Discovery-based Edit Assistance for Spreadsheets

Jácome Cunha, João Saraiva
Departamento de Informática
Universidade do Minho
Portugal
{jacome,jas}@di.uminho.pt

Joost Visser
Software Improvement Group
The Netherlands
j.visser@sig.nl

Abstract

Spreadsheets can be viewed as a highly flexible end-users programming environment which enjoys wide-spread adoption. But spreadsheets lack many of the structured programming concepts of regular programming paradigms. In particular, the lack of data structures in spreadsheets may lead spreadsheet users to cause redundancy, loss, or corruption of data during edit actions.

In this paper, we demonstrate how implicit structural properties of spreadsheet data can be exploited to offer edit assistance to spreadsheet users. Our approach is based on the discovery of functional dependencies among data items which allow automatic reconstruction of a relational database schema. From this schema, new formulas and visual objects are embedded into the spreadsheet to offer features for auto-completion, guarded deletion, and controlled insertion. Schema discovery and spreadsheet enhancement are carried out automatically in the background and do not disturb normal user experience.

1. Introduction

Recent advances in programing languages extend naive editors, to powerful language-based environments [12, 13, 15, 17]. Language-based environments use *knowledge* of the programming language to provide the users with more powerful mechanisms to develop their programs. This knowledge is based on the *structure* and the *meaning* of the language. To be more precise, it is based on the syntactic and (static) semantic characteristics of the language. Having this knowledge about a language, the language-based environment is not only able to highlight keywords and beautify programs, but it can also detect features of the programs being edited that, for example, violate the properties of the underlying language. Furthermore, a language-based environment may also give information to the user about properties of the program under consideration. Consequently, language-based environments guide the user in writing correct programs.

Spreadsheet systems can be viewed as programming environments for non-professional programmers. In this paper, we propose a technique to enhance a spreadsheet system with mechanisms to guide end-users to introduce correct data. An overview of the approach is shown in Figure 1. A background process adds formulas and visual objects

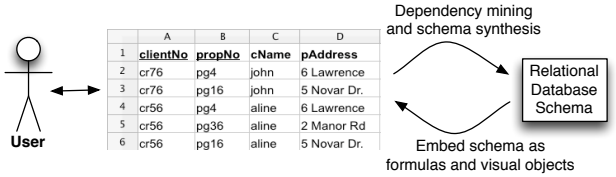

Figure 1. Edit assistance is added to an existing spreadsheet based on a relational database schema obtained by data mining.

to an existing spreadsheet, based on a relational database schema. To obtain this schema, we follow the approach used in language-based environments: we use the *knowledge* about the data already existing in the spreadsheet to guide end-users to introduce correct data. The knowledge about the spreadsheet under consideration is based on the *meaning* of its data that we infer by using data mining and database normalization techniques.

Data mining techniques are used to infer *functional dependencies* from the spreadsheet data. These functional dependencies define how some spreadsheet columns determine the values of other columns. Database normalization techniques, namely the use of normal forms [5], are used to eliminate redundant functional dependencies induced by the data mining techniques, and to define a relational database model. This model may include several tables, which may contain primary keys: a column, or a set of columns, that determine the value of other columns. Knowing the relational database model induced by the spreadsheet, we con-

	A	B	C	D	E	F	G	H	I	J	K
1	**clientNo**	**propNo**	**cName**	**pAddress**	**rentStart**	**rentFinish**	**days**	**rent**	**total**	**ownerNo**	**oName**
2	cr76	pg4	john	6 Lawrence	01/07/00	08/31/01	602	50	30100	co40	tina
3	cr76	pg16	john	5 Novar Dr.	09/01/01	09/01/02	365	70	25550	co93	tony
4	cr56	pg4	aline	6 Lawrence	09/02/99	06/10/00	282	50	14100	co40	tina
5	cr56	pg36	aline	2 Manor Rd	10/10/00	12/01/01	417	60	25020	co93	tony
6	cr56	pg16	aline	5 Novar Dr.	11/01/02	08/10/03	282	70	19740	co93	tony

Figure 2. A spreadsheet representing a property renting system.

struct a new spreadsheet environment that not only contains the data of the original one, but that also includes advanced features which provide information to the end-user about correct data that can be introduced.

