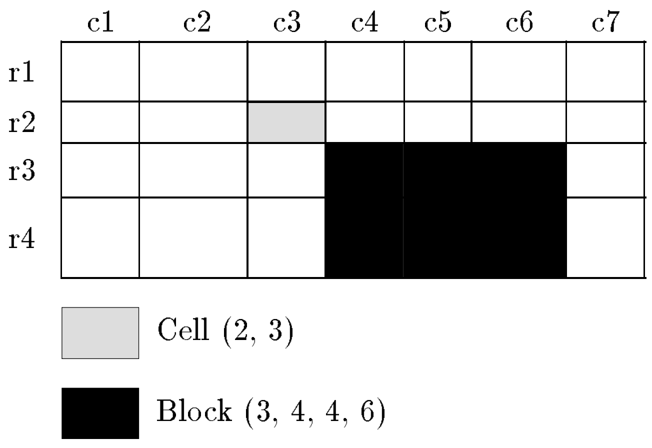


Figure 5.1: A 4×7 grid.

5.2 Grid structure

A *grid structure* describes the placement of tabular items in a two-dimensional lattice. We inherited this concept from Beach's system [Bea85] and make some changes. A grid structure consists of two components: a grid and a set of non-overlapping items that are placed on the grid.

An *$m \times n$ grid* is a planar integer lattice with m rows and n columns. For example, Fig. 5.1 shows a 4×7 grid. The rows are identified from top to bottom by $1, 2, \ldots, m$ and the columns are identified from left to right by $1, 2, \ldots, n$. The intersection of a row and a column is called a *cell* and the cell that is the intersection of the ith row and the jth column is identified by (i, j). A *block* is a rectangular region that completely surrounds a set of cells, and it is identified by (t, l, b, r), where (t, l) is its upper left cell and (b, r) is its lower right cell.

An *item* is an object that is placed in a block of a grid. The content of an item can be a string, a number, a textual object, a fixed-sized picture and image, or a table. The *size function* of an item is a decreasing step function that describes the line-breaking characteristics of the item for a particular output device. It takes a width as its argument and returns the height of the item when the item is typeset within the given width. We can assume that both the width and the height are integers. The characteristics of a size function for a textual item are shown in Fig. 5.2, from which we can see that:

1. The height of an item is monotonically non-increasing as the width increases, because an item does not require more lines when the width increases.

2. The height of an item does not change continuously. When we increase the width of an item, the height is unchanged until the width is large enough to allow the first non-broken unit in a line to move to the previous line. Thus, at some specific widths (break points b_2, b_3, and b_4 in Fig. 5.2), the height of an item decreases. For the range of widths between two consecutive break points, the height of an item is constant.

3. There is a minimal width for an item (b_1 in Fig. 5.2). The minimal width should be the width of the longest non-broken unit in the item. We designate the minimal width as a special break point. The height is maximized when the width is minimized.

4. There is a maximal width for an item (b_4 in Fig. 5.2). The maximal width is the width of the item without any line breaking. The height is minimized when the width is maximal.

These characteristics also hold for tables, mathematical equations, and fixed-sized pictures and images. They do not, however, hold for variable-sized pictures and images, because the height of a picture or an image also increases when the width increases. We use a *step* to denote the range of widths in $[b_k, b_{k+1})$, where b_k and b_{k+1} are two adjacent break points or b_k is the maximal break point and b_{k+1} is $+\infty$. The lower bound of a step is called a *step head* and the upper bound of a step, which is $b_{k+1} - 1$ if the step is $[b_k, b_{k+1})$ or $+\infty$ if the step is $[b_k, +\infty)$, is called a *step tail*. A size function returns the same height for all the widths in a step. In Fig. 5.2, the size function consists of four steps $[b_1, b_2)$, $[b_2, b_3)$, $[b_3, b_4)$, and $[b_4, +\infty)$.

We can specify an item by a six-element tuple (t, l, b, r, ξ, ψ), where (t, l, b, r) is the block in which the item is placed in the grid, ξ is its size function, and ψ is the set of step heads for ξ. For convenience, we need to define some more notation. If s is a step,

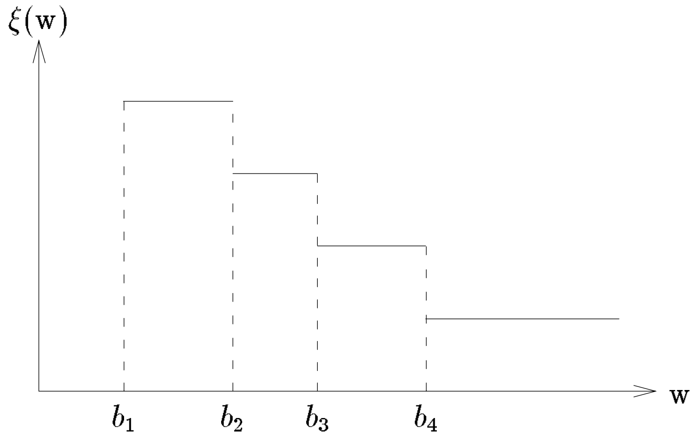


Figure 5.2: The characteristics of a size function.

we use $s.head$ to denote its head and $s.tail$ to denote its tail. If ψ is a set of step heads for a size function, we use $\psi[min]$ to denote the minimal step head and $\psi[max]$ to denote the maximal step head.

5.3 The tabular formatting problem

The goal of tabular formatting is to calculate the final geometric positions of all the tabular components. By applying a topological specification and a style specification to an abstract table, we are able to generate an $m \times n$ grid, a set of items placed on the grid, and a set of size constraints. After that, we need to determine the physical dimensions of the columns and the rows in the grid so that all the size constraints are satisfied and all the items are placed completely inside the block they occupy. We can formally define *Tabular Formatting* as follows:

INSTANCE:	An $m \times n$ grid, r non-overlapping items: $o_k = (t_k, l_k, b_k, r_k, \xi_k, \psi_k)$ $(1 \leq k \leq r)$, and s size constraints: $e_1, e_2, \ldots, e_s$.
QUESTION:	Are there $n + m$ integers $w_1, w_2, \ldots, w_n$ and $h_1, h_2, \ldots, h_m$ such that 1. $W = w_1, w_2, \ldots, w_n$ satisfy all width constraints among $e_1, e_2, \ldots, e_s$; 2. $H = h_1, h_2, \ldots, h_m$ satisfy all height constraints among $e_1, e_2, \ldots, e_s$; 3. $\forall o_k (1 \leq k \leq r)$, $\sum_{p=l_k}^{r_k} w_p \geq \psi_k[min]$ and $\xi_k(\sum_{p=l_k}^{r_k} w_p) \leq \sum_{q=t_k}^{b_k} h_q$.

The $w_j (1 \leq j \leq n)$ are the column widths and the $h_i (1 \leq i \leq m)$ are the row heights of the grid. The first two conditions ensure that $w_j (1 \leq j \leq n)$ and $h_i (1 \leq i \leq m)$ satisfy all the size constraints. The third condition ensures that the width of the block for each item is at least the minimal width of the item and the height of the block should be sufficient to hold the item when it is typeset within the width of the block. If $w_j (1 \leq j \leq n)$ and $h_i (1 \leq i \leq m)$ satisfy all three conditions for an instance, we say that the instance has *solution* (W, H). If they satisfy only the third condition for an instance, we say the instance has *layout* (W, H).

For an instance of the tabular formatting problem, there may be more than one solution. Suppose we have an instance that consists of a 5×3 grid and the 13 items shown in Table 5.2. The size constraints for this instance are:

$$290pt \leq w_1 + w_2 + w_3 \leq 380pt$$
$$h_1 + h_2 + h_3 + h_4 + h_5 \leq 350pt$$
$$w_3 \geq 120pt.$$

there are several solutions for this instance. One solution, shown in Table 5.2, is:

$$w_1 = 65pt, \quad w_2 = 68pt, \quad w_3 = 230pt,$$
$$h_1 = 31pt, \quad h_2 = 47pt, \quad h_3 = 45pt, \quad h_4 = 47pt, \quad h_5 = 45pt$$

and another solution, shown in Table 5.3, is:

$$w_1 = 65pt, \quad w_2 = 105pt, \quad w_3 = 125pt,$$
$$h_1 = 33pt, \quad h_2 = 62pt, \quad h_3 = 74pt, \quad h_4 = 76pt, \quad h_5 = 74pt.$$

Table 5.2: The tournament schedule.

Activity	Final Entry Date	Starting Date, Location, Times
Men's & Women's squash	Monday, Jan. 23, 1:00pm, PAC 2039	Prelim. Sat. Jan. 28, Finals Sun. Jan. 29, 11:00am-6:00pm, Court 1068-1073, PAC
Singles Tennis		Prelim. Sun. Feb. 5, 10:00am-6:00pm, Finals Sun. Feb. 12, 10:00am-6:00pm, Waterloo Tennis Club
Mixed Volleyball	Friday, Mar. 3, 1:00pm, PAC 2039	Prelim. Wed. Mar. 8, 8:00pm-11:30pm, Finals Mon. Mar. 13, 8:00pm-11:30pm, Main Gym, PAC
Men's & Co-Rec Broomball		Prelim. Fri. Mar. 17, 12:00pm-5:00pm, Finals Sat. Mar. 18, 3:00pm-1:00am, Columbia Icefield

5.4 Tabular formatting is NP-complete

We show that *Tabular Formatting* is NP-complete by reducing it to *Subset Sum* [GJ79], which is known to be NP-complete. The definition of the *Subset Sum* is as follows:

INSTANCE:	A finite set A, a size $s(a) \in Z^+$, for each $a \in A$, and a positive integer B.
QUESTION:	Is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = B$.

For convenience, we abbreviate *Tabular Formatting* as TF and *Subset Sum* as SS.

Theorem 5.1 *TF is NP-complete.*

Proof: It is easy to see that TF $\in$ NP since we can check in polynomial time whether a given set of $n+m$ integers satisfies all three conditions.

We reduce SS to TF. Given an instance of SS, we first divide A into d nonempty subsets $A_1, A_2, \ldots, A_d$ by putting elements of the same size into the same subset. For

Table 5.3: The tournament schedule.

Activity	Final Entry Date	Starting Date, Location, Times
Men's & Women's squash	Monday, Jan. 23, 1:00pm, PAC 2039	Prelim. Sat. Jan. 28, Finals Sun. Jan. 29, 11:00am-6:00pm, Court 1068-1073, PAC
Singles Tennis		Prelim. Sun. Feb. 5, 10:00am-6:00pm, Finals Sun. Feb. 12, 10:00am-6:00pm, Waterloo Tennis Club
Mixed Volleyball	Friday, Mar. 3, 1:00pm, PAC 2039	Prelim. Wed. Mar. 8, 8:00pm-11:30pm, Finals Mon. Mar. 13, 8:00pm-11:30pm, Main Gym, PAC
Men's & Co-Rec Broomball		Prelim. Fri. Mar. 17, 12:00pm-5:00pm, Finals Sat. Mar. 18, 3:00pm-1:00am, Columbia Icefield

example, if $A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$, $s(a_1) = s(a_3) = s(a_5) = 3$, $s(a_2) = s(a_6) = 5$, and $s(a_4) = 2$, then A is partitioned into three subsets: $A_1 = \{a_1, a_3, a_5\}$, $A_2 = \{a_2, a_6\}$ and $A_3 = \{a_4\}$. We use $s(A_i)$ to denote the size of the elements in subset A_i and $|A_i|$ to denote the number of elements in A_i. Thus, SS is equivalent to this question: Are there d integers $z_1, z_2, \ldots, z_d$ such that $0 \le z_k \le |A_k|$ $(1 \le k \le d)$ and $\sum_{k=1}^{d}(z_k \times s(A_k)) = B$. We can construct an instance of TF from this version of SS as follows:

1. Let $m = n = d$.
2. Let $s = 2$ and the size constraints be:

$$\begin{array}{lcl} \sum_{j=1}^{n} w_j & = & B + \sum_{k=1}^{d} s(A_k), \\ \sum_{i=1}^{m} h_i & = & \sum_{k=1}^{d}((|A_k| + 1) \times s(A_k)) - B. \end{array}$$

3. Let $r = d$ and, for each k, $1 \le k \le r$, $o_k = (k, k, k, k, \xi_k, \psi_k)$, where

$$\psi_k = \{i \times s(A_k) \mid i = 1, 2, \ldots, |A_k| + 1\}$$

$$\begin{array}{lcll} \xi_k(x) & \text{is} & \text{undefined} & \text{if } x < s(A_k) \\ & = & (|A_k| + 1) \times s(A_k) & \text{if } s(A_k) \le x < 2 \times s(A_k) \\ & \vdots & & \\ & = & (|A_k| + 2 - i) \times s(A_k) & \text{if } i \times s(A_k) \le x < (i+1) \times s(A_k) \\ & \vdots & & \\ & = & 2 \times s(A_k) & \text{if } |A_k| \times s(A_k) \le x < (|A_k| + 1) \times s(A_k) \\ & = & s(A_k) & \text{if } x \ge (|A_k| + 1) \times s(A_k). \end{array}$$

From the previous example of an instance of SS, we can construct a 3×3 grid in which three items are to be placed in the cells along the diagonal (Fig. 5.3(a)). The size function for o_1 is shown in Fig. 5.3(b). It is should be clear that the construction takes polynomial time in the size of the instance.

Suppose that there is a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = B$. Then, there must be d integers $z_1, z_2, \ldots, z_d$ such that $0 \le z_k \le |A_k|$ $(1 \le k \le d)$ and $\sum_{k=1}^{d}(z_k \times s(A_k)) = B$. We let $w_j = (z_j + 1) \times s(A_j)$ $(1 \le j \le n)$ and $h_i = (|A_i| + 1 - z_i) \times s(A_i)$ $(1 \le i \le m)$. Now we prove that w_j $(1 \le j \le n)$ and h_i $(1 \le i \le m)$ satisfy the three conditions of TF.

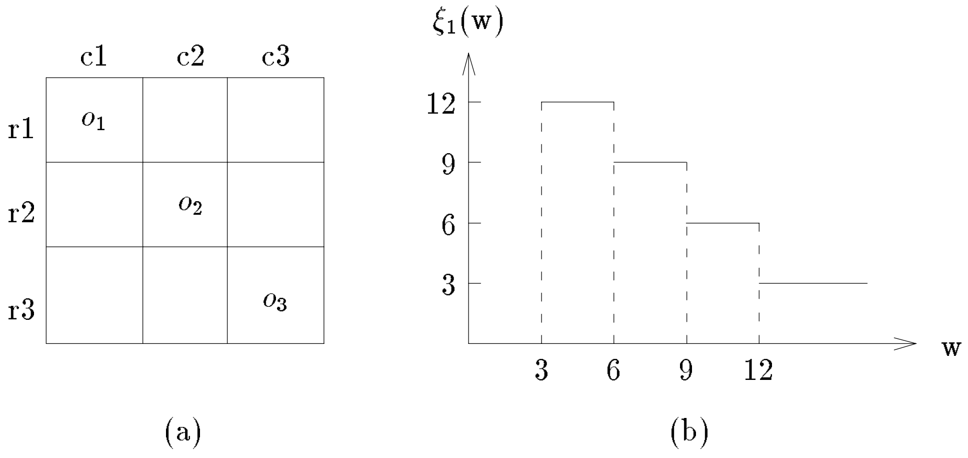


Figure 5.3: An example of constructing an instance of TF from an instance of SS.

First,

$$\begin{aligned}
\textstyle\sum_{j=1}^{n} w_j &= \textstyle\sum_{j=1}^{d}((z_j+1)\times s(A_j)) \\
&= \textstyle\sum_{k=1}^{d}(z_k \times s(A_k)) + \sum_{k=1}^{d} s(A_k) \\
&= B + \textstyle\sum_{k=1}^{d} s(A_k),
\end{aligned}$$

which implies that the first condition holds.

Second,

$$\begin{aligned}
\textstyle\sum_{i=1}^{m} h_i &= \textstyle\sum_{i=1}^{d}((|A_i|+1-z_i)\times s(A_i)) \\
&= \textstyle\sum_{k=1}^{d}((|A_k|+1)\times s(A_k)) - \sum_{k=1}^{d}(z_k \times s(A_k)) \\
&= \textstyle\sum_{k=1}^{d}((|A_k|+1)\times s(A_k)) - B,
\end{aligned}$$

which implies that the second condition holds.

Now, for each $o_k (1 \leq k \leq r)$,

$$\begin{aligned}
\textstyle\sum_{p=k}^{k} w_p &= w_k \\
&\geq s(A_k) \\
&= \psi_k[min],
\end{aligned}$$

and

$$\begin{array}{rcl}
\xi_k(\sum_{p=k}^{k} w_p) & = & \xi_k(w_k) \\
& = & \xi_k((z_k+1)\times s(A_k)) \\
& = & (|A_k|+1-z_k)\times s(A_k) \\
& = & h_k \\
& \leq & \sum_{q=k}^{k} h_q,
\end{array}$$

which implies that the third condition holds.

Therefore, the instance of TF has solution (W, H).

Conversely, if the instance of TF has a solution (W, H), then w_j $(1 \leq j \leq n)$ must fall in a step $s_j = [u_j \times s(A_j), t_j)$ of item o_j, where $1 \leq u_j \leq |A_j| + 1$ and t_j is either $(u_j + 1) \times s(A_j)$ or $+\infty$. We let $z_k = u_k - 1$ $(1 \leq k \leq d)$, in which case $0 \leq z_k \leq |A_k|$. We prove that $\sum_{k=1}^{d}(z_k \times s(A_k)) = B$ in two steps.

First,

$$\begin{array}{rcl}
\sum_{k=1}^{d}(z_k \times s(A_k)) & = & \sum_{k=1}^{d}((u_k-1)\times s(A_k)) \\
& = & \sum_{k=1}^{d}(u_k \times s(A_k)) - \sum_{k=1}^{d} s(A_k) \\
& \leq & \sum_{k=1}^{n} w_k - \sum_{k=1}^{d} s(A_k) \\
& = & B,
\end{array}$$

which implies that $\sum_{k=1}^{d}(z_k \times s(A_k)) \leq B$.

