



MODEL-BASED PROGRAMMING ENVIRONMENTS FOR SPREADSHEETS

Jácome Cunha
Departamento de Informática
jacome@di.uminho.pt

João Saraiva
Departamento de Informática
jas@di.uminho.pt

Joost Visser
Software Improvement Group
j.visser@sig.eu

ABSTRACT

Although spreadsheets can be seen as a flexible programming environment, they lack some of the concepts of declarative programming languages, such as structured data types. This can lead the user to edit the spreadsheet in a wrong way and perhaps cause corrupt or redundant data.

We devised a method for extraction of a relational model from a spreadsheet and the subsequent embedding of the model back into the spreadsheet to create a model-based spreadsheet programming environment. The extraction algorithm is specific for spreadsheets since it considers particularities such as layout and column arrangement. The extracted model is used to generate formulas and visual elements that are embedded in the spreadsheet helping users to edit correct data.

We present preliminary experimental results from applying our approach to a sample of spreadsheets from the EUSES Spreadsheet Corpus.

INTRODUCTION

Developments in programming languages are changing the way in which we construct programs: naive text editors are now replaced by powerful programming language environments which are specialized for the programming language under consideration and which help the user throughout the editing process. Helpful features like highlighting keywords of the language or maintaining a beautified indentation of the program being edited are now provided by several text editors. Recent advances in programming languages extend such naive editors to powerful language-based environments (Kuiper and Saraiva 1998; Reps and Teitelbaum 1984; van den Brand et al. 1999; Holzner 2004). 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 and more reliable programs.

Moreover, advances in declarative programming languages are providing programmers with powerful mechanisms to structure and develop their programs. Indeed, the development of powerful type systems, modular systems, abstract models, support for generic programming and testing, lazy evaluation engines, incremental models of computation, etc. are changing the way we write our programs. However, spreadsheets when viewed as a declarative programming language does not support any of those developments! Indeed, with the exception of the pioneer work of Martin Erwig (Erwig and Burnett 2002; Abraham and Erwig 2004; Erwig et al. 2005), little work has been done on the foundations of spreadsheets by the declarative programming language community.

In this paper, we propose a declarative approach 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.

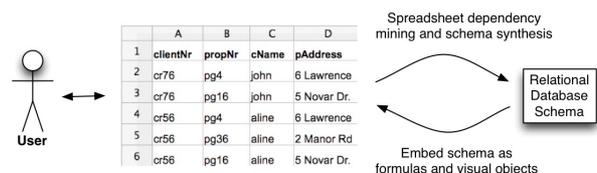


Figure 1: Edit assistance is added to a spreadsheet based on a relational database schema.

A background process adds formulas and visual objects 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 in introducing correct data. The knowledge about the spreadsheet under consideration is based on the meaning of its data that we infer using data mining and database normalization techniques. Data mining techniques specific to spreadsheets are used to infer functional dependencies from the spreadsheet data. These functional dependencies define how certain spreadsheet columns determine the values of other columns. Database normalization techniques, namely the use of normal



forms (Codd 1970), 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 data, we construct a new spreadsheet

MOTIVATING EXAMPLE

In order to present our approach we shall consider the following well-known example taken from (Connolly and Begg 2002) and modeled in a spreadsheet as shown in Figure 2.

This spreadsheet contains information related to a housing renting system. It gathers information about

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

Figure 2: A spreadsheet representing a property renting system.

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, non-editable columns and safe deletion of rows. Auto-completion means that when the user inserts a primary key (by typing or selecting from a combo box), the columns that are functionally dependent on this primary key will be filled with values automatically. The non-editable column feature prevents the end-user from 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 information not represented elsewhere is lost. Like in a modern programming language environment, the refactored spreadsheet system also offers the possibility of using traditional editing, that is, the introduction of data by editing any 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 assisted editing of the spreadsheet.

Our techniques not only work for database-like spreadsheets, like the example we will use throughout the paper, but they work also for realistic spreadsheets defined in other contexts (for example, inventory, grades or modeling). In this paper we present our first experimental results obtained by considering a large set of spreadsheets included in the EUSES Spreadsheet Corpus (Fisher II and Rothermel 2005).

clients, owners, properties, prices and renting periods. The name of each column gives a clear idea of the information it represents. We extend this example with three additional columns, named days (that computes the total number of renting days by subtracting the column rentStart to rentFinish), total (that multiplies the number of renting days by the rent per day value, rent) and country (that represents the property's country). The columns days and rent are expressed by formulas.