We consider three types of advanced features: *auto-completion of column values*, where by writing or selecting from a *combo box* the value of an existing primary key, the values of the columns that depend on that primary key are automatically filled in with the correspondent values. The *non-editable columns* feature prevents the end-user of editing columns that depend on a primary key value. Note that, such columns are automatically filled in by selecting a primary key. By using the auto-completion feature the spreadsheet system guarantees that the end-user does not introduce data that violates the relational model inferred. The *safe deletion of rows* feature warns if by deleting a selected row some critical information is lost. Like in modern programming language environment, the refactored spreadsheet system also offers the possibility of using traditional editing, that is, the introduction of data by editing each of the columns. When using traditional editing the end-user is able to introduce data that may violate the relational database model inferred from the previous spreadsheet data. The spreadsheet environment includes a mechanism to recalculate the relational database model after traditional editing. This new relational model is used to guide the end-user in future non-standard editing of the spreadsheet.

This paper is organized as follows: Section 2 presents a motivating example used throughout the paper. Section 3 discusses how to embed modern programming environments in spreadsheet. Section 4 discusses related work and Section 5 concludes the paper.

2. Motivating Example

In order to present our approach we shall consider the following well-known example taken from [6] and modeled in a spreadsheet as shown in Figure 2.

This sheet stores information about a housing renting system, gathering information about clients, owners and properties. It also stores prices and dates of renting. The name of each column gives a clear idea of the information it represents. We extend this example with two additional columns, named $days$ (that computes the number of days of renting by subtracting the column $rentFinish$ to $rentStart$) and $total$ (that multiplies the number of days of renting by the rent per day value, $rent$). As usually in spreadsheets, these columns are expressed by formulas.

This spreadsheet defines a valid model to represent the information of the renting system, however, it contains redundant information. For example, the displayed data specifies the house renting of two clients (and owners) only, but their names are included 5 times. This kind of redundancy makes the maintenance and update of the spreadsheet complex and error-prone. A mistake is easily made, for example by mistyping a name and thus corrupting the data.

Two common problems exist in redundant data: *Update Anomalies* and *Deletion Anomalies* [16]. The former problem occurs when we change information in one place but leave the same information unchanged in the other places. The problem also occurs if the update in not performed exactly in the same way. In our example, this may happen if we change the rent per day of property number `pg4` from `50` to `60`. The latter problem happens when we delete some tuple and we lose other information as a side effect. For example, if we delete row 5 in the original spreadsheet all the information concerning property `pg36` is eliminated.

The database community has developed techniques, such as data normalization, to eliminate such redundancy and improve data integrity [8, 16]. Database normalization is based on the detection and exploitation of functional dependencies inherent in the data [4]. Can we leverage these database techniques for spreadsheets systems so that the system eliminates the update and deletion anomalies by guiding the end-user introducing correct data? Based on the data in our example spreadsheet, we would like to discover the following functional dependencies which represent the four entities involved in our house renting system: *clients*, *owners*, *properties* and the *renting* action itself.

A functional dependency $A \rightharpoonup B$ means that if we have two equal inhabitants of A, then the corresponding inhabitants of B are also equal. For instance, the client number functionally determines his/her name, since no two clients have the same number. Spreadsheet formulas can induce functional dependencies too [7].

Using these functional dependencies it is possible to construct a relational database schema. Each functional dependency is translated into a table where the attributes are

	A	B	C	D	E	F	G	H	I	J	K	L
1	**clientNo**	**propNo**	**cName**	**pAddress**	**rentStart**	**rentFinish**	**days**	**rent**	**total**	**ownerNo**	**oName**	
2	cr76	pg4	john	6 Lawrence	01/07/00	08/31/01	602	50	30100	co40	tina	Delete
3	cr76	pg16	john	5 Novar Dr.	09/01/01	09/01/02	365	70	25550	co93	tony	Delete
4	cr56	pg4	aline	6 Lawrence	09/02/99	06/10/00	282	50	14100	co40	tina	Delete
5	cr56	pg36	aline	2 Manor Rd	10/10/00	12/01/01	417	60	25020	co93	tony	Delete
6	cr56	pg16	aline	5 Novar Dr.	11/01/02	08/10/03	282	70	19740	co93	tony	Delete
7	cr56	pg36	aline	2 Manor Rd			0	70	0	co93	tony	Delete

Figure 3. A spreadsheet with auto completion based on relational tables.

the ones participating in the functional dependency and the primary key is the left hand side of the functional dependency. In some cases, foreign keys can be inferred from the schema. The relational database schema can be normalized in order to eliminate redundancy. A possible normalized relational database schema created to the house renting spreadsheet is presented below.