Second,

$$\begin{array}{rcl}
\sum_{k=1}^{d}(z_k \times s(A_k)) & = & \sum_{k=1}^{d}((|A_k|+1)\times s(A_k)) - \\
& & \quad (\sum_{k=1}^{d}((|A_k|+1)\times s(A_k)) - \sum_{k=1}^{m}(z_k \times s(A_k))) \\
& = & \sum_{k=1}^{d}((|A_k|+1)\times s(A_k)) - \sum_{k=1}^{m}((|A_k|+1-z_k)\times s(A_k)) \\
& = & \sum_{k=1}^{d}((|A_k|+1)\times s(A_k)) - \sum_{k=1}^{m}((|A_k|+2-u_k)\times s(A_k)) \\
& = & \sum_{k=1}^{d}((|A_k|+1)\times s(A_k)) - \sum_{k=1}^{m} \xi_k(u_k \times s(A_k)) \\
& = & \sum_{k=1}^{d}((|A_k|+1)\times s(A_k)) - \sum_{k=1}^{m} \xi_k(w_k) \\
& \geq & \sum_{k=1}^{d}((|A_k|+1)\times s(A_k)) - \sum_{k=1}^{m} h_k \\
& = & \mathrm{B},
\end{array}$$

which implies that $\sum_{k=1}^{d}(z_k \times s(A_k)) \geq B$.

Thus, combining the two inequalities, we have shown that $\sum_{k=1}^{d}(z_k \times s(A_k)) = B$. We can define a subset A' by choosing z_k elements from A_k $(1 \leq k \leq d)$, in which case $\sum_{a \in A'} a = \sum_{k=1}^{d}(z_k \times s(A_k)) = B$.

Therefore, TF is NP-complete. □

NP-complete problems do not have polynomial-time algorithms unless $P = NP$, which is considered unlikely. With this assumption, we can provide only an exponential-time algorithm for TF that solves every instance. We first present an exponential-time algorithm for TF in Section 5.5, and then we describe a polynomial-time greedy algorithm in Section 5.6 that partially solves TF for many common instances. Finally, in Section 5.7, we combine these two algorithms to obtain an algorithm that guarantees to solve TF completely and correctly and takes only polynomial time for many instances.

5.5 An exponential-time algorithm

The simplest way to solve TF is to check all the possible combinations of row heights and column widths. The first combination that satisfies all three conditions of TF is selected as a solution. Suppose we know the maximal range W_j for the width of the jth column and the maximal range H_i for the height of the ith row; then the number of possible combinations of row heights and column widths is

$$N = (\prod_{j=1}^{n} W_j) \times (\prod_{i=1}^{m} H_i).$$

Since we usually use small units such as points or even small fractions of a point to measure length in a formatting system, W_j and H_i may have values in the hundreds or even thousands. Increasing the values m and n leads to an exponential increase in the size of N. We can avoid examining all combinations of row heights and column widths by solving the size constraints to obtain row heights for given column widths. Once the column widths are fixed, the heights and widths of the items are also fixed; thus, we can use Beach's approach to find the row heights in polynomial time. Thus, we need to examine only

$$N = \prod_{j=1}^{n} W_j$$

```
Algorithm 1 TF_Exponential-Time_Algorithm(column_widths, row_heights): bool;

    var integer column_widths[1..column_number], row_heights[1..row_number];
begin
    integer pair current_steps[1..item_number];

    for current_steps := each step combination of all the items do
        if Find_Column_Widths(current_steps, column_widths) and
          Find_Row_Heights(current_steps, row_heights) then
              return(true);
        end if
    end for;

    return(false);
end
```

Figure 5.4: An exponential-time algorithm for TF.

combinations.

We can reduce this number further by taking advantage of the characteristics of size functions. Since the height of an item will be the same when it is typeset within the widths of a step, we need to test only one of the widths in a step. For each combination of steps, we can find the column widths and row heights by solving inequalities. Suppose that item o_j has K_j steps; then the number of combinations can be reduced to

$$N = \prod_{j=1}^{r} K_j.$$

N still increases at an exponential rate when most of the items have more then one step. In many tables, however, most of the items contain only one step. The number of combinations that used to be checked is not too large for these cases. Based on this approach, we design the exponential-time algorithm that completely solves TF shown in Fig. 5.4.

Based on a given step combination $C = \{s_1, s_2, \ldots, s_r\}$ of all the items, where s_k is a step of item o_k, `Find_Column_Widths` attempts to find column widths $w_j (1 \leq j \leq n)$ such that:

1. $w_j (1 \leq j \leq n)$ satisfy all the width constraints.
2. For each item $o_k = (t_k, l_k, b_k, r_k, \xi_k, \psi_k)$, $s_k.head \leq \sum_{p=l_k}^{r_k} w_p \leq s_k.tail.$

Similarly, `Find_Row_Heights` attempts to find row heights $h_i (1 \leq i \leq m)$ such that

1. $h_i (1 \leq i \leq n)$ satisfy all the height constraints.
2. For each item $o_k = (t_k, l_k, b_k, r_k, \xi_k, \psi_k)$, $\xi_k(s_k.head) \leq \sum_{q=t_k}^{b_k} h_q$.

`Find_Column_Widths` is **false** only when there are no column widths for the step combination and `Find_Row_Heights` is **false** only when there are no row heights for the step combination. Therefore, Algorithm 1 has a solution for a given step combination if and only if both `Find_Column_Widths` and `Find_Row_Heights` have a solution. We give pseudo code algorithms for `Find_Column_Widths` and `Find_Row_Heights` in Appendix B.

`Find_Column_Widths` and `Find_Row_Heights` find solutions by solving a set of linear equalities and inequalities. There is an algorithm for this problem based on the simplex method that runs in $O(t^3)$ time, where t is the number of equalities and inequalities [Dan63]. Moreover, the algorithm guarantees that the sum of the values of the variables in the equalities and inequalities is minimum. Therefore, both `Find_Column_Widths` and `Find_Row_Heights` can find solutions in $O((r+s)^3)$ time, where r is the number of items and s is the number of size constraints. The total running time of Algorithm 1 is then

$$O((\prod_{j=1}^{r} K_j) \times (r+s)^3),$$

where K_j is the number of steps for item o_j.

We need to introduce some notation before we prove that Algorithm 1 completely and correctly solves TF. Suppose C is a step combination for an instance of TF. We use WIE(C) to denote the set of equalities and inequalities generated by `Find_Column_Widths` for C and we use HIE(C) to denote the set of equalities and inequalities generated by

`Find_Row_Heights` for C. Clearly WIE(C) has solutions if and only if `Find_Column_Widths` has a solution $w_j(1 \leq j \leq n)$, and HIE(C) has solutions if and only if `Find_Row_Heights` returns a solution $h_i(1 \leq i \leq m)$. If $w_j(1 \leq j \leq n)$ satisfy only the inequalities for item sizes (Condition 2) in WIE(C), then $w_j(1 \leq j \leq n)$ is called *a layout* of WIE(C); if $h_i(1 \leq i \leq m)$ satisfy only the inequalities for item sizes (Condition 2) in HIE(C), then $h_i(1 \leq i \leq m)$ is called a *a layout* of HIE(C). We are now ready to prove the following results.

Lemma 5.2 *Given an instance I of TF, there is a solution for I if and only if there is a step combination C such that WIE(C) and HIE(C) each have a solution.*

Proof: Suppose instance I has a solution (W, H); then, for each $o_k = (t_k, l_k, b_k, r_k, \xi_k, \psi_k)$, we can find a step s_k such that $s_k.head \leq \sum_{p=l_k}^{r_k} w_p \leq s_k.tail$. Let step combination $C = \{s_1, s_2, \ldots, s_r\}$. Since (W, H) is a solution of I, then $w_j(1 \leq j \leq n)$ must satisfy all the width constraints and $h_i(1 \leq i \leq m)$ must satisfy all the height constraints. Moreover, for each item o_k, $\sum_{q=t_k}^{b_k} h_q \geq \xi_k(\sum_{p=l_k}^{r_k} w_p)$. Since $\sum_{p=l_k}^{r_k} w_p$ is inside step s_k, we have $\xi_k(\sum_{p=l_k}^{r_k} w_p) = \xi_k(s_k.head)$; thus, $\sum_{q=t_k}^{b_k} h_q \geq \xi_k(s_k.head)$. Thus, $w_j(1 \leq j \leq n)$ is a solution of WIE(C) and $h_i(1 \leq i \leq m)$ is a solution of HIE(C). Therefore, WIE(C) and HIE(C) each have a solution.

Conversely, suppose there is a step combination $C = \{s_1, s_2, \ldots, s_r\}$ such that WIE(C) has a solution w_j $(1 \leq j \leq n)$ and HIE(C) has a solution $h_i(1 \leq i \leq m)$. Then, w_j $(1 \leq j \leq n)$ must satisfy all the width constraints and h_i $(1 \leq i \leq m)$ must satisfy all the height constraints. Moreover, for each item $o_k = (t_k, l_k, b_k, r_k, \xi_k, \psi_k)$, $s_k.head \leq \sum_{p=l_k}^{r_k} w_p \leq s_k.tail$ and $\sum_{q=t_k}^{b_k} h_q \geq \xi_k(s_k.head)$. Because $\sum_{p=l_k}^{r_k} w_p$ is inside step s_k, we know that $\xi_k(\sum_{p=l_k}^{r_k} w_p) = \xi_k(s_k.head)$. Thus, $\sum_{p=l_k}^{r_k} w_p \geq \psi_k[min]$ and $\sum_{q=t_k}^{b_k} h_q \geq \xi_k(\sum_{p=l_k}^{r_k} w_p)$. Therefore, (W, H) is a solution for I, where $W = w_1, \ldots, w_n$ and $H = h_1, \ldots, h_m$. □

Theorem 5.3 *Given an instance I of TF, there is a solution for I if and only if Algorithm 1 has a solution.*

Proof: If there is a solution (W, H) for I, then, by Lemma 5.2, there must be a step combination C such that both WIE(C) and HIE(C) have solutions. Thus, `Find_Column_Widths` has a solution $w'_j(1 \leq j \leq n)$ for C and `Find_Row_Heights` has a solution $h'_i(1 \leq i \leq m)$ for C. If Algorithm 1 has not found a solution before C, then it will terminate with C and return solution (W', H'), where $W' = w'_1, \ldots, w'_n$ and $H' = h'_1, \ldots, h'_m$. Therefore, Algorithm 1 must find a solution.

Conversely, if Algorithm 1 finds a solution (W, H), then it must terminate after checking a step combination C for which `Find_Column_Widths` finds $w_j(1 \leq j \leq n)$ and `Find_Row_Heights` finds $h_i(1 \leq i \leq m)$. Thus, both WIE(C) and HIE(C) have at least one solution. Therefore, by Lemma 5.2 there is a solution for the instance. □

5.6 A polynomial-time greedy algorithm

Algorithm 1 takes exponential time, in most cases, to find a solution for TF. Most tables, however, usually have few size constraints. For many such cases, we are able to find a solution in polynomial time by taking advantage of the monotonicity property of size functions. Given an instance I of TF, the monotonicity property of size functions enables us to generate a list $L_I = C_1, C_2, \ldots, C_z$ of step combinations, where $C_u = \{s_1^u, s_2^u, \ldots, s_r^u\}(1 \leq u \leq z)$, that satisfies the following properties:

Property 1 For the first step combination C_1, WIE(C_1) must have at least one solution.

Property 2 For each item $o_k = (t_k, l_k, b_k, r_k, \xi_k, \psi_k)$, s_k^{u+1} is either the same as s_k^u or the successor of s_k^u; thus, $\xi_k(s_k^{u+1}.head) \leq \xi_k(s_k^u.head)$.

Property 3 There is at least one item such that its step in C_{u+1} is larger than its step in C_u.

Property 4 In the last step combination C_z, for each k, $1 \leq k \leq r$, s_k^z is the largest step of item o_k.

Property 5 For each step combination $C_u(1 \leq u \leq z)$, there is a layout $w_j^u(1 \leq j \leq n)$ for WIE(C_u) and a layout $h_i^u(1 \leq i \leq m)$ for HIE(C_u).

By checking only the step combinations in this list, we may be able to determine whether there is a solution for the instance. Before we describe how we generate a list of step combinations that satisfies Properties 1–5 for an instance, we prove that these properties enable us to obtain a polynomial-time algorithm for TF that returns solutions for many tables.

Lemma 5.4 *If there is a solution for an instance I of TF, then there is a step combination that satisfies Property 1.*

Proof: If there is a solution (W, H) for I, by Lemma 5.2 there must be a step combination C such that both WIE(C) and HIE(C) have at least one solution. Thus, C is a step combination that satisfies Property 1. □

An implication of Lemma 5.4 is that if there is no step combination C such that WIE(C) has at least one solution, then there is no solution for the instance. Given an instance I of TF, if there is a step combination C such that WIE(C) has a solution, then we are able to generate a list L_I of step combinations that satisfies Properties 1–5. For these instances, we obtain the following results.

Lemma 5.5 *The number of step combinations in L_I is at most $\sum_{j=1}^{r} K_j$, where K_j is the number of steps for item o_j.*

Proof: By Properties 2 and 3, there is at least one item such that its step in C_{u+1} is the successor of its step in C_u. Since there are only K_j steps for item $o_j (1 \leq j \leq r)$, the number of step combinations in L_I is at most $\sum_{j=1}^{r} K_j$. □

Lemma 5.6 *If there is a solution for instance I, then there is a solution for HIE(C_z).*

Proof: Suppose there is a solution for the instance. Then, by Lemma 5.2, there is a step combination $C = \{s_1, s_2, \ldots, s_r\}$ such that both WIE(C) and HIE(C) have at least one solution. Suppose $h_i (1 \leq i \leq m)$ is a solution of HIE(C). Since C_z consists of the largest steps for all the items (Property 4), s_k must be either the same step as s_k^z or a smaller step than s_k^z; thus, $\xi_k(s_k^z.head) \leq \xi_k(s_k.head)$. Since $h_i (1 \leq i \leq m)$ is a solution of HIE(C), it must satisfy all the height constraints. Moreover, for each item $o_k = (t_k, l_k, b_k, r_k, \xi_k, \psi_k)$, $\sum_{q=t_k}^{b_k} h_q \geq \xi_k(s_k.head)$; thus, $\sum_{q=t_k}^{b_k} h_q \geq \xi_k(s_k^z.head)$. Therefore, $h_i (1 \leq i \leq m)$ is also a solution of HIE(C_z). □

Lemma 5.7 *If there is a solution for HIE(C_u), then there is a solution for HIE(C_v), where $u \leq v \leq z$.*

Proof: By Property 2, for each item $o_k = (t_k, l_k, b_k, r_k, \xi_k, \psi_k)$, s_k^v is either the same as s_k^u or larger than s_k^u; thus, $\xi_k(s_k^v.head) \leq \xi_k(s_k^u.head)$. If HIE($C_u$) has a solution $h_i (1 \leq i \leq m)$, then $h_i (1 \leq i \leq m)$ satisfy all the height constraints and, for each item o_k, $\sum_{q=t_k}^{b_k} h_q \geq \xi_k(s_k^u.head) \geq \xi_k(s_k^v.head)$; thus, a solution for HIE(C_u) is also a solution for HIE(C_v). □

Theorem 5.8 *Instance I has a solution, if the following three conditions all hold:*

1. *There is a step combination C_1 such that WIE(C_1) has a solution.*
2. *There is a solution for HIE(C_z).*
3. *If $h(1 \leq h \leq z)$ is the largest integer such that WIE(C_h) has a solution and $l(1 \leq l \leq z)$ is the smallest integer such that HIE(C_l) has a solution, then $l \leq h$.*

Proof: If there is no step combination C_1 such that WIE(C_1) has a solution, then, by Lemma 5.4, there is no solution for the instance. Similarly, if there is no solution for HIE(C_z), by Lemma 5.6, there is no solution for the instance. Since the first two conditions hold, both l and h in the third condition are well defined. By Lemma 5.7, there is a solution for HIE(C_h) because HIE(C_l) has a solution and $l \leq h$. Since WIE(C_h) also has a solution, by Lemma 5.2, there is a solution for the instance. □

Based on Theorem 5.8, Table 5.4 indicates that we have three possible conclusions while checking the step combinations in L_I (see Table 5.4):

- We have found a solution for I
- We are sure there is no solution for I
- We are uncertain whether there is a solution for I

It is clear that we get an uncertain answer only if L_I satisfies the first two conditions of Theorem 5.8 and fails the third condition. When attempting to solve WIE(C_u) or

Table 5.4: The conditions that determine if there is a solution for an instance.

	WIE(C_1) has solution	Cannot find C_1
HIE(C_z) has solution	$l \leq h$ Yes $l > h$ Uncertain	No
HIE(C_z) has no solution	No	No

HIE(C_u) ($1 \leq u \leq z$), we get two possible results: there is a solution (Yes) or there is no solution (No). An *assignment* of L_I is a combination of the two possible results (yes or no) for each WIE(C_u) and HIE(C_u) ($1 \leq u \leq z$). The following theorem gives the proportion of the assignments of L_I that generate an uncertain answer for instance I.

Theorem 5.9 *If $L_I = C_1, C_2, \ldots, C_z$ is such that WIE(C_1) and HIE(C_z) each have a solution, then the proportion of the assignments of L_I that give an uncertain answer is*

$$\frac{2^{z-1}-1}{z2^{z-1}} \approx \frac{1}{z},$$

where z is the number of step combinations in L_I.

Proof: We need to count the total number of assignments of L_I and the number of assignments that give an uncertain answer. The results for WIE(C_1) and HIE(C_z) are certain (both are 'Yes') and the results for WIE(C_u) ($2 \leq u \leq z$) and HIE(C_u) ($1 \leq u \leq z-1$) are uncertain (see Table 5.5). Thus, we have at most $2^{2(z-1)}$ possible assignments of L_I. By Lemma 5.7, however, if l is the smallest integer such that HIE(C_l) has a solution, then HIE(C_{l+1}), HIE(C_{l+2}), ..., HIE(C_z) also have solutions. Thus, there are only z possible combinations of the results for HIE(C_1), HIE(C_2), ..., HIE(C_{z-1}). Therefore, the total number of assignments of L_I is $z2^{z-1}$. For each of these assignments, we assume that $h (1 \leq h \leq z)$ is the largest integer such that WIE(C_h) has a solution and $l (1 \leq l \leq z)$ is the smallest integer such that HIE(C_l) has a solution. Since L_I satisfies the

Table 5.5: The possible assignments of L_I.