This spreadsheet defines a valid model to represent the information of the renting system. However, it contains redundant information: the displayed data specifies the house renting of two clients (and owners) only, but their names are included five times, for example. 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, thus corrupting the data on the spreadsheet. Two common problems occur as a consequence of redundant data: update anomalies and deletion anomalies (Ullman and Widom 1997). 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 is not performed exactly in the same way. In our example, this happens if we change the rent of property number pg4 from 50 to 60 only one row and leave the others unchanged, for example. The latter problem occurs when we delete some data and lose other information as a side effect. For example, if we delete row 5 in the our example all the information concerning property pg36 is lost. The database community has developed techniques, such as data normalization, to eliminate such redundancy and improve data integrity (Ullman and Widom 1997; Date 1995). Database normalization is based on the detection and exploitation of functional dependencies inherent in the data (Beer et al. 1977). Can we leverage these database techniques for spreadsheets systems so that the system eliminates the update and deletion anomalies by guiding the end-user to introduce correct data? Based on the data contained in our example spreadsheet, we



would like to discover the following functional dependencies which represent the four entities involved in our house renting system: countries, clients, owners and properties.

properties, such that he can choose the value to fill in the cell. Figure 3 shows a possible spreadsheet environment where possible properties can be chosen from a combo box. Columns with underlined labels and

	A	B	C	D	E	F	G	H	I	J	K	L	M
	<u>clientNr</u>	<u>propNr</u>	<u>cName</u>	<u>pAddress</u>	<u>country</u>	<u>rentStart</u>	<u>rentFinish</u>	<u>days</u>	<u>rent</u>	<u>total</u>	<u>ownerNr</u>	<u>oName</u>	
1													
2	cr76	pg4	john	6 Lawrence	UK	01/07/00	08/31/01	602	50	30100	co40	tina	Delete
3	cr76	pg16	john	5 Novar Dr.	UK	09/01/01	09/01/02	365	70	25550	co93	tony	Delete
4	cr56	pg4	aline	6 Lawrence	UK	09/02/99	06/10/00	282	50	14100	co40	tina	Delete
5	cr56	pg36	aline	2 Manor Rd	UK	10/10/00	12/01/01	417	70	25020	co93	tony	Delete
6	cr56	pg16	aline	5 Novar Dr.	UK	11/01/02	08/10/03	282	70	19740	co93	tony	Delete
7													
8													
9													
10													

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

$country \rightarrow \{ \}$
 $clientNr \rightarrow cName$
 $ownerNr \rightarrow oName$
 $propNr \rightarrow pAddress, rent, ownerNr$

A functional dependency $A \rightarrow B$ 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. The right hand side of a functional dependency can be an empty set. This occurs, for example, in the country functional dependency. Note that there are several columns (labeled rentStart, rentFinish, days and total) that are not included in any functional dependency. This happens because their data do not define any functional dependency.

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 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 data redundancy. A possible normalized relational database schema created for the house renting spreadsheet is presented below.

country
clientNr, cName
ownerNr, oName
propNr, pAddress, rent, ownerNr

This database schema defines a table for each of the entities described before. Having defined a relational database schema we would like to construct a spreadsheet environment that respects that relational model, as shown in Figure 3.

For example, this spreadsheet would not allow the user to introduce two different properties with the same property number propNr. 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 3 shows a possible spreadsheet environment where possible properties can be chosen from a combo box. Columns with underlined labels and

in green color (e.g. propNr) correspond to primary keys. By choosing one possible value from the combo box, the other attributes/columns in such table schema are automatically filled in. The environment also allows users to select non-primary key attributes/columns values from a combo box. In this case, however, the values do not uniquely determine their primary key. They are used to filter the set of possible values only. For example, if in column I the user selects the rent of 70, then in column B there are only two properties the user may choose: pg16 and pg36. Since the primary key is now restricted, the other non-primary key columns become also restricted. This means that the environment uses the functional dependencies in both directions. The delete button performs safe deletion. Besides the combo box and the button support, the spreadsheet would also provide users with traditional editing.

Using the relational data base schema we would like to achieve that our house renting spreadsheet offers the following features: auto-completion of column values, non-editable columns, safe deletion of rows, traditional editing and recalculation 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 Figure 3 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.

FROM SPREADSHEETS TO RELATIONAL DATABASES

This section briefly explains how to extract functional dependencies from the spreadsheet data and how to construct a normalized relational database schema modeling such data. These techniques were introduced in detail in our work on defining a bidirectional mapping between spreadsheets and relational databases (Cunha et al. 2009a). Although such techniques work well for database-like spreadsheets, the algorithm that infers functional dependencies has several limitations when processing other types of spreadsheets included in



the EUSES Corpus. In this section we briefly present an extension to that algorithm that uses spreadsheet specific properties to infer more realistic functional dependencies.

Relational Databases: A relational schema R is a finite set of attributes $\{A_1, \dots, A_k\}$. Corresponding to each attribute A_i is a set D_i called the domain of A_i . These domains are arbitrary, non-empty sets, finite or countably infinite. A relation (or table) r on a relation schema R is a finite set of tuples (or rows) of the form $\{t_1, \dots, t_k\}$. For each $t \in r$, $t(A_i)$ must be in D_i . A relational database schema is a collection of relation schemas $\{R_1, \dots, R_n\}$. A Relational Database (RDB) is a collection of relations $\{r_1, \dots, r_n\}$.

