Computer Languages, Systems & Structures 50 (2017) 31–52

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

Financial information description language and visualization/analysis tools

Youichi Horry

Matsudo Research Center, Industry and Water Business Administration Division, Hitachi, Ltd., 537 Kami Hongo, Matsudo, Chiba 271-5296, Japan

ARTICLE INFO

Article history:
Received 22 December 2016
Revised 31 May 2017
Accepted 31 May 2017
Available online 15 June 2017

ABSTRACT

It has been known for some time that understanding the content of spreadsheet software used by many enterprises and organizations and discovering errors in that content is a difficult task. In particular, it is known that it takes time to create a spreadsheet, and many errors are included in the spreadsheet due to mistakes of the creator. In response to this problem, we have developed a financial information description language and associated visualization and analysis tools to facilitate the understanding of inter-element structure and simplify various types of analyses. A key feature of this development is clear separation between the input of formulas and numerical values and the output of calculation results and between operators and operands in formulas. The financial information description language describes input information such as item values and relationships in a simple format that enumerates the name, operators, and operands of each item. The visualization and analysis tools, meanwhile, perform calculations and output results in table or graph form. In a comparison experiment performed with conventional spreadsheet software, the proposed method can efficiently create a calculation sheet in a short time and has the effect of reducing errors.

1. Introduction

Spreadsheet software has come to be used in a wide variety of organizations [1–3], and it is said that about 10 million people create more than a hundred million spreadsheets every year [4]. According to a survey by Gable et al. [5], 84% of users feel that spreadsheet software is extremely important or important and that 96% of those users would recreate a spreadsheet that was somehow lost. In addition, spreadsheets are widely used in mission-critical decisions [6] and more than 88% of users use them in reports to management on the vice-president and higher level [2]. At the same time, the number of basic units or *cells* making up a spreadsheet can range from several thousand to ten thousand [7,8] and formulas are frequently used. Hall [9] reports that macros are used in 45% of spreadsheets and that spreadsheets are generally becoming more complex in layout as a result of absolute and relative references, logic that includes conditional branching, links with other software programs and databases, etc.

It is also known that spreadsheets include many errors [1,4,10–14]. The results compiled by Panko [4] indicate that errors exist in 86% of spreadsheets and that the percentage of all cells having an error (cell error rate) can run as high as 2.5%.

E-mail address: youichi.horii.jx@hitachi.com

http://dx.doi.org/10.1016/j.cl.2017.05.005

In other words, a large-scale spreadsheet using a large number of cells has a high probability of containing errors. The cause of these errors is not simply carelessness on the part of the spreadsheet creator—it also lies in the limitations of human cognition, a more fundamental problem. It is known that humans will make errors at a rate of about 1/200 when faced with a simple task such as pushing the right switch and at a rate of about 1/50–1/20 in more complex work such as programming [4]. Input degree of freedom is particularly high in the case of spreadsheet software where much is left to the discretion of the creator. In work such as this, the occurrence of errors is inevitable.

One specific reason as to why errors occur is that the spreadsheet creator falsely recognizes what type of value a cell indicates. In most spreadsheet software, references and formulas are described using cell addresses. However, as Abraham [15] and Roy [16] have pointed out, it is not a trivial task to automatically extract the value that each cell indicates. In general, the left-most and top-most cells contain row and column descriptions. Consequently, for tables that are exceptionally long in the horizontal or vertical direction, the user may find it necessary at times to scroll back to the top or far left of the spreadsheet or to set the first row or column to be a non-scrolling target. There is also spreadsheet software having a function for assigning names to each cell, but the work of doing so to all cells in a large spreadsheet can be formidable, and since those names will not be reflected when printing out the spreadsheet, such a function is not that effective in actual use. Furthermore, a function for automatically moving relative references when inserting/deleting rows or columns or performing copy-and-paste operations is generally effective, but it may sometimes move a reference contrary to the creator's intention. In the above ways, it can be time consuming to understand the structure of a complicated spreadsheet, particularly one created by another person, which make error discovery all the more difficult. As described later in Section 2, techniques to help understanding the contents of spreadsheets and methods to reduce errors have been studied. However, while the need for some guidelines in the creation of spreadsheets has been pointed out, there are still fundamental problems inherent to spreadsheet software. One of these is that the same cell is used for both input and output. In many spreadsheet software, a cell in which a computational expression has been input will ordinarily display the result of that computation, although selecting that cell will display the expression. Such specifications are intuitive in nature and have come to be accepted by most users. They are not, however, suitable for obtaining a comprehensive view of computational logic described in a complicated and huge collection of cells. Against this problem, Lotus Improv [17] was an attempt to redefine the way a spreadsheet program should work. It separated the concept of data, views of the data, and formula into three portions. In addition, formulas in conventional spreadsheets are described as a combination of operators and operands. This is a rational method of description for understanding a formula, but when referencing cells of other sheets or files, the resulting character strings can be long hampering legibility.

Another problem is that only one value can be set in each cell. To give calculations a range of fluctuation, that formula needs to be defined using another cell making the spreadsheet all the more complicated. Moreover, when performing sensitivity analysis to investigate the results of varying numerical values in steps or when changing the currency or unit to be displayed, all related cells must be dealt with accordingly. Finally, visualization of calculation results using graphs or other means can be time consuming.

This paper describes the financial information description language (referred to below as "FDL") and associated visualization and analysis tools that we have developed to solve these problems. A key feature of FDL is clear separations (1) between the input of formulas and numerical values and the output of calculation results and (2) between operators and operands in formulas. Compared to conventional spreadsheet software, the creation time was reduced by about 40% and the number of errors reduced by about 83%.

The remainder of this paper is organized as follows. Section 2 describes related work especially on domain-specific languages. Sections 3 and 4 explain the basic concept and details of FDL. Section 5 introduces visualization and analysis tools using FDL. Experiments comparing FDL and conventional spreadsheet software are described in Section 6, and Section 7 concludes.

2. Related work

The algorithms for extracting spreadsheet errors have come to be researched [18–24]. For example, Abraham and others [15,16,25,26] infer the data, header, and footer areas of a spreadsheet, infer hierarchical metadata that indicates data content, and extract errors based on compliance with metadata description rules. Relationships of metadata are restricted to trees, preventing DAGs (directed acyclic graphs).

Additionally, to support the understanding of spreadsheet structure, research has been performed on extracting concepts from spreadsheets created by scientists [27] and techniques for extracting metadata for search purposes have been attempted [28–30]. Other proposals include design methods [31–33], a cognitive approach [34], a test method [35], and visualization of spreadsheet dataflow [36].

The domain-specific language (DSL) is tailored to a specific application domain [37,38], and FDL can also be classified as DSL. Recently, compared to general-purpose languages (GPLs), the efficiency and accuracy of DSLs has been investigated.

For example, Kosar et al. [39] investigated the difference in understanding of text-based DSL and GPL programs. In software development, more than 28% of the developer's time is spent on learning and understanding of the program. In their experiments, the participants were asked to analyze the source code of DSL and GPL on the problem of feature diagram, graph descriptions, and graphical user interface, and let them answer the tasks shown on the paper. As a result, they reported that using DSL is more accurate and effective than GPL.