L_I	WIE	HIE
C_1	Yes	?
C_2	?	?
.	.	.
.	.	.
.	.	.
C_k	?	?
.	.	.
.	.	.
.	.	.
C_z	?	Yes

first two conditions of Theorem 5.8, an assignment of L_I generates an uncertain answer only if it fails the third condition, that is, if $h < l$. Thus, we need to calculate only how many assignments of L_I satisfy $h < l$. Assume that $h < l$. Since C_h is the last step combination such that WIE(C_h) has a solution, there is no solution for WIE(C_l), WIE(C_{l+1}), ..., WIE(C_z). Because WIE(C_1) must have a solution and $l > h \geq 1$, l can only be 2, 3, ..., or z. For each l, only WIE(C_2), ..., WIE(C_{l-1}) have two possible results; thus, there are only 2^{l-2} possibilities such that $l > h$. Hence the total number of possibilities that satisfy $l > h$ is $\sum_{l=2}^{z} 2^{l-2} = 2^{z-1} - 1$. Dividing this number by the total number of assignments of L_I, it follows that the proportion of the assignments that generate an uncertain answer is

$$\frac{2^{z-1} - 1}{z 2^{z-1}},$$

which converges to $1/z$ as $z \to \infty$. □

For each step combination C_u of L_I, from Property 5, we know that WIE(C_u) and HIE(C_u) each have a layout; thus, it is the size constraints that determine whether there are solutions for WIE(C_u) and HIE(C_u). If we assume that size constraints generate each assignment of L_I with equal probability, then the probability of giving the uncertain answer is $(2^{z-1}-1)/(z2^{z-1})$. Let

$$P_I = \frac{2^{z-1}-1}{z2^{z-1}}.$$

$P_I = 0$ if $z = 1$. When $z > 1$, P_I becomes smaller as z become larger. $P_I = 0.25$ if $z = 2$, and $P_I = 0.1$ if $z = 10$. When $z = 100$, $P_I = 0.01$. From Theorem 5.9, we obtain the following heuristic: A long list, L_I, of step combinations for instance I tends to reduce the possibility of generating an uncertain answer. Therefore, we should try to find as many step combinations as possible for the list L_I.

Given an instance I of TF, we try to generate a list L_I of step combinations that satisfies Properties 1–5. While we are checking the step combinations in L_I, we have three possible results: yes, no, and uncertain. Based on this approach, we obtain a polynomial-time algorithm that partially solves TF as given in Fig. 5.5.

In Algorithm 2, `Find_First_Combination`, given in Appendix B, generates the first step combination C_1 that satisfies Property 1 and a layout (W^1, H^1), where $W^1 = w_1^1, \ldots, w_n^1$ and $H^1 = h_1^1, \ldots, h_m^1$, in which all items are typeset within the corresponding steps in C_1. The algorithm returns one of three possible answers:

- `Not_Found`, if there is no step combination that satisfies Property 1.
- `Both_Ok`, if C_1 exists and WIE(C_1) and HIE(C_1) each have a solution. In this case, $w_j^1 (1 \le j \le n)$ is a solution of WIE(C_1) and $h_i^1 (1 \le i \le m)$ is a solution of HIE(C_1).
- `Wid_Ok`, if C_1 exists and only WIE(C_1) has a solution. In this case, $w_j^1 (1 \le j \le n)$ is a solution of WIE(C_1) and $h_i^1 (1 \le i \le m)$ is a layout of HIE(C_1).

To find the first step combination, `Find_First_Combination` first attempts to find the column widths $w_j^1 (1 \le j \le n)$ such that

1. $w_j^1 (1 \le j \le n)$ satisfy the width constraints.
2. For each item $o_k = (t_k, l_k, b_k, r_k, \xi_k, \psi_k)$, $\sum_{p=l_k}^{r_k} w_p^1 \ge \psi_k[min]$.

If there are no such column widths, the function returns `Not_Found`; otherwise, it finds the steps for all the items based on $w_j^1 (1 \le j \le n)$ and generates the row heights

Algorithm 2 `TF_Polynomial-Time_Algorithm(column_widths, row_heights):` **enum**

```
    var integer column_widths[1..column_number], row_heights[1..row_number];
begin
    integer pair com_steps[1..item_number];
    enum {Not_Found, Wid_Ok, Hei_Ok, Both_Ok, None_Ok, End} result, pre_result;

    result := Find_First_Combination(com_steps, column_widths, row_heights);
    if result = Not_Found then
        return(No)
    else while not result = Both_Ok and not result = End do
              pre_result := result;
              result := Find_Next_Combination(com_steps, column_widths,
                                              row_heights);
          end while;
          if result = Both_Ok then
              return(Yes)
          else if result = End and not pre_result = Hei_Ok then
                  return(No)
               else return(Uncertain) end if
    end if
end
```

Figure 5.5: A polynomial-time algorithm that partially solves TF.

$h_i^1 (1 \le i \le m)$, which ensures that L_I satisfies Property 1 and C_1 satisfies Property 5. Moreover, `Inequality_Solver` guarantees that $\sum_{j=1}^{n} w_j^1$ is the minimum among the step combinations that satisfy Property 1.

Given a step combination C_u and its layout (W^u, H^u), where $W^u = w_1^u, \ldots, w_n^u$ and $H^u = h_1^u, \ldots, h_m^u$, `Find_Next_Combination`, given in Appendix B, finds a new step combination C_{u+1}, generates a new layout (W^{u+1}, H^{u+1}), where $W^{u+1} = w_1^{u+1}, \ldots, w_n^{u+1}$ and $H^{u+1} = h_1^{u+1}, \ldots, h_m^{u+1}$, in which all items are typeset within the corresponding steps in C_{u+1}, and ensures that L_I satisfies Properties 2–5. It returns one of the five possible answers:

- `End`, if all steps in C_u are the largest steps of their corresponding items.
- `Both_Ok`, if C_{u+1} exists and both WIE(C_{u+1}) and HIE(C_{u+1}) have solutions. In this case, $w_j^{u+1} (1 \le j \le n)$ is a solution of WIE(C_{u+1}) and $h_i^{u+1} (1 \le i \le m)$ is a solution of HIE(C_{u+1}).
- `Wid_Ok`, if C_{u+1} exists and only WIE(C_{u+1}) has a solution. In this case, $w_j^{u+1} (1 \le j \le n)$ is a solution of WIE(C_{u+1}) and $h_i^{u+1} (1 \le i \le m)$ is a layout of HIE(C_{u+1}).
- `Hei_Ok`, if C_{u+1} exists and only HIE(C_{u+1}) has a solution. In this case, $h_i^{u+1} (1 \le i \le m)$ is a solution of HIE(C_{u+1}) and $w_j^{u+1} (1 \le j \le n)$ is a layout of WIE(C_{u+1}).
- `None_Ok`, if C_{u+1} exists and neither WIE(C_{u+1}) nor HIE(C_{u+1}) has a solution. In this case, $w_j^{u+1} (1 \le j \le n)$ is a layout of WIE(C_{u+1}) and $h_i^{u+1} (1 \le i \le m)$ is a layout of HIE(C_{u+1}).

To reduce the number of uncertain responses, we try to find a step combination that can generate a solution or lead us to a solution rapidly by selecting as few items as possible whose steps we increase, to avoid reaching the largest steps of the items as long as possible. Based on these ideas, we use the following heuristics to obtain C_{u+1}:

1. For each column $1 \le k \le n$, we increase its width w_k^u to a new width w_k^* such that w_k^* is the minimal width to cause at least one item to fall into the next step. Based on $w_1^u, \ldots, w_{k-1}^u, w_k^*, w_{k+1}^u, \ldots, w_n^u$, we generate a new step combination C'_k and a layout (W'_k, H'_k), where $W'_k = w'_{k1}, \ldots, w'_{kn}$ and $H'_k = h'_{k1}, \ldots, h'_{km}$. The n step combinations $C'_1, C'_2, \ldots, C'_n$ are possible candidates for C_{u+1}.

2. During Step 1, if we find that all items have reached their largest steps, we return `End`.

3. If there is a C'_k such that both WIE(C'_k) and HIE(C'_k) have solutions, then C_{u+1} is chosen as this C'_k and (W^{u+1}, H^{u+1}) as (W'_k, H'_k).

4. If we do not find a C_{u+1} in Step 3, we let C_{u+1} be a C'_k such that $\sum_{j=1}^{n} w'_{kj} + \sum_{i=1}^{m} h'_{ki}$ is a minimum. In this case, (W^{u+1}, H^{u+1}) is chosen as (W'_k, H'_k).

Step 1 guarantees that each C'_k satisfies Properties 2 and 3. It also guarantees that each C'_k satisfies Property 5 because $w^u_1, \ldots, w^u_{k-1}, w^*_k, w^u_{k+1}, \ldots, w^u_n$ must be a layout for WIE(C'_k) and $h^u_i (1 \le i \le m)$ must be a layout for HIE(C'_k). Step 2 ensures that L_I satisfies Property 4. Steps 3 and 4 increase the likelihood that we find a solution. Step 4 is based on the observation that we usually specify size constraints for table width and height. If we make the table width and height as small as possible, we are more likely to find a solution in the succeeding search.

The running time for `Find_First_Combination` is $O((r+s)^3)$ and the running time for `Find_Next_Combination` is $O(n(n+m+(r+s)^3))$. By Lemma 5.5, the number of the step combinations in the list is at most $\sum_{j=1}^{r} K_j$. Therefore, the total running time for Algorithm 2 is

$$O(\sum_{j=1}^{r} K_j n(n+m+(r+s)^3)),$$

where n is the number of columns, m is the number of rows, r is the number of items, s is the number of size constraints, and K_j is the number of steps for item o_j. The running time increases at a polynomial rate as n, m, r, and s increase.

5.7 An efficient algorithm

By combining Algorithms 1 and 2, we obtain a more efficient algorithm that can completely and correctly solve TF as given in Fig. 5.6. For each instance of TF, Algorithm 3 first uses Algorithm 2 to check a list of step combinations $C_1, C_2, \ldots, C_z$ that satisfy Properties 1–5. If Algorithm 2 does not find a solution for the instance, then Algorithm 1 is used. By Theorems 5.3 and 5.8, Algorithm 3 guarantees to solve TF completely and

```
Algorithm 3 TF_Efficient_Algorithm(column_widths, row_heights): bool

    var integer column_widths[1..column_number], row_heights[1..row_number];
begin
    enum {Yes, No, Uncertain} result;

    result := TF_Polynomial_Algorithm(column_widths, row_heights);
    if result = Uncertain then
        return(TF_Exponential_Algorithm(column_widths, row_heights));
    else if result = Yes then
            return(true);
        else return(false); end if

end
```

Figure 5.6: An efficient algorithm that always solves TF.

correctly. Although Algorithm 3 is still an exponential-time algorithm in the worst case, it is many more efficient than Algorithm 1 for many instances.

We can divide the instances of TF into two groups, G_e and G_p. G_e includes the instances for which Algorithm 2 returns Uncertain and G_p includes the instances for which Algorithm 2 returns either Yes or No. Thus, Algorithm 3 takes polynomial time to solve the instances in G_p and takes exponential time to solve the instances in G_e. By Theorem 5.9, the probability of giving an uncertain answer by Algorithm 2 for each instance of TF is no more than 0.25 if we assume that the size constraints generate each assignment of the corresponding list of step combinations with equal probability. In addition, given a rectangular region, text is usually typeset to fill a region that is as wide as possible. If the region is not wide enough, text is broken into lines to vertically fill the region. Thus, we usually specify only width constraints to control the layout of a table. In these cases, HIE(C_1) must have solutions; thus, we can decide whether there are solutions for the instances by Algorithm 2. The height constraints may be necessary when a table is too long to fit into a region and it is possible to shorten it by widening the table. Therefore, we believe that G_p contains many more common instances than

Table 5.6: A schedule of computer science courses.

Time	Monday	Tuesday	Wednesday	Thursday	Friday
	Introduction to computer science	Data structure	System softwares	Algorithm analysis	Software engineering
Morning 9:00-12:00	This section is for those who don't know anything about computer science and just want to know something about it.	This section is for those who already know something about computer science and intend to have a career in the software industry in the future.			
Afternoon 1:00-4:00		This section is for those who already know something about computer science and intend to learn how to write simple programs.		This section is for those who know quite a lot about computer science and intend to learn more so that they can have a career in the software industry in the future.	
Evening 7:00-10:00	This section is for those who don't know anything about computers and intend to learn how to write simple programs.				

G_e. For the languages in which people are used to reading text from top to bottom (such as Chinese and Japanese), a similar algorithm holds when we interchange the roles of widths and heights in the algorithm.

We end this chapter with an example that was generated by our tabular composition system. Table 5.6 consists of a 5×6 grid and 19 items. We have the following size constraints:

$$400pt \leq w_1 + w_2 + w_3 + w_4 + w_5 + w_6 \leq 450pt$$
$$w_2 \geq 100pt$$
$$h_4 \leq 100pt$$
$$h_1 + h_2 + h_3 + h_4 + h_5 \leq 400pt$$

For this instance, Algorithm 3 is able to find a solution in polynomial time. If we add the additional size constraint

$$h_3 \geq 200pt$$

to the instance, Algorithm 3 takes exponential time to discover that there is no solution for the new instance.

Chapter 6

Implementation

A tabular composition system should help users to design and produce high-quality tables. A user friendly system should allow users to concentrate primarily on the manipulation of the logical structure of a table and to specify the layout structure using a style-based approach. To achieve this goal, a tabular system should be able to abstract and manipulate a table's logical structure and provide the ability to specify the layout requirements, including topology and style. In Chapter 2, we presented an abstract model for the specification of a table's logical structure. This model can be used as the basis of a tabular composition system. The editing model described in Chapter 3 provides operations for the logical manipulation of tables. The topological rules and the style rules described in Chapter 4 provide one method of specifying a table's layout structure through a set of presentational rules. Based on these ideas, we have implemented a prototype tabular composition system XTABLE. XTABLE runs in a UNIX and X Windows environment. In the remainder of this chapter, we first describe, in Section 6.1, the objectives of XTABLE. Then, in Section 6.2, we describe the steps that are involved in the generation of a concrete table from an abstract table by applying a set of user-defined topological and style rules. In Section 6.3, we present a hierarchical object-oriented view of various tabular objects and their operations. Finally, we introduce the overall system structure in Section 6.4 and the user interface in Section 6.5.

6.1 Objectives

XTABLE was designed to provide an interactive environment for the composition of high-quality tables in two dimensions. It should meet following objectives:

- To describe and manipulate tables based on their logical structure

  The logical relationships among the components of a table should be abstracted to form an abstract table that is independent of the layout structure of the table. In addition, XTABLE should provide operations to edit tables based on their logical structure.

- To topologically arrange the tabular components in two dimensions

  XTABLE should topologically arrange objects in both horizontal and vertical dimensions, should allow a user to order labels, and should automatically place the entries in appropriate positions so as to convey clearly the logical relationships among tabular components.

- To specify style rules for different kinds of tabular components

  XTABLE should allow a user to specify both collective style rules for a collection of tables and specific style rules for particular tables. These style rules should be applied to presentational objects, logical objects, and layout objects.

- To format tables based on user-defined layout specifications

  XTABLE should automatically determine the physical dimensions of a final layout according to user-defined topological and style specifications. It should provide both fixed and automatic line-breaking methods and should satisfy column and row constraints simultaneously. The formatting should satisfy row and column constraints simultaneously.

- To provide a WYSIAWYG environment to edit the logical structure, topology and styles of tables

  The presentational-oriented, logical and layout objects should be organized hierarchically. Users should be able to select these objects by using a mouse and to indicate operations by menu, tool-box and dialog-box techniques. The new presentation of a table should be redisplayed on the screen right after each operation.

- To create a stand-alone tabular system that can support various formatting systems

  XTABLE should be independent of any existing document formatting system. It should generate formatted tabular output for several typesetting systems; for example, for LaTeX, `troff`, and Postscript.

6.2 Abstract to concrete

We specify, in XTABLE, the logical structure of a table using the abstract model given in Chapter 2 and the layout structure using the topological and style rules described in Chapter 4. Given an abstract table, a topological specification, and a style specification, we generate a concrete table using a two-step process. First, the *arrangement step* generates a grid structure and a set of size constraints for the columns and rows in the grid structure. Then, the *formatting step* determines the physical dimensions of the columns and rows for the grid structure according to the size constraints.

6.2.1 Grid structure

The implemented grid structure is more complex than the definition we used in Chapter 5, where we extracted only the properties necessary for the formal description of tabular formatting. In XTABLE, a *grid structure* consists of three components: a grid, a set of nonoverlapping items, and a set of separations. Recall that an $m \times n$ grid is a planar integer lattice with m rows and n columns. An *item* is an object that is placed in a block of a grid. We use a four-element tuple (position, content, format, size function) to define an item. The *position* of an item is the block in which the item is placed. The *content* of an item can be any kind of data, such as a string of characters, a fixed-size picture and image, a table, and so on. At present, we allow only strings of characters. The *format* of an item includes the typographic attributes that determine the appearance of the item, such as font families and sizes, background patterns, line spacing, and so on. The *size function* is a decreasing step function that describes the line-breaking characteristics of the items. Before we define separation, we need to introduce more terminology. The lines that horizontally separate the rows are called *row grid lines* and the lines that vertically separate the columns are called *column grid lines*. A *grid point* is the intersection of a row grid line and a column grid line. A *separation* is either a rule surrounded by white space or white space that we use to separate cells, blocks, rows, and columns in a table. We can use a three-element tuple (position, rule style, spacing) to define a separation. The *position* of a separation specifies two grid points between which the separation lies. These two grid points must be on the same horizontal line or the same vertical line. The *rule style* consists of the rule type and the rule width. The *spacing* specifies the extents of the left and right (or upper and lower) spacing on each side of the rule.

6.2.2 Size constraints

Although the formatting algorithm in Chapter 5 supports any size constraints expressed as linear equalities or inequalities, we further restrict the size constraints in XTABLE to simplify the user interface and decrease the execution time of the tabular formatting algorithm. XTABLE allows only four kinds of linear inequalities for the size constraints:

- $l \leq \sum_{j=p}^{q} w_j$ (the width of a set of consecutive columns is no less than l)
- $\sum_{j=p}^{q} w_j \leq u$ (the width of a set of consecutive columns is no more than u)
- $l \leq \sum_{i=p}^{q} h_i$ (the height of a set of consecutive columns is no less than l)
- $\sum_{i=p}^{q} h_i \leq u$ (the height of a set of consecutive columns is no more than u)

We have used w_j to denote the width of the jth column and h_i to denote the height of the ith row, and l and u are positive integer constants. We believe that these four kinds of size constraints are sufficient to specify most size requirements for tables. XTABLE, however, does not allow the specification of equality constraints for columns or rows, which imposes the equality of column widths or row heights in a table.