Each tuple is uniquely identified by a minimum non-empty set of attributes called a Primary Key (PK). On certain occasions there may be more than one set suitable for becoming the primary key. They are designated candidate keys and only one is chosen to become primary key. A Foreign Key (FK) is a set of attributes within one relation that matches the primary key of some relation.

The normalization of a database is important to prevent data redundancy. Although there are several different normal forms, in general, a RDB is considered normalized if it respects the Third Normal Form (3NF) (Connolly and Begg 2002).

Discovering Functional Dependencies: In order to define the RDB schema, we first need to compute the functional dependencies presented in a given spreadsheet data. In (Cunha et al. 2009a) we reused the well known data mine algorithm, named FUN, to infer such dependencies. This algorithm was developed in the context of databases with the main goal of inferring all existing functional dependencies in the input data. As a result, FUN may infer a large set of functional dependencies depending on the input data. For our example, we list the functional dependencies inferred from the data using FUN:

$clientNr \rightarrow cName, country$
 $propNr \rightarrow country, pAddress, rent, ownerNr, oName$
 $cName \rightarrow clientNr, country$
 $pAddress \rightarrow propNr, country, rent, ownerNr, oName$
 $rent \rightarrow propNr, country, pAddress, ownerNr, oName$
 $ownerNr \rightarrow country, oName$
 $oName \rightarrow country, ownerNr$

Note that the data contained in the spreadsheet exhibits all those dependencies. In fact, even the non-natural dependency $rent \rightarrow propNr, country, pAddress, ownerNr, oName$ is inferred. Indeed, the functional dependencies derived by the FUN algorithm depend heavily on the quantity and quality of the data. Thus, for small samples of data, or data that exhibits too many or too few dependencies, the FUN algorithm may not produce the desired functional dependencies. Note also

that the country column occurs in most of the functional dependencies although only a single country actually appears in a column of the spreadsheet, namely UK. Such single value columns are common in spreadsheets. However, for the FUN algorithm they induce redundant fields and redundant functional dependencies.

In order to derive more realistic functional dependencies for spreadsheets we have extended the FUN algorithm so that it considers the following spreadsheet properties:

- *Single value columns:* these columns produce a single functional dependency with no right hand side ($country \rightarrow \{\}$, for example). This columns are not considered when finding other functional dependencies.
- *Semantic of labels:* we consider label names as strings and we look for the occurrence of words like code, number, nr, id, giving them more priority when considered as primary keys.
- *Column arrangement:* we give more priority to functional dependencies that respect the order of columns. For example, $clientNr \rightarrow cName$ has more priority than $cName \rightarrow clientNr$.

Moreover, to minimize the number of functional dependencies we consider the smallest subset that includes all attributes/columns in the original set computed by FUN. The result of our spreadsheet functional dependency inference algorithm is:

$country \rightarrow \{\}$
 $clientNr \rightarrow cName$
 $ownerNr \rightarrow oName$
 $propNr \rightarrow pAddress, rent, ownerNr, oName$

This set of functional dependencies is very similar to the one presented previously. The exception is the last dependency which has an extra attribute (oName).

User Defined Functional Dependencies: Although the system can work in a fully automatic way, it also has the possibility of receiving user defined functional dependencies. This functional dependencies are merged with the automatically inferred ones, and then normalized as a single set (see below the normalization paragraph).

Spreadsheet Formulas: Spreadsheets use formulas to define the values of some elements in terms of other elements. For example, in the house renting spreadsheet, the column days is computed by subtracting the column rentFinish from rentStart, and it is usually written as follows: $H3 = G3 - F3$. This formulas test that the values of G3 and F3 determine the value of H3, thus inducing the following functional dependency: $rentStart, rentFinish \rightarrow days$.

Formulas can have references to other formulas. Consider, for example, the second formula of the running example $J3 = H3 * I3$, which defines the total



rent by multiplying the total number of days by the value of the rent. Because $H3$ is defined by another formula, the values that determine $H3$ also determine $J3$. As a result, the two formulas induce the following functional dependencies: $rentStart, rentFinish \rightarrow days$ and $rentStart, rentFinish, rent \rightarrow total$.

In general, a spreadsheet formula of the following form $X0 = f(X1, \dots, Xn)$ induces the following functional dependency: $X1, \dots, Xn \rightarrow X0$. In spreadsheet systems, formulas are usually introduced by copying them through all the elements in a column, thus making the functional dependency explicit in all the elements. This may not always be the case and some elements can be defined otherwise (e.g. by using a constant value or a different formula). In both cases, all the cells referenced must be used in the antecedent of the functional dependency.

These functional dependencies are useful for the mapping of spreadsheets to databases as presented in (Cunha et al. 2009a). In this work, they are not relevant since the existing formulas are used to fill in those columns.

Normalizing Functional Dependencies: Having computed the functional dependencies, we can now normalize them. Next, we show the results produced by the synthesizer algorithm introduced by Maier in (Maier 1983).