$\underline{clientNo}, cName$
$\underline{ownerNo}, oName$
$\underline{propNo}, pAddress, rent, ownerNo$
$\underline{clientNo, propNo}, rentStart, rentFinish, total, days$

Having defined a relational database schema we would like to construct a spreadsheet environment that respects that schema. For example, this spreadsheet would not allow the user to introduce two different properties with the same code number *propNo*. Instead, we would like that the spreadsheet offers to the user a list of possible properties, such that he can choose the value to fill in the cell. Figure 4 shows a possible spreadsheet environment where possible properties can be chosen from a *combo box*.

	A	B	C	D
1	**clientNo**	**propNo**	**cName**	**pAddress**
2	cr76	pg4	john	6 Lawrence
3	cr76	pg16	john	5 Novar Dr.
4	cr56	pg4	aline	6 Lawrence
5	cr56	pg36	aline	2 Manor Rd
6	cr56	pg16	aline	5 Novar Dr.
7	cr56	pg36	aline	2 Manor Rd
8		pg4		
9		pg36		
		pg16		

Figure 4. Selecting possible values of columns using a combo box.

Using the relational data base schema we would like that our house renting spreadsheet offers the following features: auto-completion of column values; no editable columns; safe deletion of rows; traditional editing and re-calculation of the relational database model.

In this section we have described an instance of our techniques. In fact, the spreadsheet programming environment shown in the Figures 3 and 4 was automatically produced from the original spreadsheet displayed in Figure 2. In the following sections we will present in detail the technique to perform such an automatic spreadsheet refactoring.

3. Spreadsheet Programming Environment

This section presents techniques to refactor spreadsheets into powerful spreadsheet programming environments as described in Section 2.

The functional dependencies and the relational database model induced by the spreadsheet data are the building block techniques for such refactoring. The (bidirectional) mapping between spreadsheets and a relational database model is presented in [7] and it was used to refactor/normalize several spreadsheets included in the EUSES spreadsheet corpus [11]. In this paper, we describe how to embed the derived relational database model in the spreadsheet in order to define a powerful spreadsheet system. The embedding is modeled in the spreadsheet itself by standard formulas and visual objects: additional formulas are included in the spreadsheet to guide the user introducing correct data.

Before we present how this embedding is defined for each of the advanced features we consider, let us first define a spreadsheet. A spreadsheet can be seen as a partial function $S : A \rightarrow V$ mapping addresses to spreadsheet values. Elements of S are called *cells* and are represented as (a, v). A cell address is taken from the set $A = \mathbb{N} \times \mathbb{N}$. A value $v \in V$ can be an input plain value $c \in C$ like a string or a number, references to other cells using addresses $a \in A$, or formulas $f \in F$ that can be applied to one or more values: $v \in V ::= c \mid a \mid f(v, \ldots, v)$.

Auto Completion: This feature is implemented by embedding each of the relational tables in the spreadsheet. This embedding is implemented by a spreadsheet formula and a combo box visual object. The combo box displays the possible values of one column, associated to the primary key of the table, while the formula is used to automatically fill in the values of the columns determined by the primary key.

Let us consider the table schema $\underline{ownerNo}, oName$ from our running example. In the spreadsheet, $ownerNo$ is in column J while $oName$ is in column K. This table is embedded in the spreadsheet introducing a combo box containing the existing values in the column J (as displayed in

Figure 4). Knowing the value in the column J we can automatically define the value in column K. To achieve this, we introduce in row 7 of column K the following formula

$$S\ (K, 7) = \textbf{if}\ (\textbf{isna}\ (\textbf{vlookup}\ (J7, J2 : K6, 2, 0)),\ \text{""}, \textbf{vlookup}\ (J7, J2 : K6, 2, 0))$$

This formula uses a (library) function **isna** to test if there is a value introduced in column J. In case that a value exists, it searches (using the function **vlookup**) the corresponding value in the column K and references it. If there is no selected value, it outputs the empty string. The combination of the combo box and this formula guides the user to introduce correct data as illustrated in Figure 3.

Note that this formula considers tables with primary keys consisting of multiple attributes (columns). Note also that the formula is defined in each column associated to non-key attribute values.