6.2.3 Arrangement

Given an abstract table, a topological specification, and a style specification, it is easy to generate a grid and the blocks occupied by the items in the grid. It is more difficult to determine the formatting attributes for the items and the separations. We have to decide on a reasonable strategy to determine what formatting attributes an item or a separation should use when multiple inheritance occurs.

As we mentioned in Section 4.4.1, there are three approaches for handling multiple inheritance of style rules. The strategy we use is a combination of priority order and combining style approaches. We try to combine the style rules of all super-objects. Whenever there is no satisfactory combination, we use the style rules of the super-object with the highest priority as specified by the user. There are two possible ways to determine the inheritance priority: fixed and free. With *fixed priority* , the inheritance ordering of style rules is predetermined by the designers of the system. For example, we, as system designers, can specify that the style rules for rows have a higher priority than the style rules for columns. In this framework, users do not have to specify the inheritance ordering of style rules. On the other hand, users are unable to change the fixed priority. With *free priority*, the inheritance ordering of style rules is dynamically specified by users based on the requirements of their tables. Although it gives users flexibility to handle style inheritance, this approach requires users to specify inheritance orderings for each table. The combination of these two approaches provides a better solution. In XTABLE, the priority for some scopes, including the whole table, the stub, the boxhead the stub head, the body, and the categories, is predetermined. We use the genealogical tree, shown in Fig. 6.1, to describe the relationships between these scopes. Therefore, we can predetermine the priority for these scopes using single inheritance. The style rules for a cell can be inherited according to the priority: the category that contains a label that occupies the cell, the region that contains this cell, and the whole table. For example, in Table 6.1, the cells that contain the label **Winter** inherit the style rules of category **Term** first, then of the stub, and finally of the whole table. Since the style rules for these scopes determine the general appearance of a table, they are appropriate for most tables. With fixed-priority inheritance, users do not need to indicate the inheritance ordering if they specify only the style rules for these scopes. Since the remaining scopes, including the rows, the columns, the blocks, the labels, the subcategories, the entries, the entry set,

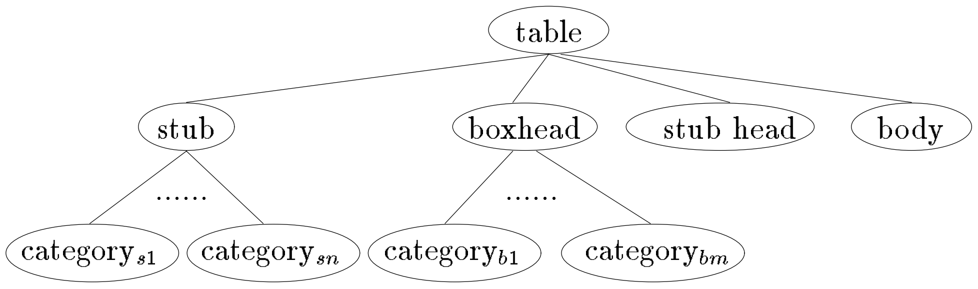


Figure 6.1: The genealogical relationship of some scopes.

and the entry values, are specified infrequently and may cause multiple inheritance, the priority for these scopes is determined by users according to their requirements. For the style rules with these scopes, the last specified style rule has the highest priority. Moreover, these style rules have higher priority than the style rules in the preceding single-inheritance ordering.

Multiple formatting attributes for a cell or for a separation are combined by inheritance of the style rules of various objects based on the priority we have described. A cell may inherit the font family from the style rule of the stub and the font size from the style rule of the whole table. If an item occupies a block that contains more than one cell, we need to determine the formatting attributes of the cell used to display the item. In XTABLE, we use the formatting attributes of the top-left cell of a block to display the item that occupies the block.

We give an example to explain our inheritance strategy. Table 6.1 is generated by specifying four style rules. We assume that the style rules are specified in this order:

TABLE:	Roman
BOXHEAD:	bold face
COLUMN 2:	Courier
ROW 4:	Helvetica

The labels in the boxhead are displayed in bold Roman by inheriting the Roman attribute from the whole table and the bold face attribute from the boxhead. The label **Spring** in cell (4, 2) is displayed in Helvetica because the style rule for the 4th row is specified

Table 6.1: 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

last. Label **1991** is presented in Roman instead of Helvetica because the top-left cell (3, 1) of its block inherits the font family from the whole table.

6.2.4 Formatting

The formatting step must calculate the physical dimensions of the columns and the rows in a grid structure so that all size constraints are satisfied and all items can be placed completely inside the block they occupy. If an item is a long string, there are two ways to break the string into lines. *Fixed line breaking* requires users to indicate the line breaks in the string and *automatic line breaking* requires the system to determine the line-break points based on the current dimension of the column. XTABLE allows both fixed line breaking and automatic line breaking. We adopt the main ideas of the formatting algorithm presented in Chapter 5 to determine the physical dimensions of a table. Since the allowable size constraints in XTABLE are simpler, we are able to reduce the running time of the algorithm by making two changes. First, we do not use the simplex method to solve the linear equalities and inequalities since the size constraints in XTABLE can be expressed using a small number of inequalities. We use a more efficient inequality solver. Second, we use a branch-and-bound strategy to generate only those step combinations that guarantee to give a layout for a table. Any step combination for which does not give to a layout is not considered. For example, suppose two items o_1 and o_2 are placed in the same column and o_1 has a step [20, 30) and o_2 has a step [50, 60); then there is no layout for a step combination that contains these two steps because they do not overlap. By omitting such step combinations, the number of step combinations that are checked in the exponential-time search is greatly reduced.

The size functions of items are dependent on the medium that is used to display the table. Since different media use different font sizes, the line breaks of an item may be different with different media and the same table may be presented differently with different media. Thus, the formatting process needs to take a size function as a parameter and generate a concrete table that is dependent on the given size function.

6.3 Tabular objects and their operations

We adopt an object-oriented technology in XTABLE to provide an interactive environment for the manipulation of the logical structure, the topology, and the styles of tables. Tabular components are classified into object classes and editing operations are associated with them. Table 6.2 shows the object classes and their operations in XTABLE.

There are three kinds of object classes: presentational objects, logical objects, and layout objects. The *presentational objects* include the entire table and the four major regions: the stub, the boxhead, the stub head, and the body. The *logical objects* are the logical components of an abstract table including category, subcategory, label, entry, entry set, and entry value. The *layout objects* are the layout components of a concrete table including block, row, and column.

There are also three kinds of operations for the object classes: logical, topological, and style. A *logical operation* changes the logical structure of a table for example, by adding a category to a table, deleting a label from a category, or editing an entry. Logical operations can be decomposed into sequences of the editing operations introduced in Chapter 3. A *topological operation* changes only the topological specification of a table, for example, transposing a table, moving a category from the stub to the boxhead, or changing the ordering for a category. A *style operation* changes only the style specification of a table, for example, changing the cell style, the separation style, or the arrangement

Table 6.2: The object classes and their operations.

Objects		Operations		
		Logical operations	Topological operations	Style operations
Present-ational-oriented objects	Table	Clear	Transpose	Frame style, grouping style, size constraints, category h. style, basic style*
	Stub			Arrangement style, category h. style, basic style*
	Boxhead			
	Stub head			Arrangement style, basic style*
	Body			Spanning style, basic style*
Logical objects	Category	Add, remove, copy, combine, split, text edit	Move, change order	Category h. style, basic style*
	Subcategory	Add, remove, copy, logical move, combine, split, text edit	Topological move, change order	Frame style, basic style*
	Label	Remove, copy, move, text edit		Frame style, cell style
	Entry	Copy, move, remove, compute, text edit		
	Entry value			
	Entry set			Frame style, basic style*
Layout objects	Block	Copy, move, remove, text edit		Spanning style, frame style, size constraints, basic style*
	Row			
	Column			

*Basic style includes cell style and separation style.

style for different objects.

After introducing some internal object classes, we obtain the object hierarchy shown in Fig. 6.2. The operations of the object classes in this hierarchy are synthesized. The objects at lower levels may inherit the operations of ancestral objects. The hierarchy enables us to use object-oriented technology to implement an interactive editing environment.

6.4 Overall system structure

We separate the collective style specification from the specific style specification in XTABLE. The collective style rules are in a separate file and the specific style rules are associated with a specific table. Thus, the collective style rules can be applied to multiple tables.

6.4.1 Input and output

As shown in Fig. 6.3, XTABLE accepts three kind of input: table files, collective style files, and user instructions. A table file has three parts: an abstract table, a topological specification, and a specific style specification. A collective style file contains only collective style rules. Appendix D gives examples of a table file and a collective style file. Through an interactive editing environment, users provide XTABLE with instructions for the manipulation of the logical structure, the topological specification, and the style specification (both specific and collective). At any time, the current status of the abstract table, the topological specification, and the specific style specification can be saved as a table file, and the current status of collective style specification can be saved as a collective style file. The updated presentation of an edited table is displayed on the fly. Users can create an abstract table in a particular topology without specifying any style. In this case, the table file contains only the abstract table and the topology, and the table is displayed on the screen using the default style specification. XTABLE is designed to be a preprocessor for some formatting systems, including LaTeX, Postscript, and `troff`. Currently XTABLE generates only LaTeX output.

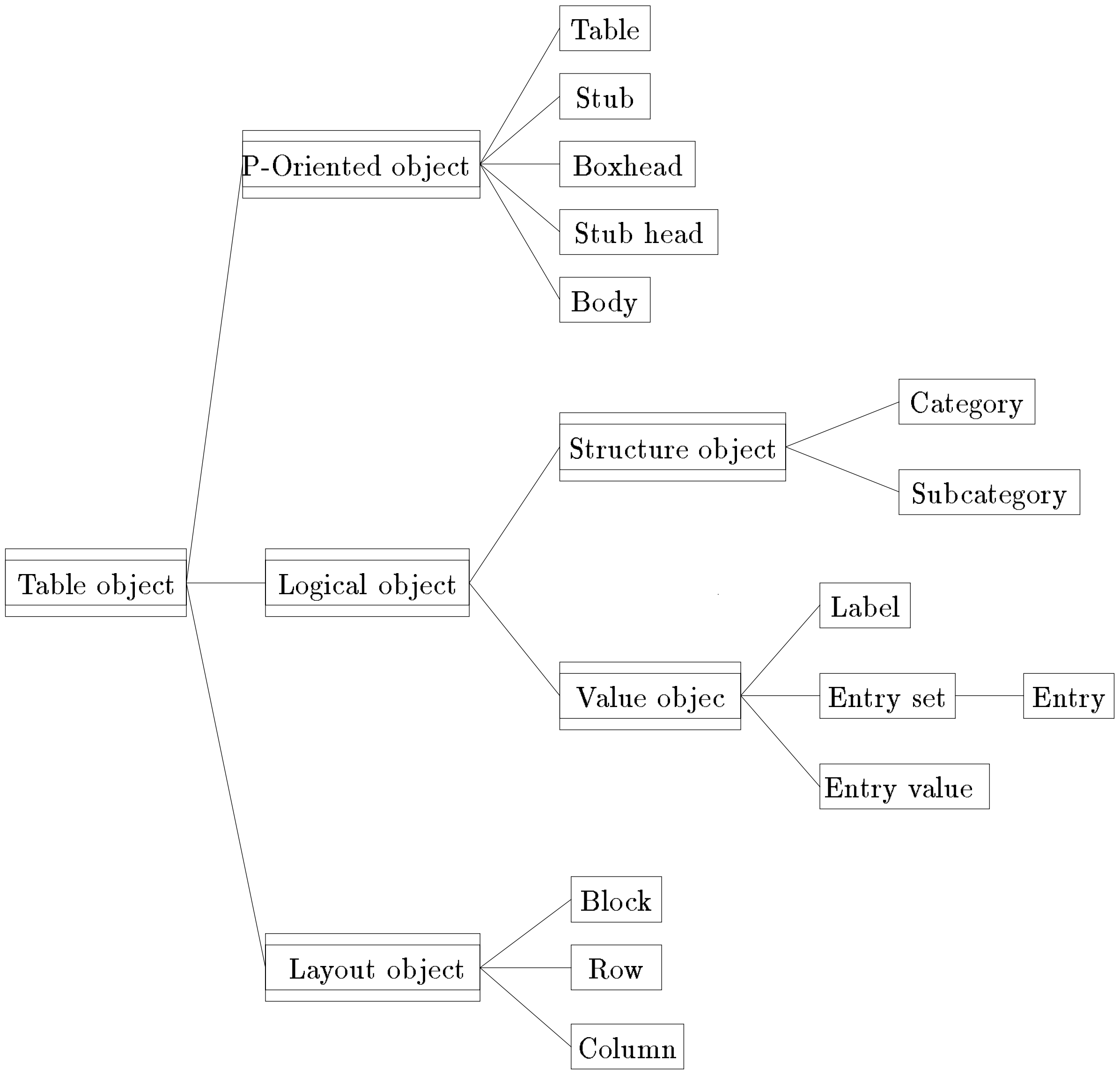


Figure 6.2: The object class hierarchy.

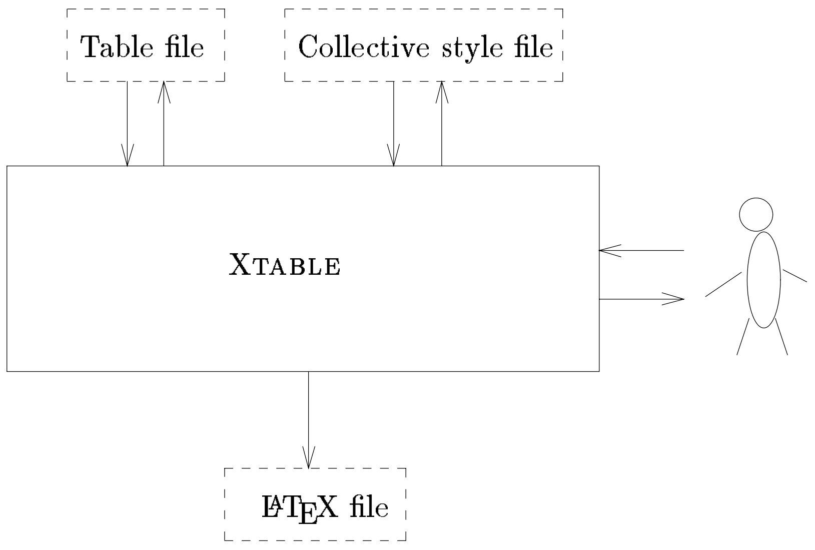


Figure 6.3: The input/output of XTABLE.

6.4.2 Internal data structures and processes

XTABLE, as shown in Fig. 6.4, maintains four major data structures for the abstract table, the topological specification, the specific style specification, and the collective style specification. Their initial values are given by a table file and a collective style file or assume defaults if neither table file nor collective style file is provided. During the interactive editing process, these data structures are updated according to user commands. We use Motif as the interface between a user and the system. We generate three intermediate data structures whenever XTABLE displays a table or compiles a table specification into a LaTeX file. The arrangement process generates a grid structure and a set of size constraints, and then the formatting process generates a concrete table. A concrete table can either be displayed through the Motif interface or be transformed into a source file for LaTeX. Since different systems use different font sizes, the formatting process needs to know the size function for a particular system before calculating the absolute positions of all the items and rules. Thus, we need to implement the size functions for Motif, LaTeX, Postscript, and `troff`. Due to limitations of the LaTeX table environment, we transform a concrete table to the LaTeX picture environment in which all tabular items and rules

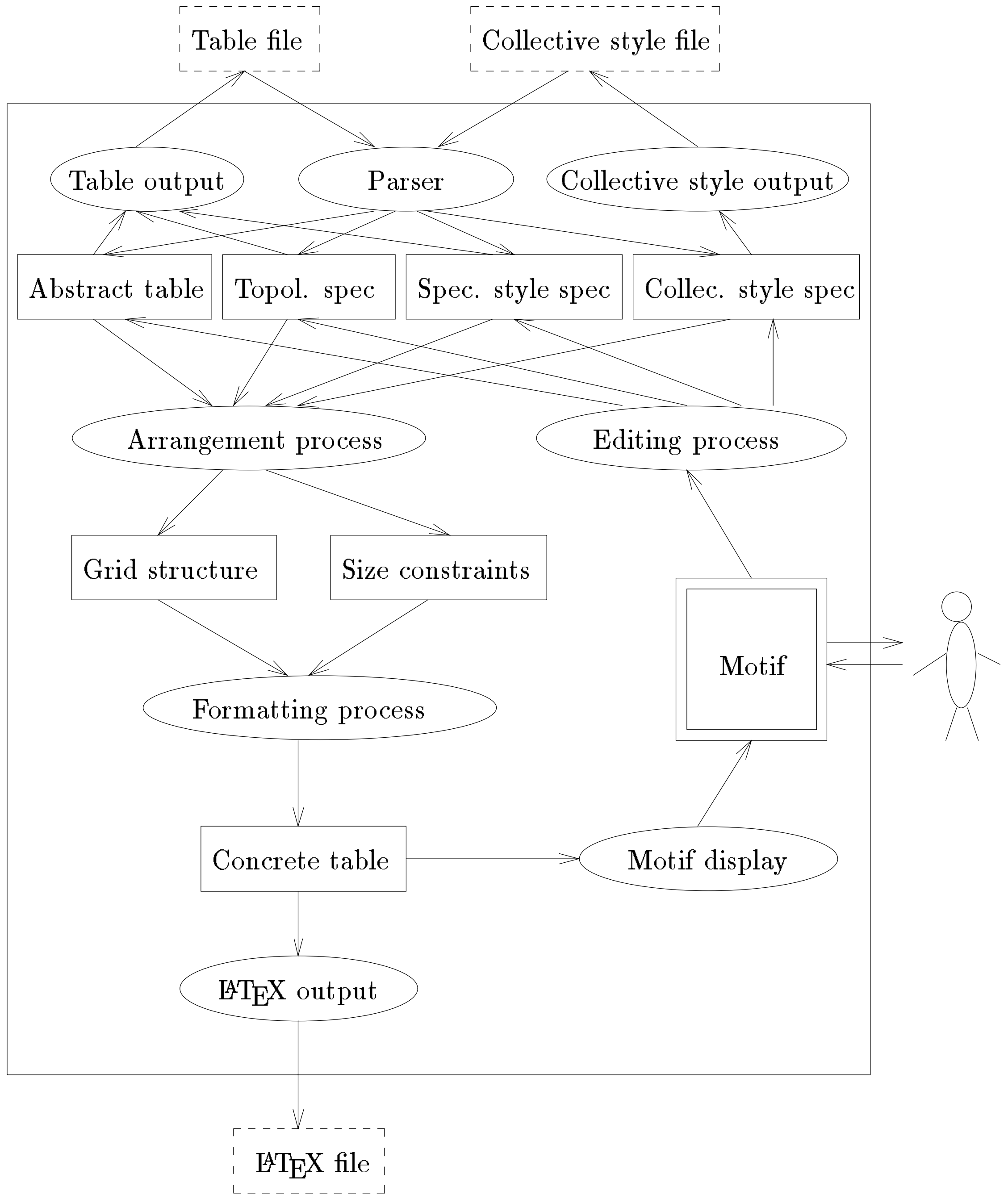


Figure 6.4: The internal system structure of Xtable.

are treated as graphical objects. To implement the size function for LaTeX, we need to use METAFONT `tfm` files that are used in TeX.