The synthesizer algorithm receives a set of functional dependencies as argument and returns a new set of compound functional dependencies. A compound functional dependency (CFD) has the form $(X1, \dots, Xn) \rightarrow Y$, where $X1, \dots, Xn$ are all distinct subsets of a scheme R and Y is also a subset of R . A relation r satisfies the CFD $(X1, \dots, Xn) \rightarrow Y$ if it satisfies the functional dependencies $Xi \rightarrow Xj$ and $Xi \rightarrow Y$, where $1 \leq i$ and $j \leq k$. In a CFD, $(X1, \dots, Xn)$ is the left side, $X1, \dots, Xn$ are the left sets and Y is the right side.

Next, we list the compound functional dependencies computed from the functional dependencies induced by our running example.

$\{\{country\}\} \rightarrow \{\}$
 $\{\{clientNr\}\} \rightarrow \{cName\}$
 $\{\{ownerNr\}\} \rightarrow \{oName\}$
 $\{\{propNr\}\} \rightarrow \{pAddress, rent, ownerNr\}$

Computing the Relational Database Schema: Each compound functional dependency defines several candidate keys for each table. However, to fully characterize the relational database schema we need to choose the primary key from those candidates. To find such keys we use a simple algorithm: we produce all the possible tables using each candidate key as the primary key; we then use the same algorithm that is used to choose the initial functional dependencies to choose the best table. Note that before applying the synthesizer algorithm, all the functional dependencies with antecedents' attributes representing formulas should be

eliminated since a primary key must not change over time. The final result is listed below.

country
clientNr, cName
 ownerNr, oName
 propNr, pAddress, rent, ownerNr

This relational database model corresponds exactly to the one shown in Section . Note that the synthesizer algorithm removed the redundant attribute oName that occurred in the last functional dependency.

SPREADSHEET PROGRAMMING ENVIRONMENT

This section presents techniques to refactor spreadsheets into powerful spreadsheet programming environments. The functional dependencies and the relational database model induced by the data included in the original spreadsheet are the building block techniques for such a refactoring. In fact, the spreadsheet refactoring is implemented as the embedding of the relational database model in the spreadsheet. This embedding is modeled in the spreadsheet itself by standard formulas and visual objects: additional formulas are included in the spreadsheet to guide the user to introduce correct data.

Before we present how this embedding is defined, 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 $A = N \times N$. 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 or formulas $f \in F$ that can be applied to one or more values: $v \in V ::= c \mid a \mid f(v_1, \dots, v_n)$.

Auto-completion of Column Values: The columns corresponding to primary keys (with underlined labels and green combo boxes columns in Figure 3) in the relational model uniquely determine the values of the other columns in such a relation. Moreover, choosing one of the non-primary key columns (red combo boxes columns in Figure 3) the possible values of the primary key become more restricted. We want the spreadsheet environment to be able to automatically fill those columns provided the end-user defines the value of the primary key or, in the other hand, to restrict the primary key when one of the other attributes is chosen. For example, the value of the property number (propNr, column B) determines the values of the address (pAddress, column D), rent per day (rent, column I), and owner number (ownerNr, column K). Consequently, the spreadsheet environment should be able to automatically fill in the values of the columns D, I and K, given the value of column B. Since ownerNr (column K) is a primary key of another table, transitively the value of oName (column L) is also



defined. In the other hand, if a rent per day (column I is chosen, the corresponding primary column B values should be restricted to those values that can determine such a choice. We consider two mechanisms to model such an auto-completion feature.

- First, by using visual objects, that is a combo box, to choose the primary key and/or the non-primary key column values as shown in the running example.
- Second, by providing traditional editing via the keyboard that obeys to the underlying relational model. In this case the end-user may edit the primary key and the columns that it determines are automatically filled in. This traditional editing uses the (usual) column value completion provided by commercial systems: as soon as a known prefix is inserted the complete string is suggested. When the insertion is completed, the auto-completable columns become also filled.

These two editing models use the derived functional dependencies/relational schema in both directions.

Next we present the details of the embedding of the relational model in the original spreadsheet. Let us consider the table ownerNr, oName from our running example. In the spreadsheet, ownerNr is in column K and oName in column L. This table is embedded in the spreadsheet introducing a combo box containing the existing values in the column K (as displayed in Figure 3). Knowing the value in the column K we can automatically introduce the value in column L. To achieve this, we introduce the following formula in row 7 of column L: $S(L,7) = \text{if}(\text{isna}(\text{vlookup}(K7,K2:L6,2,0)), "", \text{vlookup}(K7, K2:L6, 2, 0))$. This formula uses a (library) function *isna* to test if there is a value introduced in column K. In case that value exists, it searches (with the function *vlookup*) the corresponding value in the column L and references it. If there is no selected value, it produces the empty string. The combination of the combo box and this formula guides the user to introduce correct data as illustrated in Figure 3.