Foreign keys[1] pointing to primary keys become very helpful in this setting. Consider, for example, two relational table schemas $\underline{A}$, B and $\underline{B}$, C where B is a foreign key from the second table to the first one. When we perform auto completion in column A, then both B and C are automatically filled in. This was the case presented in Figure 3.

No Editing: To prevent the introduction of incorrect data and, thus, producing update anomalies, we protect some columns from edition. Thus, the relational table $\underline{s}, ..., \underline{t}, u, ..., v$ induces the non-edition of columns $u, ..., v$.

That is to say that columns that form a table but are not part of its primary key are not editable.

In case the end-user needs to change the value of such protected columns, then we provide traditional editing as described in one of the next features.

Safe Deletion: Another usual problem with non-normalized data is the deletion problem. Suppose in our running example that row 5 is deleted. All the information concerning property `pg36` will be lost (although probably the end-user only wanted to delete the transaction itself).

To correctly delete rows in the spreadsheet, a button is added to each row in the spreadsheet, as follows: For each relational table $\underline{s}, ..., \underline{t}, u, ..., v$ each button checks, on its corresponding row, the columns that are part of the primary key, $s, ..., t$. For each key column, it verifies if the value that is being removed is the last one proceeding, as follows: Let $c \in \{s, ..., t\}$, r be the button row, $r1$ be the first row of column c with data and rn be the last row of column c with data. The test is defined using the following formula:

$$\textbf{if}\ (\textbf{isLast}\ ((c, r), (c, r1) : (c, rn)),\ \textbf{showMessage}, \textbf{deleteRow}\ (r))$$

If the value is the last one, the spreadsheet warns the user, **showMessage**. In the case the value is not the last one, the row will be removed, **deleteRow** (r).

[1] A *Foreign Key* is a set of attributes within one relation that matches the primary key of some relation.

For example, in column $propNo$ of our running example, the row 5 contains the only data about the house with code `pg36`. If the user tries to delete such row, the warning message will be triggered.

Traditional Editing: Programming language environments provide both advanced editing mechanisms and traditional ones (*i.e.*, text editing). In a similar way, a spreadsheet environment should allow the user to perform traditional spreadsheet editing too. Thus, the environment should provide a mechanism to enable/disable the advanced features described in this section. When advanced features are disabled, then the end-user would be able to introduce data that violates the (previously) inferred relational model. However, when the end user returns to advance editing, then the spreadsheet should infer a new relational model that will be used in future (advanced) interactions.

HaExcel Add-in: We have implemented the `FUN` and the `synthesize` algorithms, and the embedding of the relational model in the HASKELL [14] programming language. We have also defined the bi-directional mapping from spreadsheet to relational databases in the same framework named HaExcel. Finally, we have extended this framework to produce the visual objects and formulas to model the relational tables in the spreadsheet. An Excel add-in has been also constructed so that the end-user can use spreadsheets in this popular system and at the same time our advanced features.

4. Related Work

Our work is strongly related to a series of techniques by Abraham *et al.*. Firstly, they designed and implemented an algorithm that uses the labels within a spreadsheet for *unit checking* [1, 10]. By typing the cells in a spreadsheet with unit information and tracking them through references and formulas, various types of users errors can be caught. We have adopted the view of Abraham *et al.* of a spreadsheet as a collection of tables and we have reused their algorithm for identifying the spatial boundaries of these tables. Rather than exploiting the labels in the spreadsheet to reconstruct implicit user intentions, we exploit redundancies in data elements. Consequently, the errors caught by our approach are of a different kind.

Secondly, Abraham *et al.* developed a type system and corresponding inference algorithm that assigns types to values, cells, formulas, and entire spreadsheets [3]. The type system can be used to catch errors in existing spreadsheets or to infer models for spreadsheets that can help to prevent future errors. These models are condensed representations of areas of repeating types. These models are not relational database models, as in our approach, but similar to collection types (e.g. lists) in regular programming.

Thirdly, Abraham *et al.* developed a tool for generating spreadsheets from spreadsheet specifications (models) [9]. Generated spreadsheets are guaranteed to be free of reference, range, or type errors. This implies a significant departure of normal spreadsheet usage, where domain experts create spreadsheet specifications and others use these to generate spreadsheets. A system has also been developed to extract specifications from existing spreadsheets [2]. Our approach does not require such a paradigm shift. The inferred model is present in the background only, of a familiar spreadsheet environment enhanced with features for completion, protection, and insertion of data.