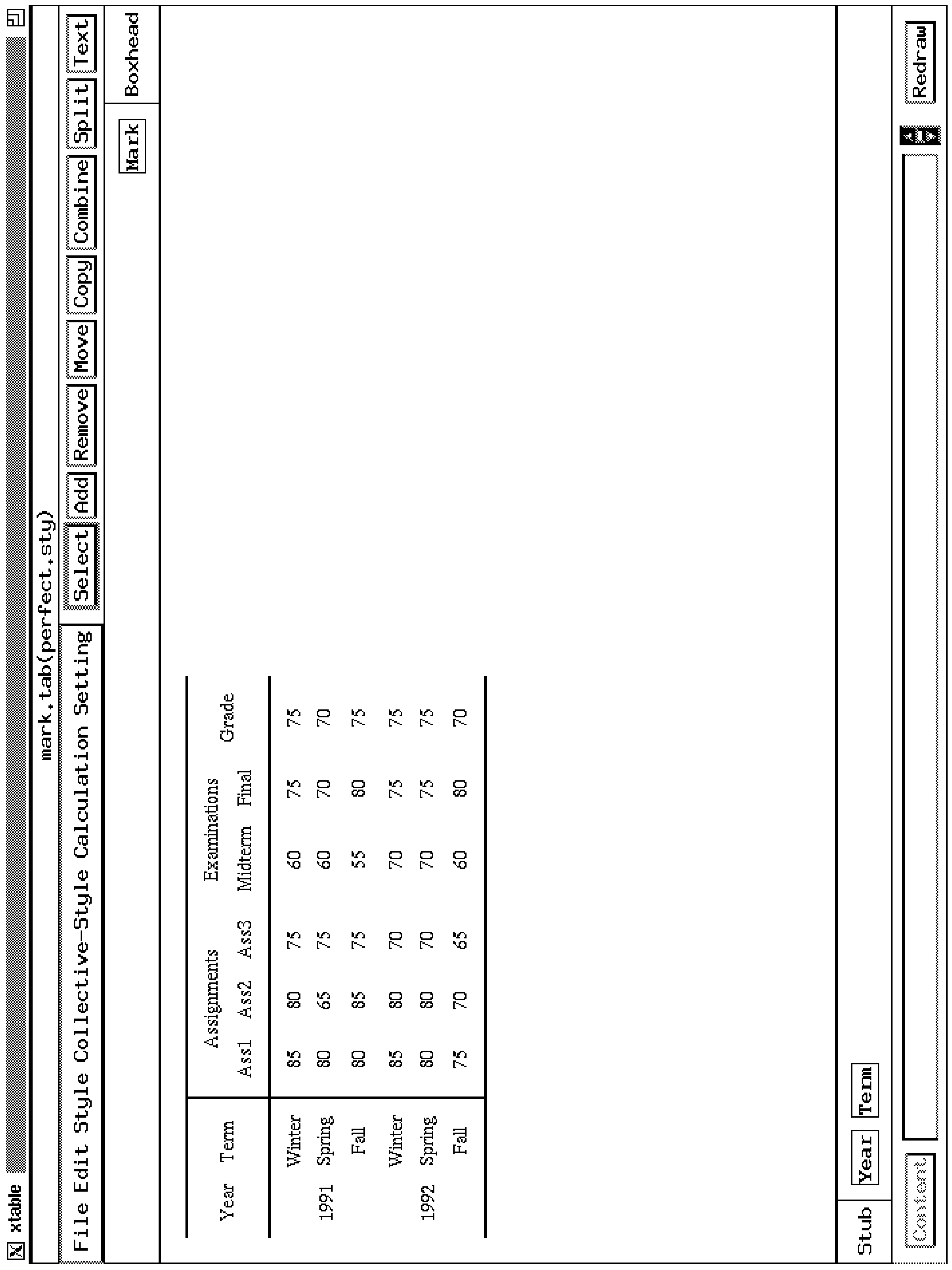
6.5 User interface

XTABLE's user interface enables users to select editing objects by using mouse and to indicate the operations by the menu, tool-box, and dialog-box techniques. Fig. 6.5 shows the main window of XTABLE in which a table is displayed. There are three editing areas in the main window: stub, boxhead, and table. The categories that are assigned to the stub (the boxhead) appear in the stub area (the boxhead area) and the concrete table is presented in the table area. A menu bar and a set of tool boxes are created for users to use for editing. Once users have selected an object and indicated an operation and its arguments, a new presentation of the table is generated in the table area after applying the operation to the object. Two approaches are used to specify editing operations: tool boxes and menus.

6.5.1 Tool boxes

The most frequently used operations (**add**, **remove**, **copy**, **move**, **combine**, **split,** and **text**) are provided as tool boxes. Once the user clicks on a tool box, the corresponding operation is active until the user clicks on another tool box. When a tool box is active, the user needs to indicate to which object the operation is applied and to specify the required arguments by pointing and dragging in the three editing areas. We use different mouse buttons to distinguish the insertion modes: the left, middle, and right keys are used to insert an object before, under, and after the active object, respectively. The tool box **select** is used to indicate the editing objects for menu operations. The content of the current active object is displayed in the subwindow at the bottom of the main window. Users can edit the content of the object in that subwindow and press the button **content** on its left after the editing is down. The button **redraw** at the bottom is used to redisplay the edited table on the screen.

To show how users edit tables using XTABLE, we have included some screen shots in Appendix C. Suppose we have constructed the table given in Fig. C.1. To move the


xtable
mark.tab(perfect.sty)
File Edit Style Collective-Style Calculation Setting
Select Add Remove Move Copy Combine Split Text
Mark
Boxhead
Stub
Year
Term
Content
Redraw

Year	Term	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

Figure 6.5: The main window of XTABLE.

category **Year** from the stub to the boxhead immediately before the category **Mark,** we first click on the tool box **move,** click on **Year** in the stub area, and, finally, click on **Mark** in the boxhead area by pressing the left key of the mouse. Now we obtain the table given in Fig. C.2. To add a new label **Ass4** under the subcategory **Assignments** and place it after **Ass3** in the ordering, we first click on tool box **add,** then click on the label **Assignments** in the table area by pressing the middle key of the mouse, and finally enter **Ass4** in the content widget and press **content**. Figs. C.3 and C.4 show the changes to the table after adding a new label and assigning the new name **Ass4**. Now we can enter the marks that are associated with **Ass4** to obtain Fig. C.5. We first click on tool box **text**, then drag the cursor to select all the cells for the new marks, and finally enter the marks in the table.

6.5.2 Menus

Most topological operations, style operations, and system commands are listed in the menu bar. The menu **File** consists of input and output commands, such as reading a table file or a collective style file, and generating a LaTeXsource files; the menu **Edit** consists of the other logical and topological operations that are not available as tool boxes; the menu **Style** consists of the style operations for specific style specification that can be applied only to the current edited table; menu **Collective-Style** consists of the style operations for collective style specification that can be applied to a collection of tables; the menu **Calculation** consists of the operations average, total, minimum, and maximum that are used to compute entry values; and the menu **Setting** consists of the commands for the selection of the system parameters. Users have to select an editing object with the tool box **select** before pulling down the menu and clicking on an option. If an operation is associated with only one object, such as **transpose** or **clear** for the whole table, then the user can directly click on the operation without first indicating the editing object, independently of which tool box is active. When a style operation is selected, a dialog box pops up to assist users to edit the formatting attributes of the style rule for the selected object.

6.6 Merits and limitations

XTABLE is a tool that helps users to design high-quality tables in two dimensions. It provides an interactive environment for editing the logical structure, topology, and style of a table and for presenting a table easily with multiple layout structures. XTABLE is also a tool that helps users to explore the data from different viewpoints. By arranging table items flexibly in two dimensions, users are able to discover relationships among of or patterns in the data. This ability helps users to analyze and understand tabular data in an efficient way. Tables 6.3 shows the correlations for 10 TV programs based on whether people in a sample of 7,000 UK adults said they "really liked to watch" the range of programs such as World of Sport (WoS), Match of the day (MoD), and Panorama (Pan). In Table 6.3, TV programs are subcategories of two TV broadcasting stations: ITV and BBC. This presentation does not show any clear pattern in the data. After combining the TV programs with the corresponding TV broadcasting stations and reordering them, we obtain Table 6.4 that shows a cluster for the five Sports programs and another cluster for the five Current Affairs programs. Now we can clearly see the main pattern of the data: correlations of 0.3 to 0.6 between the five Sports programs and of 0.2 to 0.5 between the five Current Affairs programs, with correlations of approximately 0.1 between these two clusters. What we have done in this example is similar to knowledge discovery and data mining that extracts understandable rules and patterns from a large database. Resently there has been an increased interest in exploring various data mining techniques for database applications [FPSSU95]. XTABLE can be used as a visual data-mining tool for database applications if we can establish a connection between XTABLE and a database system.

Since XTABLE is a prototype for validating our tabular model, it does not provide some functionality that a production system will provide. For example, we did not provide well-designed languages for the specifications of table files and the collective style files, because we originally did not expect users to edit them directly. However, there are at least two advantages to allow users to edit these files directly. First, it provides a batch-oriented approach for users to compose tables. In this way, XTABLE can be used as a formatting system that compiles table specifications and generates formatted tables for various systems. Second, other systems can direct their output to XTABLE so that

Table 6.3: The initial table of correlations for 10 TV programs.

Programs		PrB	Thw	ToD	WoS	GrS	LnU	MoD	Pan	Rgs	24H
ITV	PrB		0.1	0.1	0.5	0.5	0.1	0.5	0.2	0.3	0.1
	Thw	0.1		0.3	0.1	0.1	0.2	0.1	0.4	0.1	0.4
	ToD	0.1	0.3		0.1	0.1	0.2	0.0	0.2	0.1	0.2
	WoS	0.5	0.1	0.1		0.6	0.1	0.6	0.2	0.3	0.1
BBC	GrS	0.5	0.1	0.1	0.6		0.1	0.6	0.2	0.3	0.1
	LnU	0.1	0.2	0.2	0.1	0.1		0.0	0.2	0.1	0.3
	MoD	0.5	0.1	0.0	0.6	0.6	0.0		0.1	0.3	0.1
	Pan	0.2	0.4	0.2	0.2	0.2	0.2	0.1		0.1	0.5
	Rgs	0.3	0.1	0.1	0.3	0.3	0.1	0.3	0.1		0.1
	24H	0.1	0.4	0.2	0.1	0.1	0.3	0.1	0.5	0.1	

Table 6.4: The modified table of correlations for 10 TV programs.

Programs	WoS	MoD	GrS	PrB	Rgs	24H	Pan	Thw	ToD	LnU
ITV WoS		0.6	0.6	0.5	0.3	0.1	0.2	0.1	0.1	0.1
BBC MoD	0.6		0.6	0.5	0.3	0.1	0.1	0.1	0.0	0.0
BBC GrS	0.6	0.6		0.5	0.3	0.1	0.2	0.1	0.1	0.1
ITV PrB	0.5	0.5	0.5		0.3	0.1	0.2	0.1	0.1	0.1
BBC Rgs	0.3	0.3	0.3	0.3		0.1	0.1	0.1	0.1	0.1
BBC 24H	0.1	0.1	0.1	0.1	0.1		0.5	0.4	0.2	0.3
BBC Pan	0.2	0.1	0.2	0.2	0.1	0.5		0.4	0.2	0.2
ITV Thw	0.1	0.1	0.1	0.1	0.1	0.4	0.4		0.3	0.2
ITV ToD	0.1	0.0	0.1	0.1	0.1	0.2	0.2	0.3		0.2
BBC LnU	0.1	0.0	0.1	0.1	0.1	0.3	0.2	0.2	0.2	

they can use XTABLE's abilities to edit, analyze, and format tabular data. For example, as we have mentioned, XTABLE can be a data-mining tool if a database table can be transformed to an XTABLE table.

Since XTABLE is based on the abstract model and the presentational model, it inherits the merits and the limitations of these models (see Sections 2.4 and 4.5). All the tables except Tables 2.3 and 5.4 in this thesis were generated by XTABLE in LaTeX format. We have edited the LaTeX files for Table 5.1 to add the footnotes. For Tables 2.2, 4.15, and 4.16, we treated the labels in the body as entries and introduced some empty labels with which the fake entries can be associated.

Chapter 7

Concluding remarks

We presented a tabular model that can support the different stages of tabular composition, including the description and manipulation of logical structure, the specification of topology and style, and the formatting of concrete tables. Based on this model, we have implemented a prototype tabular composition system XTABLE that helps users to design high-quality table layouts. XTABLE enables users to concentrate primarily on the manipulation of a table's logical structure and the specification of the layout with presentational rules. The resulting concrete tables are automatically generated by applying user-defined topology and style specifications to the logical structure. By separating the logical structure of tables from their layout structure, we are able to edit tables based on the logical relationships among tabular items, regardless of where the items appear in the layout structure. We can also easily present a table with different topologies and styles so as to compare different presentations and select the most appropriate one.

We have investigated only the basic requirements for tabular editing, presentation, and formatting. As a result of our exploration, we believe that there are many issues that should be investigated further. In the following sections, we discuss some issues regarding abstract models, presentation, formatting, and browsing.

7.1 Relational database tables

The basic difference between relational database tables and abstract tables is the logical dimension. A database table is two-dimensional with attributes in one dimension and tuples in the other. To represent an abstract table in a relational database, we need to determine which category corresponds to the attribute names, which category corresponds to the primary keys, and which category corresponds to the non-primary keys. Other database models exist, however, for the direct representation of multidimensional tables. Darrell Raymond [Ray96] proposes the use of partial orders as a unifying data model for databases. His model makes it possible to represent multidimensional tables directly in a partial-order database and present them in different topological layouts by applying partial-order operators. Using his model, each dimension of an abstract table can be specified with a partially ordered set and the topology of a table can be specified with a nested partial-order product of these dimensions. For example, using the three-dimensional abstract table defined on page 34, if we place the categories **Year** and **Term** in the stub and the category **Mark** in the boxhead, the topology can be specified as $(Year \times Term) \times Mark$, where the parentheses indicate the grouping.

7.2 Extending the abstract model

As we have mentioned in Section 2.4, we can extend the model by allowing multiple mappings to specify the tables that are a combinations of several tables in a multidimensional structure.

Our abstract model captures only the logical relationships among labels and entries; there are often other relationships among entries. For example, an entry may be the sum of some other entries, as in a spreadsheet. If we extend the abstract model to capture this kind of relationship, we can update an entry once the values of its associated entries are changed. To achieve this objective, we may allow entry values to be formulas whose variables are other entries. We can use, for example, $Average(\{Year.1991, Mark.Final\})$ to represent the value of an entry that is the average of the final marks for the three terms of **1991.**

The abstract model does not distinguish entry and label types; for example, string, number, date, time, and so on. Such distinctions could be used for specifying styles and for computing derived values.

7.3 Different abstract model

Our abstract model requires the distinction between labels and entries. An item has to be a label or an entry, but not both. This limits the representation of the logical associations among such items. For example, a table that converts temperatures between Celsius and Fahrenheit contains two groups of items: the temperatures in Celsius and the temperatures in Fahrenheit. Items in both groups can be either labels or entries. To specify this table with our abstract model, we need to determine which group acts like labels and which like entries. A possible approach to this problem is to make no distinction between labels and entries. We can specify entries as a category and use a relation rather than a function to specify the logical associations among items in different categories. This, however, increases the complexity of arranging items in two dimensions if a table contains more than two categories. We need add one more topological rule to specify which category is put in the body. If one places a category that contains entries in the stub or in the boxhead, the labels may need to be put in the table body. This kind of presentations is, however, against the convention for high-quality tables [Wri68].

7.4 Logical structure recognition

We map an abstract table into a concrete table, but what about the reverse? Can we derive the logical structure of a table when given a concrete table? Image processing techniques enable us to determine the tabular items in a two-dimensional grid. To reconstruct a logical structure we need to distinguish labels from entries. We can use some presentational heuristics to distinguish them. For example, the font and the size of labels may be different from the font and size of entries, or the stub and boxhead separation may be different from other separations in rule type, in rule width, or in spacing. If a user can provide the positions of stub and boxhead separation, recognition is much easier. Douglas `et al.` [DHQ94] present an approach that extracts the logical structure from a table in plain text in two steps: first recognizing its canonical layout, which is similar to a relational database table, and then applying a series of transformations to the canonical layout. Reconstructing a multidimensional logical structure from a two-dimensional database table is comparatively easy, because the attribute names and the items in a column that is a part of the primary key are used mostly as indices; thus, they can be classified as labels. Automatic recognition of tabular logical structure can improve the efficiency of table construction from published documents, hand-written drafts, and database output.

7.5 Different presentational methods

Our tabular model focuses only on presenting tables as a row–column structure in two dimensions. We have not addressed the issue of presenting tables in other forms, such as with bar graphics, line graphics, pie charts, and so on. We would need to introduce different presentational rules to specify the topology and style for these graphical elements. Also we need to investigate possible graphical techniques that utilize the full capabilities of the human visual system. In addition, how might we present an abstract table in three dimensions? We could use different pages as sheets to present the third dimension or we could use a two-dimensional display to present a three-dimensional layout.