We have just presented a particular case of the formula and visual object induced by a relational table. Next we present the general case.

Let *minr* be the very next row after the existing data in the spreadsheet, *maxr* the last row in the spreadsheet, and *r1* the first row with already existing data. Each relational database table *a1, ..., an, c1, ..., cm*, with *a1, ..., an, c1, ..., cm* column indexes of the spreadsheet, induces firstly, a combo box defined as follows:

$$\forall c \in \{a1, \dots, an\}, \forall r \in \{minr, \dots, maxr\}: \\ S(c,r) = \text{combobox} := \{\text{linkedcell} := (c,r); \\ \text{source cells} := (c,r1):(c,r-1)\}$$

secondly, a spreadsheet formula defined as:

$$\forall c \in \{c1, \dots, cm\}, \forall r \in \{minr, \dots, maxr\}:$$

$$S(c,r) = \\ \text{if}(\text{if}(\text{isna}(\text{vlookup}((a1,r),(a1,r1):(c,r-1),r-a1+1,0)), \\ "", \\ \text{vlookup}((a1,r),(a1,r1):(c,r-1),r-a1+1,0)) \\ == \\ \text{if}(\text{isna}(\text{vlookup}((a2,r),(a2,r1):(c,r-1),r-a2+1,0)), \\ "", \\ \text{vlookup}((a2,r),(a2,r1):(c,r-1),r-a2+1,0)) \\ == \dots == \\ \text{if}(\text{isna}(\text{vlookup}((an,r),(an,r1):(c,r-1),r-an+1,0)), \\ "", \\ \text{vlookup}((an,r),(an,r1):(c,r-1),r-an+1,0)), \\ \text{vlookup}((a1,r),(a1,r1):(c,r-1),r-a1+1,0), ""))$$

This formula must be used for each non primary key column created by our algorithm. Each if inside the main if is responsible for checking a primary key column. In the case a primary key column value is chosen, *isna(vlookup(...))*, the formula calculates the corresponding non primary key column value, *vlookup(...)*. If the values chosen by all primary key columns are equal, then that value is used in the non primary key column.

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.

The example table analyzed before is an instance of this general one. In the table ownerNr,oName, ownerNr is *a1*, oName is *c1*, *c* is *L*, *r1* is 2, *minr* is 7. The value of *maxr* is always the last row supported by the spreadsheet system.

Foreign keys pointing to primary keys become very helpful in this setting. For example, if we have the relational tables A, B and B, C where B is a foreign key from the second table to the first one, then when we perform auto-completion in column A, both B and C are automatically filled in. This was the case presented in Figure 3. The combo box visual object is implemented as a script defined in the macro's language used by the spreadsheet systems1.

Non-Editable Columns: To prevent wrong introduction of data and, thus, producing update anomalies, we protect some columns from edition. The relational table *a1, ..., an, c1, ..., cm* induces the non-edition of columns *a1, ..., an, c1, ..., cm*. That is to say that all columns that form a table become non-editable. For example, column L is part of the owner table but it is not part of its primary key. In the case where the end-user needs to change the value of such protected columns, we provide traditional editing as described in Subsection .

Safe Deletion of Rows: Another usual problem with non-normalized data is the deletion problem. Suppose in our running example that row 5 is deleted. All the knowledge about the pg36 property is lost, although the end-user would probably want to delete the renting



transaction only. To correctly delete rows in the spreadsheet, a button is added to each row in the spreadsheet as follows: for each relational table a_1, \dots, a_n , c_1, \dots, c_m each button checks, on its corresponding row, the columns that are part of the primary key, a_1, \dots, a_n . For each primary key column, it verifies if the value that is being removed is the last one. Let $c \in \{a_1, \dots, a_n\}$, let r be the button row, r_1 be the first row of column c with data and r_n be the last row of column c with data. The test is defined as follows: *if (isLast((c,r), (c,r1):(c,rn)), showMessage, deleteRow(r))*. If the value is the last one, the spreadsheet warns the user (showMessage). If the user presses the OK button, the spreadsheet will remove the row. In the other case, Cancel, no action will be performed. In the case the value is not the last one, the row will simply be removed, deleteRow (r).

For example, in column propNr of our running example, the row 5 contains the last data about the house with code pg36. If the user tries to delete this row, the warning will be triggered.

Traditional Editing: Advanced 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, the end-user is able to introduce data that violates the (previously) inferred relational model. However, when the end-user returns to advanced editing, the spreadsheet infers a new relational model that will be used in future (advanced) interactions.

Recalculation of the Relational Database Model: because standard editing allows the end-user to introduce data violating the underlying relational model, we would like that the spreadsheet environment may enable/disable the advanced features described in this section. When advanced features are disabled, the end user would be able to introduce data that violates the (previously) inferred relational model. However, when the end-user returns to advanced editing, then the spreadsheet should infer a new relational model that will be used in future (advanced) interactions.