5. Conclusions

We have demonstrated how implicit structural properties of spreadsheet data can be exploited to offer edit assistance to spreadsheet users. To discover these properties, we have made use of our previously developed approach for mining functional dependencies from spreadsheets and subsequent synthesis of a relational database schema [7]. On this basis, we have made the following contributions:

- Derivation of formulas and visual elements that capture the knowledge encoded in the reconstructed relational database schema.
- Embedding of these formulas and visual elements into the original spreadsheet in the form of features for auto-completion, guarded deletion, and controlled insertion.
- Integration of the algorithms for reconstruction of a schema, for derivation of corresponding formulas and visual elements, and for their embedding into a *add-in* for spreadsheet environments.

A spreadsheet environment enhanced with our *add-in* compensates to a significant extent for the lack of the structured programming concepts in spreadsheets. In particular, it assists users to prevent common update and deletion anomalies during edit actions .

There are several extensions of our work that we would like to explore. The algorithms running in the background need to re-calculate the relational schema and the ensuing formulas and visual elements every time new data is inserted. For larger spreadsheets, this recalculation may incur waiting time for the user. Several optimizations of our algorithms can be attempted to eliminate such waiting times.

Our approach could be integrated with complementary approaches to cover a wider range of possible user errors. In particular, the work of Abraham *et al.* [2, 3] for preventing range, reference, and type errors could be combined with our work for preventing data loss and inconsistency.

Acknowledgments The authors would like to thank Martin Erwig and his team for providing us the code from the UCheck project.

References

[1] R. Abraham and M. Erwig. Header and unit inference for spreadsheets through spatial analyses. *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*, pages 165–172, Sept. 2004.

[2] R. Abraham and M. Erwig. Inferring templates from spreadsheets. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 182–191, New York, NY, USA, 2006. ACM.

[3] R. Abraham and M. Erwig. Type inference for spreadsheets. In A. Bossi and M. J. Maher, editors, *Proc. of the 8th Int. ACM SIGPLAN Conf. on Principles and Practice of Declarative Programming, Venice, Italy*, pages 73–84. ACM, 2006.

[4] C. Beeri, R. Fagin, and J. Howard. A complete axiomatization for functional and multivalued dependencies in database relations. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 47–61, 1977.

[5] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[6] T. Connolly and C. Begg. *Database Systems, A Practical Approach to Design, Implementation, and Management*. Addison-Wesley, 3 edition, 2002.

[7] J. Cunha, J. Saraiva, and J. Visser. From spreadsheets to relational databases and back. In *Proc. of the 2009 ACM SIGPLAN workshop on partial evaluation and program manipulation*, pages 179–188, New York, NY, USA, 2008. ACM.

[8] C. J. Date. *An Introduction to Database Systems*. Addison-Wesley, 1995.

[9] M. Erwig, R. Abraham, S. Kollmansberger, and I. Cooperstein. Gencel: a program generator for correct spreadsheets. *J. Funct. Program*, 16(3):293–325, 2006.

[10] M. Erwig and M. M. Burnett. Adding apples and oranges. *4th Int. Symp. on Practical Aspects of Declarative Languages*, pages 173–191, 2002.

[11] M. Fisher II and G. Rothermel. The EUSES Spreadsheet Corpus:A shared resource for supporting experimentation with spreadsheet dependability mechanism. In 1^{st} *Workshop on End-User Software Engineering*, pages 47–51, May 2005.

[12] S. Holzner. *Eclipse*. O'Reilly, May 2004.

[13] M. Kuiper and J. Saraiva. Lrc - A Generator for Incremental Language-Oriented Tools. In K. Koskimies, editor, *7th International Conference on Compiler Construction*, volume 1383 of *LNCS*, pages 298–301. Springer-Verlag, April 1998.

[14] S. Peyton Jones. Haskell 98: Language and libraries. *J. Funct. Program.*, 13(1):1–255, 2003.

[15] T. Reps and T. Teitelbaum. The synthesizer generator. *SIGSOFT Softw. Eng. Notes*, 9(3):42–48, 1984.

[16] J. D. Ullman and J. Widom. *A First Course in Database Systems*. Prentice Hall, 1997.

[17] M. van den Brand, P. Klint, and P. Olivier. Compilation and Memory Management for ASF+SDF. In Stefan Jähnichen, editor, *8th International Conference on Compiler Construction*, volume 1575 of *LNCS*, pages 198–213, Mar. 1999.