7.6 Complexity of tabular formatting

We have given the complexities of tabular formatting problems with different combinations of restrictions in Table 5.1 on page 102, in which 3 problems were solved, 10 problems have conjectures, and 11 problems are unsolved. We obtained the conjectured results by inference and did not give proofs in this thesis. We now give a brief discussion of these conjectured results. For convenience, we define a tabular formatting problem with restrictions as TF(L, S, O), where L is either "fixed line breaking" or automatic line breaking"; S is "none", "linear" for linear equality or inequality, or "non-linear" for non-linear expression; and O is "none", "diameter", "area" or "w_space" for white space.

For example, the formatting problem TF(automatic, linear, diameter) uses automatic line breaking to find a table with the minimal diameter that satisfies the size constraints expressed as linear equalities or inequalities.

Beach proved that TF(fixed, linear, diameter) is polynomial-time solvable [Bea85]. This result implies that TF(fixed, linear, none) is polynomial-time solvable. Since all item sizes are fixed, a solution with minimal diameter is also a solution with the minimal area and a solution with the minimal white space. Thus, TF(fixed, linear, area) and TF(fixed, linear, w_space) are also polynomial-time solvable. Since TF(fixed, none, diameter), TF(fixed, none, area), and TF(fixed, none, w_space) are subproblems of TF(fixed, linear, diameter), TF(fixed, linear, area), and TF(fixed, linear, w_space), respectively, they are also polynomial-time solvable. For TF(automatic, none, none), we can first fix the item sizes by typesetting all items in their maximum widths; thus, it can be transformed to TF(fixed, none, none), which has been proved to be polynomial-time solvable by Beach [Bea85]. Therefore, TF(automatic, none, none) is polynomial-time solvable.

We have proved that TF(automatic, linear, none) is NP-complete (see Theorem 5.1). If we restrict TF such that the size constraints contain at least two linear equalities: $W_1 \leq \sum_{j=1}^{n} w_j \leq W_2$ and $H_1 \leq \sum_{i=1}^{m} h_i \leq H_2$, it is still NP-complete, because the proof of Theorem 5.1 also holds in this case. We name this NP-complete problem SUBTF. To prove that TF(automatic, linear, diameter) is NP-complete, we can define an equivalent problem of TF(automatic, linear, diameter) by changing the definition of TF in Section 5.3 on page 108 to:

INSTANCE: An $m \times n$ grid, r nonoverlapping items: $o_k = (t_k, l_k, b_k, r_k, \delta_k, \psi_k)$ $(1 \leq k \leq r)$, s size constraints: $e_1, e_2, \ldots, e_s$, and an integer D.

QUESTION: Are there $n + m$ integers $w_1, w_2, \ldots, w_n$ and $h_1, h_2, \ldots, h_m$ such that

1. $W = w_1, w_2, \ldots, w_n$ satisfy all width constraints among $e_1, e_2, \ldots, e_s$;
2. $H = h_1, h_2, \ldots, h_m$ satisfy all height constraints among $e_1, e_2, \ldots, e_s$;
3. $\forall o_k (1 \leq k \leq r)$, $\sum_{p=l_k}^{r_k} w_p \geq \psi_k[min]$ and $\delta_k(\sum_{p=l_k}^{r_k} w_p) \leq \sum_{q=t_k}^{b_k} h_q$.
4. $\sum_{j=1}^{n} w_j + \sum_{i=1}^{m} h_i \leq D$

We add D to the INSTANCE portion and add condition 4, which specifies that the sum of the table width and height is no more than D, to the QUESTION portion. We can prove that TF(automatic, linear, diameter) is NP-complete by reducing SUBTF to this equivalent problem. Similarly, we obtain an equivalent problem of TF(automatic, linear, area) by replacing condition 4 with

$$\sum_{j=1}^{n} w_j \sum_{i=1}^{m} h_i \leq D$$

and obtain an equivalent problem of TF(automatic, linear, w_space)by replacing condition 4 with

$$\sum_{j=1}^{n} w_j \sum_{i=1}^{m} h_i - \sum_{k=1}^{r}((\sum_{p=l_k}^{r_k} w_p)\delta_k(\sum_{p=l_k}^{r_k} w_p)) \leq D.$$

We can also prove that TF(automatic, linear, area) and TF(automatic, linear, w_space) are NP-complete by reducing SUBTF to their equivalent problems. The formal proofs will be given in Wang and Wood [WW96].

We have not classified the complexities of TF(automatic, none, diameter), TF(automatic, none, area), and TF(automatic, none, w_space). These problems may be polynomial-time solvable. The complexity results for all problems that handle size constraints with non-linear expressions are also unknown.

7.7 Formatting algorithms

To obtain an algorithm to solve the tabular formatting problem that runs in polynomial time for many common tables, we ignored objective functions. We can improve our algorithm by generating locally optimal solutions for an objective function among a set of layouts. In a polynomial-time search, we can check all the step combinations and select an optimal solution among the layouts found in the search, rather than terminating when we have found a layout that satisfies the size constraints. A more challenging and interesting future investigation includes the following problems:

- If we take objective functions into account in the formatting process, can we design an algorithm to solve the problem in polynomial time for many tables?

- If we simplify the problem by weakening the size constraints instead of ignoring the objective functions, can we still obtain a polynomial-time algorithm for many tables?

7.8 Large tables

When a table is too large to be presented on a given page, we need to break it into subtables. This process is more complex than the pagination of text. Where should we break the table such that difficulty of reading is minimized? Since our tabular model captures logical structure, we expect that it provides sufficient information to assist in the pagination of tables. For example, if a subcategory has subsubcategories, it is unwise to break a table such that the subsubcategories appear on different pages. To reduce the difficulty of reading a multipage table, we may have to duplicate the labels in the stub or in the boxhead for each page. Observe that when one dimension is much larger than the other dimension, we can break the table with respect to the larger dimension and we may be able to place the subtables side by side in the smaller dimension on one page.

7.9 Tabular browsing

Our tabular model provides a basis for adding tabular browsing in an interactive environment since the model captures the logical structure. Such an extension might highlight the entries that satisfy queries. Here are some example queries:

1. Highlight all marks that are less than 50 and associated with the Winter term and the final examination.

2. Highlight all the students who gained the highest mark in the midterm of a course.

3. Highlight all students whose final marks are between 90 and 100.

We might also wish to create a subtable in response to a query and then automatically lay it out using the methods described in the thesis. We should be able to borrow the ideas in database query languages such as SQL; however, the design of an appropriate query language is an open problem.

Appendix A

Expressiveness

To find out how well the abstract and presentational models described in Chapters 2 and 4 can be used to specify the tables in the real world, we performed two experiments that measure the expressive power of these models. We checked books from different sources, including statistics, sociology, science, and business. The *CRC Handbook of Chemistry and Physics* [CRC88] collects a few hundred tables used in chemistry and physics. These tables are representative of scientific tables that may contain many numbers, mathematical equations, and special symbols. The *Human Activity and the Environment* [Sta86], published by Statistics Canada, contains 148 statistical tables. Most of them contain footnotes and many of them have three or more categories. Most of the tables in *Investments: Principle/Practices/Analyses* [BR74] are two-dimensional numerical tables. *Social Problems* [Rit86] contains many tables with long text.

The result of the experiment for the logical structure, given in Table A.1, reveals that the abstract model can be used to specify 56 percent of the tables in these four books if we consider footnotes and 97 percent of the tables if we ignore footnotes. From this experiment we can see that most of the tables can be specified with a multi-dimensional logical structure.

The result of the experiment for the layout structure, given in Table A.2, shows that the presentational model can be used to specify the topology of 94 percent of the tables in the four books and to specify the style of 97 percent of the tables. These percentages also indicate that the presentational model matches the real-world situation quite well.

Table A.1: The expressiveness of the abstract model.

Books	Logical structure						Total number
	Can be specified				Cannot be specified		
	Without footnotes		With footnotes				
	Number	Percent	Number	Percent	Number	Percent	
CRC Handbook of Chemistry and Physics	322	66	150	31	16	03	488
Human Activity and Environment	23	13	148	86	1	10	172
Investment: Principles/ Practise/Analyses	106	64	56	34	4	2	166
Social Problems	47	78	12	20	1	2	60
Total number	498	56	366	41	22	3	886

Table A.2: The expressiveness of the presentational model.

Books	Topology				Style				Total number
	Can be specified		Cannot be specified		Can be specified		Cannot be specified		
	Number	Percent	Number	Percent	Number	Percent	Number	Percent	
CRC Handbook of Chemistry and Physics	451	92	37	8	463	95	25	5	488
Human Activity and Environment	168	98	4	2	172	1	0	0	172
Investmen: Principles/ Practise/Analyses	161	97	5	3	161	97	5	3	166
Social Problems	57	95	3	5	60	1	0	0	60
Total number	837	94	49	6	856	97	30	3	886

Appendix B

Pseudo-code algorithms

This appendix includes the pseudo-code algorithms invoked by Algorithms 1, 2, and 3 in Chapter 5:

```
Function  Find_Column_Widths(com_steps, column_widths):  bool

    integer pair com_steps[1..item_number];
    var integer column_widths[1..column_number];
begin
    array of inequality wid_inequ;
    integer wid_inequ_num;

    /* Generate width inequalities for item sizes */
    wid_inequ_num := 0;
    for each item $o_k = (t_k, l_k, b_k, r_k, \psi_k, \delta_k)$ do
        /* ensure that the width of the block falls into the step for the item */
        wid_inequ[wid_inequ_num] := { com_steps[k].head $\leq \sum_{p=l_k}^{r_k} w_p$ };
        wid_inequ[wid_inequ_num+1] := { $\sum_{p=l_k}^{r_k} w_p \leq$ com_steps[k].tail };
        wid_inequ_num := wid_inequ_num + 2;
    end for

    /* Generate width equalities and inequalities for width constraints */
    for each width constraint $e_l$ do
```

```
        wid_inequ[wid_inequ_num] := { $e_l$ };
        wid_inequ_num := wid_inequ_num + 1;
    end for

    /* Solve the width equalities and inequalities */
    if Inequality_Solver(wid_inequ, wid_inequ_num, column_widths)) then
        return(true);
    else return(false); end if
end


Function  Find_Row_Heights(com_steps, row_heights):  bool

    integer pair com_steps[1..item_number];
    var integer row_heights[1..row_number];
begin
    array of inequality hei_inequ;
    integer hei_inequ_num;

    /* Generate height inequalities for item sizes */
    hei_inequ_num := 0;
    for each item $o_k = (t_k, l_k, b_k, r_k, \psi_k, \delta_k)$ do
        /* ensure that the height of the item is no more than the height
           of its block */
        hei_inequ[hei_inequ_num] := { $\delta_k$(com_steps[k].head) $\leq \sum_{q=t_k}^{b_k} h_q$ };
        hei_inequ_num := hei_inequ_num + 1;
    end for

    /* Generate height equalities and inequalities for width constraints */
    for each height constraint $e_l$ do
        hei_inequ[hei_inequ_num] := { $e_l$ };
        hei_inequ_num := hei_inequ_num + 1;
    end for

    /* Solve the height equalities and inequalities */
    if Inequality_Solver(hei_inequ, hei_inequ_num, row_heights)) then
        return(true);
    else return(false); end if
end


Function  Find_First_Combination(com_steps, column_widths, row_heights):  enum

    var integer pair com_steps[1..item_number];
    var integer column_widths[1..column_number], row_heights[1..row_number];
begin
    array of inequality wid_inequ;
    integer wid_inequ_num;

    /* Generate width inequalities for item sizes */
    wid_inequ_num := 0;
    for each item $o_k = (t_k, l_k, b_k, r_k, \delta_k, \psi_k)$ do
        /* ensure that the width of the block is no less than the minimal
           step head of the item */
        wid_inequ[wid_inequ_num] := { $\sum_{p=l_k}^{r_k} w_p \geq \psi_k$[min] };
        wid_inequ_num := wid_inequ_num + 1;
    end for

    /* Generate width equalities and inequalities for width constraints */
    for each width constraint $e_l$ do
        wid_inequ[wid_inequ_num] := { $e_l$ };
        wid_inequ_num := wid_inequ_num + 1;
    end for

    /* Solve the width equalities and inequalities */
    if Inequality_Solver(wid_inequ, wid_inequ_num, column_widths)) then
        for k:=1 to column_number do
            Find_Step_Combination(k, column_widths, com_steps);
        end for
        if Find_Row_Heights(com_steps, row_heights) then
```

```
            return(Both_Ok);
        else Find_Layout_Row_Heights(com_steps, row_heights);
             return(Wid_Ok);
        end if
    else return(Not_Found); end if
end



Function  Find_Next_Combination(com_steps, column_widths, row_heights):  enum

    var integer pair com_steps[1..item_number];
    var integer column_widths[1..column_number], row_heights[1..row_number];
begin
    integer pair next_com_steps[1..item_number];
    integer next_column_widths[1..column_number], next_row_heights[1..row_number];
    integer pair selected_com_steps[1..item_number];
    integer selected_col_wids[1..column_number], selected_row_heis[1..row_number];
    integer selected_value, this_value;
    enum {Not_Found, Wid_Ok, Hei_Ok, Both_Ok, None_Ok, End} sel_result;
    bool is_wid_ok, is_hei_ok;

    selected_value := +∞;
    for k = 1 to column_number do
        next_com_steps := com_steps;
        if Find_Widened_Items(k, column_widths, next_com_steps) then
            is_wid_ok := Find_Column_Widths(next_com_steps, next_column_widths);
            is_hei_ok := Find_Row_Heights(next_com_steps, next_row_heights);
            if is_wid_ok and is_hei_ok then
                com_steps := next_com_steps;
                column_widths := next_column_widths;
                row_heights := next_column_widths;
                return(Both_Ok);
            end if
            if not is_wid_ok then
                Find_Layout_Column_Widths(next_com_steps, next_column_widths);
            end if
            if not is_hei_ok then
                Find_Layout_Row_Heights(next_com_steps, next_row_heights);
            end if
            this_value := $\sum_{j=1}^{n}$next_column_widths[j] + $\sum_{i=1}^{m}$next_row_heights[i];
            if this_value $<$ selected_value then
                selected_value := this_value;
                selected_com_steps := next_com_steps;
                selected_col_wids := next_column_widths;
                selected_row_heis := next_row_heights;
                if not is_wid_ok and not is_hei_ok then
                    sel_result := None_Ok;
                else if is_wid_ok then
                         sel_result := Wid_Ok;
                     else sel_result := Hei_Ok; end if;
            end if
        end if
    end for
    if selected_value $\neq$ +$\infty$ then
        com_steps := selected_com_steps;
        column_widths := selected_col_wids;
        row_heights := selected_row_heis;
        return(sel_result);
    else return(End); end if

end
```

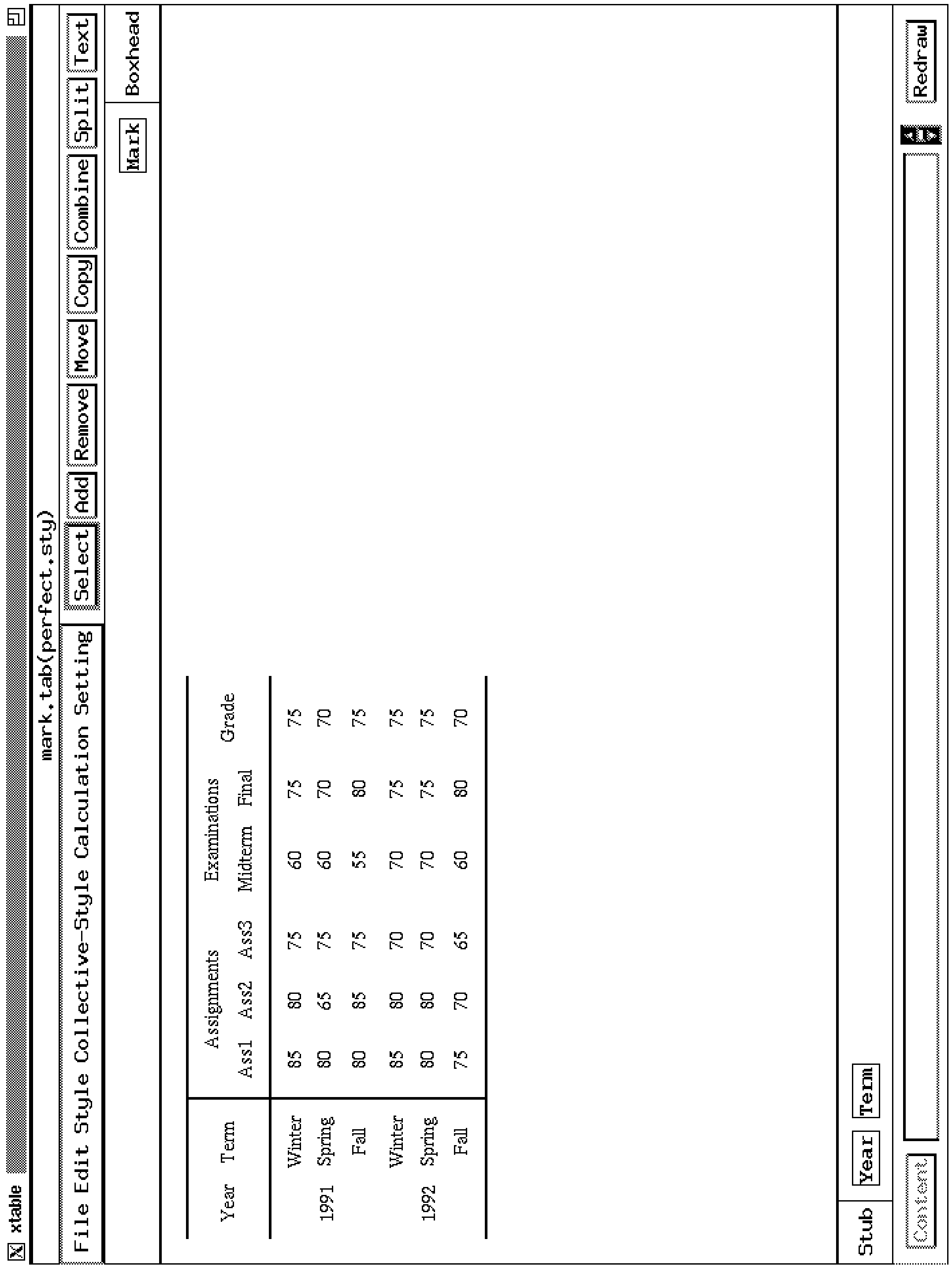
```
Function  Find_Widened_Items(column, column_widths, com_steps):  bool

    integer column, column_widths[1..column_number];
    var integer pair com_steps[1..item_number];
begin
    bool found;
    integer other_width, this_step_head, next_width;
    /* calculate the new width for the column based on current column widths */
    found := false;
    next_width := +$\infty$;
    for each item $o_k = (t_k, l_k, b_k, r_k, \delta_k, \psi_k)$ that satisfies $l_k \leq$ column $\leq r_k$ do
        if com_steps[k].head $\neq$ $\psi_k$[max] then
            this_step_head := com_steps[k].tail + 1;
            other_width := $\sum_{p=l_k}^{r_k}$column_widths[$p$] - column_widths[column];
            if (this_step_head - other_width) $<$ next_width then
                next_width := this_step_head - other_width;
            end if
            found := true;
        end if
    end for
    /* change the steps of the items for the new column width */
    if found then
        column_widths[column] := next_width;
        Find_Step_Combination(column, column_widths, com_steps);
    end if

    return(found);
end


Procedure  Find_Step_Combination(column, current_widths, com_steps)

    integer column, column_widths;
    var integer pair com_steps[1..item_number];
begin
    integer block_width;

    for each item $o_k = (t_k, l_k, b_k, r_k, \delta_k, \psi_k)$ that satisfies $l_k \leq$ column $\leq r_k$ do
        block_width :=$\sum_{p=l_k}^{r_k}$column_widths[p];
        com_steps[k] := step $s_k$ such that $s_k$.head $\leq$ block_width $\leq$ $s_k$.tail;
    end for
```

```
end


Function  Find_Layout_Column_Widths(com_steps, column_widths):  bool

    integer pair com_steps[1..item_number];
    var integer column_widths[1..column_number];
begin
    array of inequality wid_inequ;
    integer wid_inequ_num;

    /* Generate width inequalities for item sizes */
    wid_inequ_num := 0;
    for each item $o_k = (t_k, l_k, b_k, r_k, \delta_k, \psi_k)$ do
        wid_inequ[wid_inequ_num] := { com_steps[k].head $\leq \sum_{p=l_k}^{r_k} w_p$ };
        wid_inequ[wid_inequ_num+1] := { $\sum_{p=l_k}^{r_k} w_p \leq$ com_steps[k].tail };
        wid_inequ_num := wid_inequ_num + 2;
    end for

    /* Solve the width inequalities */
    if Inequality_Solver(wid_inequ, wid_inequ_num, column_widths) then
        return(true);
    else return(false); end if
end


Function  Find_Layout_Row_Heights(com_steps, row_heights):  bool

    integer pair com_steps[1..item_number];
    var integer row_heights[1..row_number];
begin
    array of inequality hei_inequ;
    integer hei_inequ_num;

    /* Generate height inequalities for item sizes */
    hei_inequ_num := 0;
    for each item $o_k = (t_k, l_k, b_k, r_k, \delta_k, \psi_k)$ do
        hei_inequ[hei_inequ_num] := { $\sum_{q=t_k}^{b_k} h_q \geq \delta_k$(com_steps[k].head)};
        hei_inequ_num := hei_inequ_num + 1;
    end for

    /* Solve the height inequalities */
    if Inequality_Solver(hei_inequ, hei_inequ_num, row_heights)) then
        return(true);
    else return(false); end if
end
```