HaExcel Addon: We have implemented the FUN algorithm, the extensions described in this paper, the synthesize algorithm, and the embedding of the relational model in the HASKELL programming language (Peyton Jones 2003). We have also defined the mapping from spreadsheet to relational databases in the same framework named HaExcel (Cunha et al. 2009a). Finally, we have extended this framework to produce the visual objects and formulas to model the relational tables in the spreadsheet. An OpenOffice addon has also been constructed so that the end-user can use

spreadsheets in this popular system and at the same time our advanced features. This can be seen in the left part of Figure 3. There are three buttons that allow HaExcel with auto-completion with combo boxes (first button), HaExcel with auto-completion without combo boxes (second button) and disable HaExcel (third button). This addon can be found in the first author's web page.

PRELIMINARY EXPERIMENTAL RESULTS

In order to evaluate the applicability of our approach, we have performed a preliminary experiment on the EUSES Corpus (Fisher II and Rothermel 2005). This corpus was conceived as a shared resource to support research on technologies for improving the dependability of spreadsheet programming. It contains more than 4500 spreadsheets gathered from different sources and developed for different domains. These spreadsheets are assigned to eleven different categories. Among the spreadsheets in the corpus, about 4.4% contain macros, about 2.3% contain charts, and about 56% do not have formulas being only used to store data. In our preliminary experiment we have selected the first ten spreadsheets from each of the eleven categories of the corpus. We then applied our tool to each spreadsheet, with different results (see also Table 1):

Table 1: The number of spreadsheets per category present in the EUSES corpus.

category	corpus	selected	processed
cs101	8	8	5
database	680	10	6
filby	44	10	2
financial	752	10	2
forms3	26	10	9
grades	632	10	5
homework	656	10	7
inventory	709	10	5
jackson	12	10	0
modeling	716	10	9
personal	5	5	0
total	4240	103	50

- Some spreadsheets were parsed, but no tables could be recognized in them, i.e., their users did not adhere to any of the supported layout conventions. The layout conventions we support are the ones presented in the UCheck project (Abraham and Erwig 2007). This was the case for about one third of the spreadsheets in our selection.

- The other spreadsheets were parsed, tables were recognized, and edit assistance was generated for them. We will focus on the last groups in the upcoming sections.



Processed Spreadsheets: The results of processing our sample of spreadsheets from the EUSES corpus are summarized in Table 2. The rows of the table are grouped by category as documented in the corpus. The first three columns contain size metrics on the spreadsheets. They indicate how many tables were

to discard redundant functional dependencies. The last three columns are metrics on the generated edit assistance. In some cases, no edit assistance was generated, indicated by zeros in these columns. This situation occurs when no (non-trivial) dependencies are induced from the recognized tables. In the other cases, the three columns respectively indicate:

Table 2: Preliminary results of processing the selected spreadsheets.

File name	Recognized tables	Columns	Cells	Functional dependencies	Col. w/ safe insert & delete	Auto-complete columns	Uneditable columns
cs101							
Act4.023-capen	5	24	402	0	0	0	0
act3_23_bartholomew	6	21	84	1	8	1	9
act4.023_bartholomew	6	23	365	0	0	0	0
meyer_Q1	2	8	74	0	0	0	0
posey_Q1	5	23	72	0	8	0	8
database							
%5CDepartmental%20Fol#A861A	2	4	3463	0	0	0	0
00061r0P802-15.TG2-Un#A7F69	23	55	491	0	18	4	21
00061r5P802-15.TG2-Un#A7F6C	30	83	600	25	21	5	26
0104TexasNutrientdb	5	7	77	1	1	1	2
01BTS_framework	52	80	305	4	23	2	25
03-1-report-annex-5	20	150	1599	12	15	8	22
filby							
BROWN	5	14	9047	2	3	1	4
CHOFAS	6	48	4288	3	3	1	4
financial							
03PFMJOURnalBOOKSFina...	15	65	242	0	7	0	7
10-formc	12	20	53	8	5	4	9
forms3							
ELECLAB3_reichwja.xl97	1	4	44	0	0	0	0
burnett-clockAsPieChart	3	8	14	0	1	0	1
chen-heapSortTimes	1	2	24	0	0	0	0
chen-insertSortTimes	1	2	22	0	0	0	0
chen-lcsTimes	1	2	22	0	0	0	0
chen-quickSortTimes	1	2	24	0	0	0	0
cs515_npeg_chart_reichwja.xl97	7	9	93	0	0	0	0
cs515_polynomials_reichwja.xl97	6	12	105	0	0	0	0
cs515_runtimeData_reichwja.XL97	2	6	45	0	0	0	0
grades							
0304deptcal	11	41	383	19	18	17	28
03_04ballots1	4	20	96	6	4	0	4
030902	5	20	110	0	0	0	0
031001	5	20	110	0	0	0	0
031501	5	15	51	31	3	1	4
homework							
01_Intro_Chapter_Home#A9171	6	15	2115	0	1	0	1
01readsdis	4	16	953	5	4	3	6
02%20fbb%20medshor	1	7	51	0	0	0	0
022timeline4dev	28	28	28	0	0	0	0
026timeline4dev	28	28	30	0	2	0	2
03_Stochastic_Systems#A9172	4	6	48	0	2	0	2
04-05_proviso_list	79	232	2992	0	25	0	25
inventory							
02MDE_framework	50	83	207	10	31	1	32
02f20assignment%234soln	37	72	246	7	20	1	21
03-1-report-annex-2	5	31	111	10	5	5	8
03singapore_elec_gene#A8236	9	45	153	3	5	2	7
0038	10	22	370	0	0	0	0
modeling							
%7B94402d63-cdd8-4cc3#A8841	1	3	561	0	0	0	0
%EC%86%90%ED%97%8C...	1	10	270	13	7	5	9
%EC%9D%98%EB%8C%80...	1	7	1442	4	4	5	6
%EC%A1%B0%EC%9B%90...	2	17	534	18	13	5	15
%ED%99%98%EA%B2%BD...	3	7	289	2	1	2	3
0.10900,0-0-45-109057-0,00	4	14	6558	9	9	2	10
00-323r2	24	55	269	31	9	6	15
00000r6xP802-15_Docum#A7D9E	3	13	3528	10	9	3	11
003.4	25	50	2090	0	0	0	0

recognized, how many columns are present in these tables, and how many cells. For example, the first spreadsheet in the financial category contains 15 tables with a total of 65 columns and 242 cells. The fourth column shows how many functional dependencies were extracted from the recognized tables by our algorithm. These are the non-trivial functional dependencies that remain after we use our extension to the FUN algorithm

- For how many columns a combo box has been generated for controlled insertion. The same columns are also enhanced with the safe deletion of rows feature.
- For how many columns the auto-completion of column values has been activated, i.e., for how many columns the user is no longer required to insert values manually.



- How many columns are locked to prevent edit actions where information that does not appear elsewhere is deleted inadvertently.

For example, for the first spreadsheet of the inventory category, combo boxes have been generated for 31 columns, auto-completion has been activated for 1 column, and locking has been applied to 32 columns. Note that for the categories jackson and personal, no results were obtained due to absent or unrecognized layout conventions or to the size of the spreadsheets (more than 150,000 cells).

Observations: On the basis of these preliminary results for our sample of corpus spreadsheets, a number of interesting observations can be made. For some categories, edit assistance is successfully added to almost all spreadsheets (e.g. inventory and database), while for others almost none of the spreadsheets lead to results (e.g. the forms/3 category). The latter may be due to the small sizes of the spreadsheets in this category. For the financials category, we can observe that in only 2 out of 10 sample spreadsheets tables were recognized, but edit assistance was successfully generated for both of these. The percentage of columns for which edit assistance was generated varies. The highest percentage was obtained for the second spreadsheet of the modeling category, with 9 out of 10 columns (90 %). A good result is also obtained for the first spreadsheet of the grades category with 28 out of 41 columns (68.3 %). On the other hand, the 5th of the homework category gets edit assistance for only 2 out of 28 columns (7.1 %). The number of columns with combo boxes often outnumbers the columns with auto-completion. This may be due to the fact that many of the functional dependencies are small, with many having only one column in the antecedent and none in consequent.

Evaluation: Our preliminary experiment justifies two preliminary conclusions. Firstly, the tool is able to successfully add edit assistance to a series of non-trivial spreadsheets. A more thorough study of these and other cases can now be started to identify technical improvements that can be made to the algorithms for table recognition and functional dependency extraction. Secondly, in the enhanced spreadsheets a large number of columns are generally affected by the generated edit assistance, which indicates that the user experience can be impacted in a significant manner. Thus, a validation experiment can be started to evaluate how users experience the additional assistance and to which extent their productivity and effectiveness can be improved.

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 (Erwig and Burnett 2002; Abraham and Erwig 2004). 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, operations, cells, formulas, and entire spreadsheets (Abraham and Erwig 2006b). The type system can be used to catch errors in existing spreadsheets or to infer models for spreadsheets that can help to prevent errors. These models are condensed representations of areas of repeating types, and so, they are not relational database models, as in our approach, but similar to collection types in regular programming.

Thirdly, Abraham et al. developed a tool for generating spreadsheets from spreadsheet specifications (models) (Erwig et al. 2006). 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 (Abraham and Erwig 2006a). 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.

In previous work we presented techniques and tools to transform spreadsheets into relational databases and back (Cunha et al. 2009a). As in the current paper, we used an algorithm for functional dependency recovery to construct a relational model, but rather than generating edit assistance, the recovered information was used to perform spreadsheet refactoring. The algorithm for extracting and filtering spreadsheets presented in the current paper is an improvement over that work. We provided a short user-centered overview of the idea of generating edit assistance for spreadsheets via extraction of functional dependencies in a previous short paper (Cunha et al. 2009b). In the current paper, we provide the technical details of the solution, including the improved algorithm for inferring and filtering functional dependencies. Also, we provide the first preliminary evaluation of the approach by



application to a sample of spreadsheets from the EUSES corpus.