Appendix C

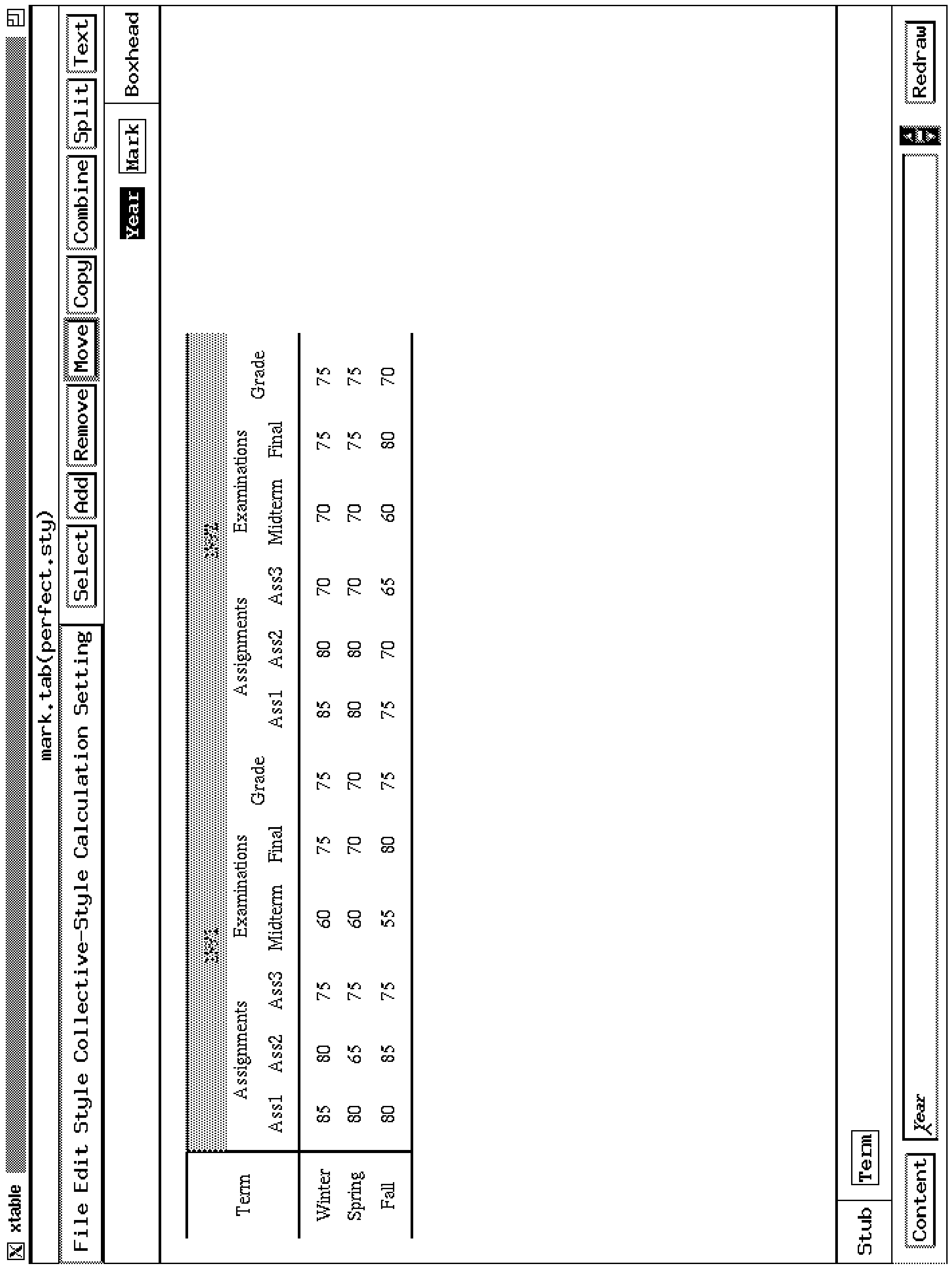
Screen shots of XTABLE

This appendix includes a number of screen shots that shows how users edit tables using XTABLE. The operations that generate these screen shots are given in Section 6.5.1.


xtable
mark.tab(perfect.sty)
File Edit Style Collective-Style Calculation Setting
Select Add Remove Move Copy Combine Split Text
Mark
Boxhead
Stub
Year
Term
Content
Redraw

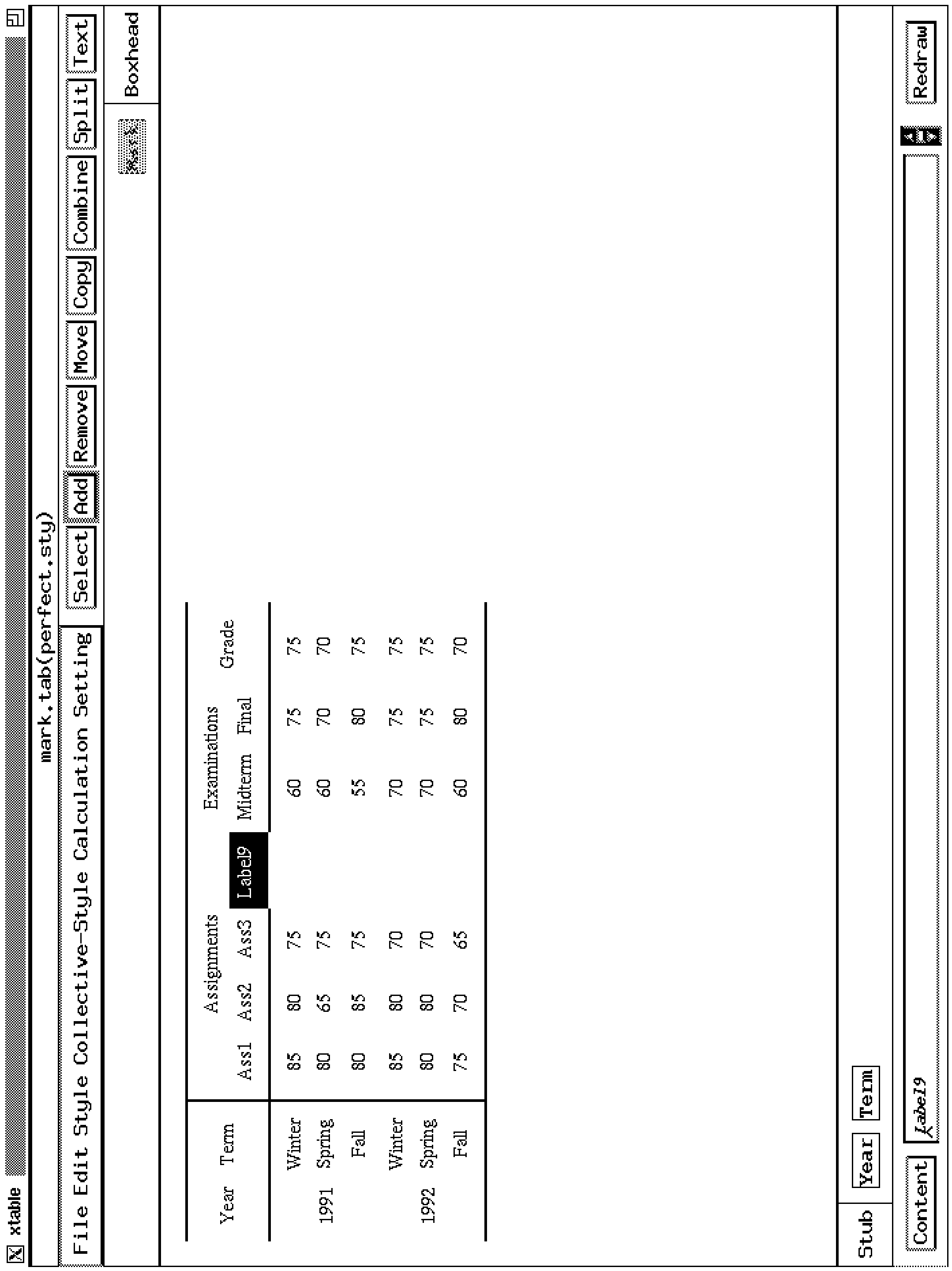
Year	Term	Assignments			Examinations		Grade
		Ass1	Ass2	Ass3	Midterm	Final	
	Winter	85	80	75	60	75	75
1991	Spring	80	65	75	60	70	70
	Fall	80	85	75	55	80	75
	Winter	85	80	70	70	75	75
1992	Spring	80	80	70	70	75	75
	Fall	75	70	65	60	80	70

Figure C.1: The original three-dimensional table.


xtable
mark.tab(perfect.sty)
File Edit Style Collective-Style Calculation Setting
Select Add Remove Move Copy Combine Split Text
Boxhead Year Mark
Stub Term
Content Year
Redraw

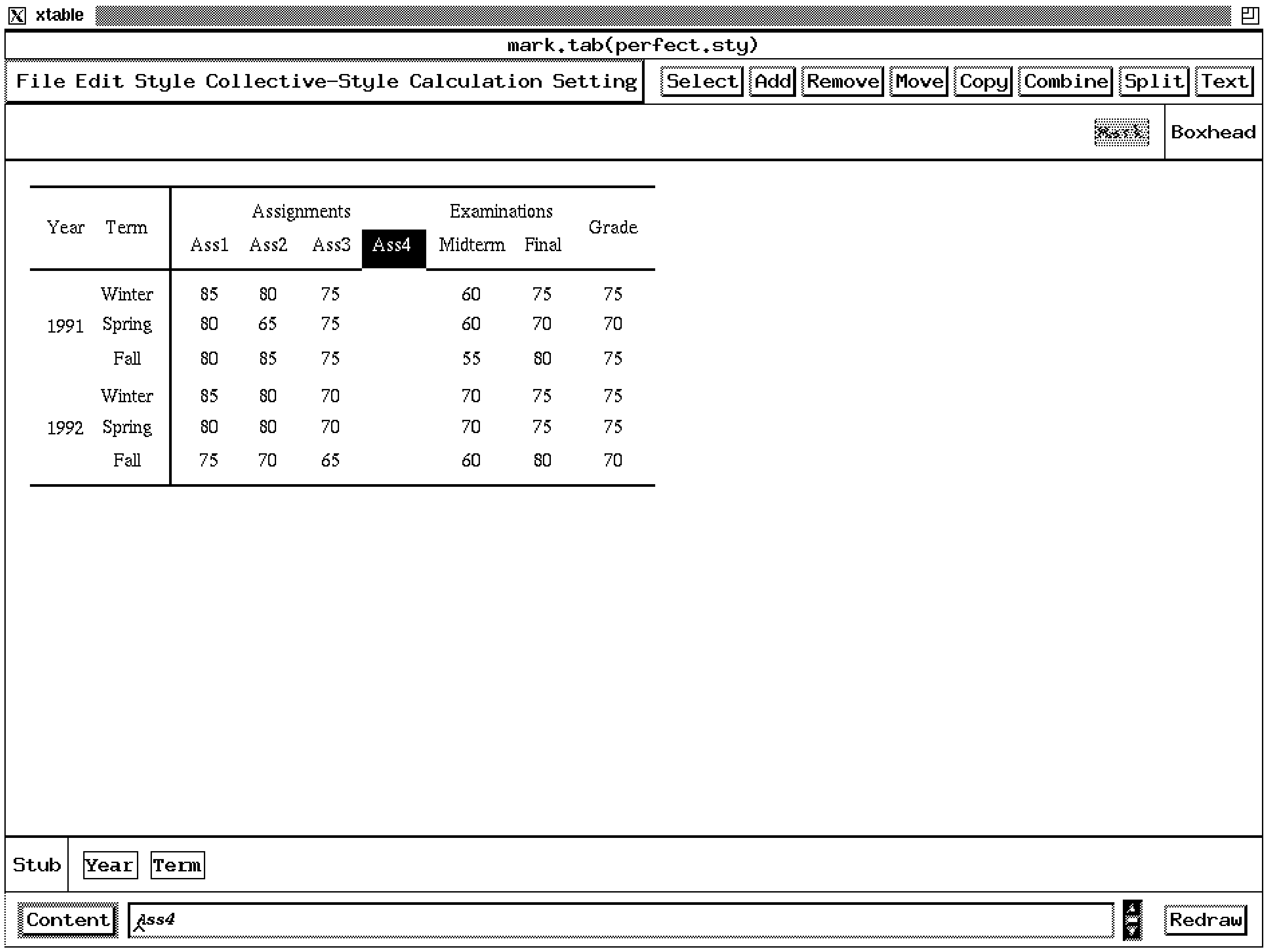
Term	1991						1992					
	Assignments			Examinations		Grade	Assignments			Examinations		Grade
	Ass1	Ass2	Ass3	Midterm	Final		Ass1	Ass2	Ass3	Midterm	Final	
Winter	85	80	75	60	75	75	85	80	70	70	75	75
Spring	80	65	75	60	70	70	80	80	70	70	75	75
Fall	80	85	75	55	80	75	75	70	65	60	80	70

Figure C.2: After moving the category **Year** to the boxhead.



Year	Term	Assignments				Examinations		Grade
		Ass1	Ass2	Ass3	Label9	Midterm	Final	
	Winter	85	80	75		60	75	75
1991	Spring	80	65	75		60	70	70
	Fall	80	85	75		55	80	75
	Winter	85	80	70		70	75	75
1992	Spring	80	80	70		70	75	75
	Fall	75	70	65		60	80	70

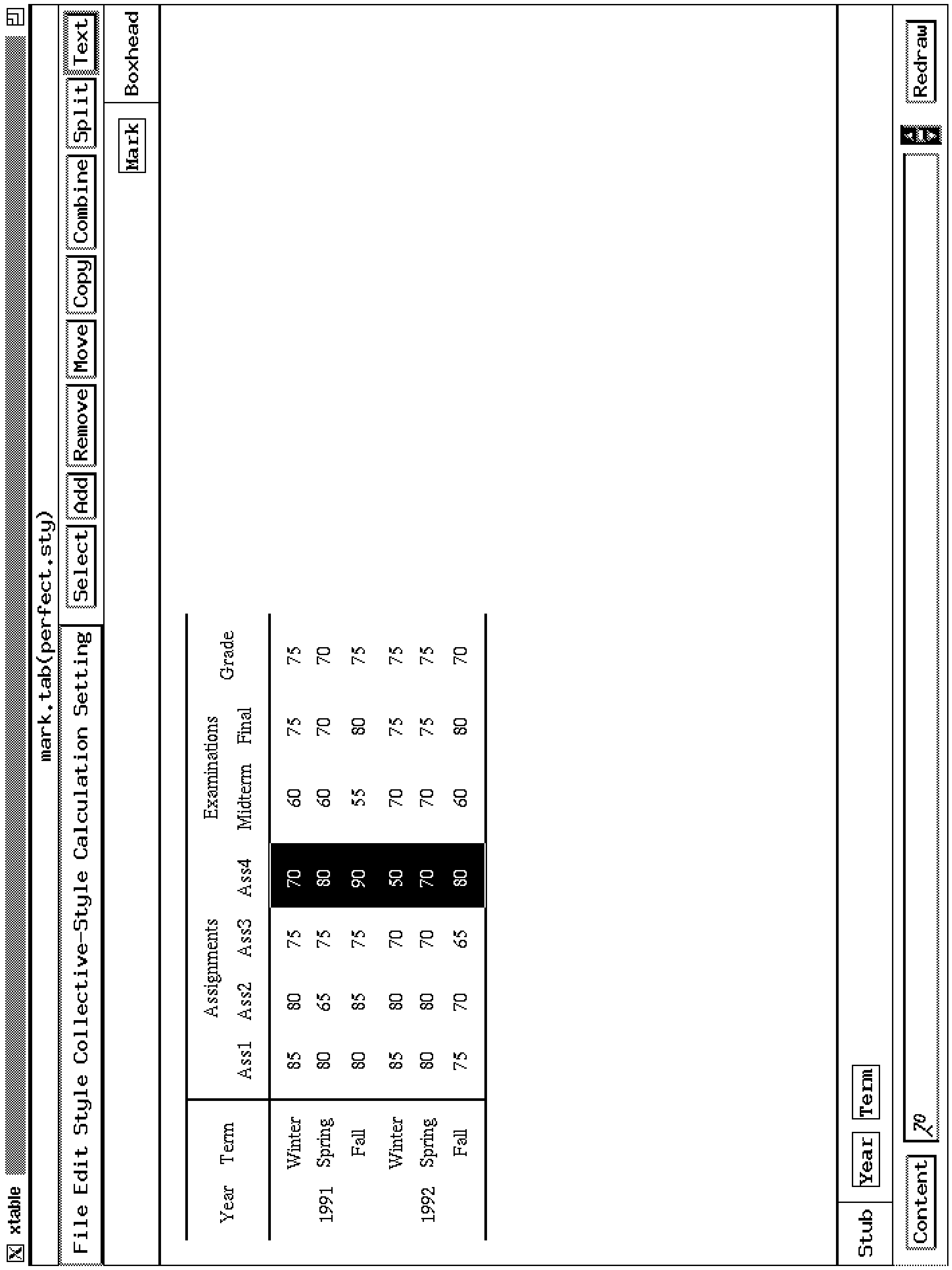
Figure C.3: After adding a new label under the subcategory **Assignments**.



Year	Term	Assignments				Examinations		Grade
		Ass1	Ass2	Ass3	Ass4	Midterm	Final	
	Winter	85	80	75		60	75	75
1991	Spring	80	65	75		60	70	70
	Fall	80	85	75		55	80	75
	Winter	85	80	70		70	75	75
1992	Spring	80	80	70		70	75	75
	Fall	75	70	65		60	80	70

Ass4

Figure C.4: After assigning the name **Ass4** for the new label.


xtable
mark.tab(perfect.sty)
File Edit Style Collective-Style Calculation Setting
Select Add Remove Move Copy Combine Split Text
Mark
Boxhead
Stub
Year
Term
Content
70
Redraw

Year	Term	Assignments				Examinations		Grade
		Ass1	Ass2	Ass3	Ass4	Midterm	Final	
	Winter	85	80	75	70	60	75	75
1991	Spring	80	65	75	80	60	70	70
	Fall	80	85	75	90	55	80	75
	Winter	85	80	70	50	70	75	75
1992	Spring	80	80	70	70	70	75	75
	Fall	75	70	65	80	60	80	70

Figure C.5: After entering the marks associated with **Ass4**.

Appendix D

Examples of XTABLE's input files

This appendix gives examples of a table file and a collective style file. We specify an abstract table, a topological specification, and a specific style specification in a table file, and specify a collective style specification in a collective style file. The expressive methods, however, are different from the ways we use in Chapters 2 and 4. For example, a label in Chapter 2 is a string of characters, whereas a label in a table file consists of a unique ID assigned by the system and a value shown on the screen. In Chapter 4 we use pseudo-code to specify the style rules. In a table file or a collective style file, however, we specify style rules in a less readable way; for instance, the cell style is specified as:

CELL = (<font>, <slant>, <shape>, <size>, <line space>, <vertical alignment>,
    <horizontal alignment>, <background pattern>, <left leading space>,
    <right leading space>, <top leading space>, <bottom leading space>)

and a separation style, say vertical separation, is specified as

VER_RULE = (<line type>, <width>, <left space>, <right space>).

The keyword INHERITANCE indicates that the option is inherited from the super object or the default value if the style rule is specified for the whole table.

D.1 An example of a table file

```
TABLE mark
BEGIN

ABSTRACTION

CATEGORY Year|"Year":
   {_X271343872|"1991", _X271343936|"1992"};
CATEGORY Term|"Term":
   {_X271344256|"Winter", _X271344384|"Spring", _X271344448|"Fall"};
CATEGORY Mark|"Mark":
   {_X270847808|"Assignments", _X270847744|"Examinations", _X271344064|"Grade"};
SUBCATEGORY Mark._X270847744|"Examinations":
   {_X270847936|"Midterm", _X271343616|"Final"};
SUBCATEGORY Mark._X270847808|"Assignments":
   {_X270847488|"Ass1", _X270847552|"Ass2", _X270847680|"Ass3"};

MAPPING:
{{ Year._X271343936, Term._X271344448, Mark._X271344064 } -> "70",
 { Year._X271343936, Term._X271344384, Mark._X271344064 } -> "75",
 { Year._X271343936, Term._X271344256, Mark._X271344064 } -> "75",
 { Year._X271343872, Term._X271344448, Mark._X271344064 } -> "75",
 { Year._X271343872, Term._X271344384, Mark._X271344064 } -> "70",
 { Year._X271343872, Term._X271344256, Mark._X271344064 } -> "75",
 { Year._X271343936, Term._X271344448, Mark._X270847744._X271343616 } -> "80",
 { Year._X271343936, Term._X271344384, Mark._X270847744._X271343616 } -> "75",
 { Year._X271343936, Term._X271344256, Mark._X270847744._X271343616 } -> "75",
 { Year._X271343872, Term._X271344448, Mark._X270847744._X271343616 } -> "80",
 { Year._X271343872, Term._X271344384, Mark._X270847744._X271343616 } -> "70",
 { Year._X271343872, Term._X271344256, Mark._X270847744._X271343616 } -> "75",
 { Year._X271343936, Term._X271344448, Mark._X270847744._X270847936 } -> "60",
 { Year._X271343936, Term._X271344384, Mark._X270847744._X270847936 } -> "70",
 { Year._X271343936, Term._X271344256, Mark._X270847744._X270847936 } -> "70",
 { Year._X271343872, Term._X271344448, Mark._X270847744._X270847936 } -> "55",
 { Year._X271343872, Term._X271344384, Mark._X270847744._X270847936 } -> "60",
 { Year._X271343872, Term._X271344256, Mark._X270847744._X270847936 } -> "60",
 { Year._X271343936, Term._X271344448, Mark._X270847808._X270847680 } -> "65",
 { Year._X271343936, Term._X271344384, Mark._X270847808._X270847680 } -> "70",
 { Year._X271343936, Term._X271344256, Mark._X270847808._X270847680 } -> "70",
 { Year._X271343872, Term._X271344448, Mark._X270847808._X270847680 } -> "75",
 { Year._X271343872, Term._X271344384, Mark._X270847808._X270847680 } -> "75",
 { Year._X271343872, Term._X271344256, Mark._X270847808._X270847680 } -> "75",
 { Year._X271343936, Term._X271344448, Mark._X270847808._X270847552 } -> "70",
 { Year._X271343936, Term._X271344384, Mark._X270847808._X270847552 } -> "80",
 { Year._X271343936, Term._X271344256, Mark._X270847808._X270847552 } -> "80",
 { Year._X271343872, Term._X271344448, Mark._X270847808._X270847552 } -> "85",
 { Year._X271343872, Term._X271344384, Mark._X270847808._X270847552 } -> "65",
 { Year._X271343872, Term._X271344256, Mark._X270847808._X270847552 } -> "80",
 { Year._X271343936, Term._X271344448, Mark._X270847808._X270847488 } -> "75",
 { Year._X271343936, Term._X271344384, Mark._X270847808._X270847488 } -> "80",
 { Year._X271343936, Term._X271344256, Mark._X270847808._X270847488 } -> "85",
 { Year._X271343872, Term._X271344448, Mark._X270847808._X270847488 } -> "80",
 { Year._X271343872, Term._X271344384, Mark._X270847808._X270847488 } -> "80",
 { Year._X271343872, Term._X271344256, Mark._X270847808._X270847488 } -> "85"
};

TOPOLOGY

STUB: Year > Term;
BOXHEAD: Mark;

STYLE

BOXHEAD:
   CELL={ INHERITANCE, INHERITANCE, INHERITANCE, INHERITANCE, INHERITANCE,
          INHERITANCE, BASELINE, INHERITANCE, INHERITANCE, INHERITANCE,
          INHERITANCE, INHERITANCE };

END
```

D.2 An example of a collective style file

```
GLOBAL_STYLE

TABLE:
   CATEGORY_HEAD_TYPE = WITHOUT_HEAD,
   STUB_RULE = { SINGLE, 24 , INHERITANCE , INHERITANCE },
   BOXHEAD_RULE = { SINGLE, 24 , INHERITANCE , INHERITANCE },
   BOX_LEFT_RULE = { DOUBLE, 24 , INHERITANCE , INHERITANCE },
   BOX_TOP_RULE = { DOUBLE, 24 , INHERITANCE , INHERITANCE },
   BOX_RIGHT_RULE = { DOUBLE, 24 , INHERITANCE , INHERITANCE },
   BOX_BOTTOM_RULE = { DOUBLE, 24 , INHERITANCE , INHERITANCE };

STUB:
   CELL = { INHERITANCE, INHERITANCE, BOLD, INHERITANCE, INHERITANCE,
            INHERITANCE, INHERITANCE, INHERITANCE, INHERITANCE,
            INHERITANCE, INHERITANCE, INHERITANCE },
   HOR_RULE = { NONE, INHERITANCE , INHERITANCE , INHERITANCE };

BOXHEAD:
   CELL = { INHERITANCE, INHERITANCE, BOLD, INHERITANCE, INHERITANCE,
            INHERITANCE, INHERITANCE, INHERITANCE, INHERITANCE,
            INHERITANCE, INHERITANCE, INHERITANCE },
   VER_RULE = { NONE, INHERITANCE , INHERITANCE , INHERITANCE };

BODY:
   TYPE = VERTICAL_SPANNING_ONLY,
   HOR_RULE = { SINGLE, INHERITANCE , INHERITANCE , INHERITANCE };

ENTRY_VALUE "" :
   CELL = { INHERITANCE, INHERITANCE, INHERITANCE, INHERITANCE,
            INHERITANCE, INHERITANCE, INHERITANCE, LIGHT_GRAY,
            INHERITANCE, INHERITANCE, INHERITANCE, INHERITANCE };
```

Bibliography

[AAU78] *One book/Five ways.* William Kaufmann, Inc., Los Altos, CA, 1978.

[Bar65] M. P. Barnett. *Computer Typesetting: Experiments and Prospects.* MIT Press, 1965.

[Bea85] 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.

[BEF84] 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.

[BR74] D. H. Bellemore and J. C. Ritchie. *Investments — Principle/Pratices/Analyses.* South-Western Publishing Co., 4th edition, 1974.

[Bri14] W. C. Brinton. *Graphic Methods for Presenting Facts.* The Engineering Magazine Company, New York, 1914.

[Cam89] 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.

[Cea82] D. C. Chamberlin and et al. JANUS: An interactive document formatter based on declarative tags. *IBM Systems Journal*, 21(3), 1982.

[Chi93] *The Chicago Manual of Style.* The University of Chicago Press, Chicago and London, 14th edition, 1993.

[CRC88] *CRC Handbook of Chemistry and Physics.* CRC Press, 68th edition, 1987-1988.

[Dan63] G. Dantzig. *Linear Programming and Extensions.* Princeton University Press, 1963.

[DHQ94] Shona Douglas, Matthew Hurst, and David Quinn. Using natural language processing for identifying and interpreting tables in plain text. Construction Industry Specification Analysis and Understanding System (CISAU) Project No: IED4/1/5818, December 1994.

[EE68] D. C. Engelbart and W. K. English. A research center for augmenting human intellect. *AFIPS Conference Proceedings*, 33, 1968.

[Ehr77] A. S. C. Ehrenberg. Rudiments of numeracy. *Journal of the Royal Statistical Society*, A. 140, part 3:277–297, 1977.

[FPSSU95] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining.* AAAI Press/The MIT Press, 1995.

[Fur82] R. Furuta. Document formatting systems: Survey, concepts, and issues. *Computing Surveys*, 14(3):417–472, September 1982.

[Fur86] 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.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, NY, 1979.

[Hal43] R. O. Hall. *Handbook of Tabular Presentation.* The Ronald Press Company, New York, 1943.

[Ils80] R. Ilson. An integrated approach to formatted document production. Technical Report MIT/LCS/TR-253, Laboratory for Computer Science, Massachusetts Institute of Technology, August 1980. Master thesis.

[Imp91] *Improv Handbook.* Lotus Development Corporation, Cambridge, MA, 1991.

[Int86] International Organization for Standardization. *ISO 8879, Information processing — Text and office systems — Standard Generalized Markup Language(SGML)*, October 1986.

[Int88] International Organization for Standardization and International Electrotechnical Commission. *ISO/IEC TR 9573:1988(E), Information processing — SGML Support Facilities — Techniques for Using SGML*, 1988.

[Int92] 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.

[Knu84] D. E. Knuth. *The TEXbook.* Addison-Wesley, Reading, MA, 1984.

[Lam85] L. Lamport. *LATEX: A Document Preparation System.* Addison-Wesley, Reading, MA, 1985.

[Les79] 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.

[Lot84] *Lotus 1-2-3 User's Handbook.* Ballantine Books, New York, NY, 1984.

[MS-90] *Microsoft Excel User's Guide.* Microsoft Corporation, Redmond, WA, 1990.

[Nor89] P. Norrish. Semantic structures of text. In R. Furuta J. André and V. Quint, editors, *Structured Documents.* 1989.

[Oss76] J. F. Ossanna. Nroff/troff user's manual. Computing Science Technical Report 54, Bell Laboratories, Murray Hill, NJ, 1976.

[Phi68] A. Phillips. *Computer Peripherals and Typesetting.* Her Majesty's Stationery Office, 1968.

[PLSS84] M. Powers, C. Lashley, P. Sanchez, and B. Shneiderman. An experimental comparison of tabular and graphic data presentation. *International Journal of Man-Machine Studies*, 20:545–566, 1984.

[QV86] 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.

[Ray96] D. R. Raymond. *Partial Order Databases.* PhD thesis, Dept. of Computer Science, University of Washington, Waterloo, Ontario, Canada, 1996.

[Rei80] B. K. Reid. *Scribe: A Document Specification Language and its Compiler.* PhD thesis, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, October 1980. Also issued as Technical Report CMU-CS-81-100, Carnegie-Mellon University.

[Rit86] George Ritzer. *Social Problems.* Random House, new York, 2nd edition, 1986.

[Rub88] R. Rubinstain. *Digital Typography: An Introduction to Type and Composition for Computer System Design.* Addison Wesley, Reading, MA, 1988.

[SKS94] K. Shin, K. Kobayashi, and A. Suzuki. TAFEL MUSIK, formating algorithm of tables. In *Principles of Document Processing'94*, pages 1–25, Lufthansa Training Center, Seeheim, May 1994.

[SL67] M. E. Stevens and J. L. Little. Automatic typographic-quality typesetting techniques: A state-of-the-art review. *NBS Monograph*, 99, April 1967.

[Spe68] H. Spencer. *The Visible Word.* Times Drawing Office Ltd, London, 1968.

[SSK94] K. Shin, A. Suzuki, and K. Kobayashi. Data model for retrieving tabular data structures and formatting techniques for retrieved structures. In preparation, 1994.

[Sta86] *Human Activity and the Environment — A statistical compendium.* Statistics Canada, 1986.

[SW84] M. T. Swanston and C. E. Walley. Factors affecting the speed of acquisition of tabulated information from visual displays. *Ergonomics*, 27(3):321–330, 1984.

[Tei84] W. Teitelman. The cedar programming environment: A midterm report and examination. Xerox PARC Technical Report CSL-83-11, June 1984.

[Tin30] M. A. Tinker. The relative legibility of modern and old style numerals. *Journal of Experimental Psychology*, 13:453–461, 1930.

[Tin60] M. A. Tinker. Legibility of mathematical tables. *Journal of Applied Psychology*, 44:83–87, 1960.

[Tuf83] E. R. Tufte. *The visual Display of Quantitative Information.* Graphics Press, Cheshire, Connecticut, 1983.

[Tuf90] E. R. Tufte. *Envisioning Information.* Graphics Press, Cheshire, Connecticut, 1990.

[Van92] 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.

[WF70] P. Wright and K. Fox. Presenting information in tables. *Applied Ergonomics*, 1(1):234–242, 1970.

[Wil83] H. Williamson. *Methods of Book Design: The Practice of An Industrial Craft.* Yale University Press, New Haven and London, 3rd edition, 1983.

[Wri68] P. Wright. Using tabulated information. *Ergonomics*, 11(4):331–343, 1968.

[Wri73] P. Wright. Understanding tabular displays. *Visible Language*, 7:351–359, 1973.

[Wri77] P. Wright. Decision making as a factor in the ease of using numerical tables. *Ergonomics*, 20:91–96, 1977.

[Wri80a] P. Wright. The comprehension of tabulated information: Some similarities between reading prose and reading tables. *NSPI Journal*, XIX(8):25–29, October 1980.

[Wri80b] P. Wright. Tables in text: the subskill needed for reading formatted information. In L. John Chapman, editor, *The Reader and The Text*. 1980.

[Wri82] P. Wright. A user-oriented approach to the design of tables and flowcharts. In *The Technology of Text, Principles for Structuring, Designing, and Displaying text*. Educational Technology Publications, Englewood Cliffs, NJ, 1982.

[WW93] X. Wang and D. Wood. An abstract model for tables. TUGBOAT, *The communications of the TEX Users Group*, 14(3):231–237, October 1993.

[WW96] X. Wang and D. Wood. Complexity results for tabular formatting problems. In preparation, 1996.

[Zei85] H. Zeisel. *Say It With Figures*. Harper & Row, Publishers, New York, NY, 6th edition, 1985.

Index