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 improved approach for mining functional dependencies from spreadsheets and subsequent synthesis of a relational database. 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 an addon for spreadsheets.

A spreadsheet environment enhanced with our addon 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 editing.

There are several extensions of our work that we would like to explore. The algorithms running in the background need to recalculate 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, for example, by use of incremental evaluation. Our approach could be integrated with similar, complementary approaches to cover a wider range of possible user errors. In particular, the work of Abraham et al. (Abraham and Erwig 2006b,a) for preventing range, reference, and type errors could be combined with our work for preventing data loss and inconsistency.

We have presented some preliminary experimental results to pave the way for a more comprehensive validation experiments. In particular, we intend to set up a structured experiment for testing the impact on end-user productivity, and effectiveness.

REFERENCES

- R. Abraham and M. Erwig. Header and unit inference for spreadsheets through spatial analyses. *Visual Languages and Human Centric Computing, IEEE Symp. on*, pages 165–172, Sept. 2004.
- R. Abraham and M. Erwig. UCheck: A spreadsheet type checker for end users. *J. Vis. Lang. Comput.*, 18(1):71–95, 2007.
- Robin Abraham and Martin Erwig. Inferring templates from spreadsheets. In *ICSE '06: Proceedings of the 28th Int. Conf. on Software Engineering*, pages 182–191, New York, NY, USA, 2006a. ACM. ISBN 1-59593-375-1.
- Robin Abraham and Martin Erwig. Type inference for spreadsheets. In Annalisa Bossi and Michael J. Maher, editors, *Proc. of the 8th Int. ACM SIGPLAN Conf. on Principles and Practice of Declarative Programming*, Venice, Italy, pages 73–84. ACM, 2006b. ISBN 1-59593-388-3.
- C. Beeri, R. Fagin, and J.H. 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.
- E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.
- T. Connolly and C. Begg. *Database Systems, A Practical Approach to Design, Implementation, and Management*. Addison-Wesley, 3 edition, 2002.
- Jácóme Cunha, João Saraiva, and Joost Visser. From spreadsheets to relational databases and back. In *PEPM '09: Proc. of the 2009 ACM SIGPLAN Workshop on partial evaluation and program manipulation*, pages 179–188, New York, NY, USA, 2009a. ACM. ISBN 978-1-60558-327-3.
- Jácóme Cunha, João Saraiva, and Joost Visser. Discovery-based edit assistance for spreadsheets. In *IEEE Symposium on Visual Languages and Human-Centric Computing*, IEEE, 2009b.
- C. J. Date. *An Introduction to Database Systems*. Addison-Wesley, 1995. ISBN 0-201-82458-2.
- Martin Erwig and Margaret M. Burnett. Adding apples and oranges. *4th Int. Symp. on Practical Aspects of Declarative Languages*, pages 173–191, 2002.
- Martin Erwig, Robin Abraham, Irene Cooperstein, and Steve Kollmansberger. Automatic generation and maintenance of correct spreadsheets. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 136–145, New York, NY, USA, 2005. ACM. ISBN 1-59593-963-2.
- Martin Erwig, Robin Abraham, Steve Kollmansberger, and Irene Cooperstein. Gencil: a program generator for correct spreadsheets. *J. Funct. Program*, 16(3):293–325, 2006.
- Marc Fisher II and Gregg Rothermel. The EUSES Spreadsheet Corpus: A shared resource for supporting experimentation with spreadsheet dependability mechanism. In *1st Workshop on End-User Software Engineering*, pages 47–51, May 2005.
- Steve Holzner. *Eclipse*. O'Reilly, May 2004. ISBN 0596006411.
- Matthijs Kuiper and João Saraiva. Lrc - A Generator for Incremental Language-Oriented Tools. In Kay Koskimies, editor, *7th International Conference on Compiler Construction*, volume 1383 of



- LNCS, pages 298–301. Springer-Verlag, April 1998.
- D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1983. ISBN 0-914894-42-0. S. Peyton Jones. Haskell 98: Language and libraries. *J. Funct. Program.*, 13(1):1–255, 2003.
- A.D. Pitonyak, C.R. Pearsall, J.H. Weber, and A. Boyer. *OpenOffice.org macros explained*. Hentzenwerke Pub Inc, 2004.
- Thomas Reps and Tim Teitelbaum. The synthesizer generator. *SIGSOFT Sof. Eng. Notes*, 9(3):42–48, 1984. ISSN 0163-5948.
- J. D. Ullman and J. Widom. *A First Course in Database Systems*. Prentice Hall, 1997.
- Mark van den Brand, Paul Klint, and Pieter Olivier. Compilation and Memory Management for ASF+SDF. In Stefan Jahnichen, editor, 8th International Conference on Compiler Construction, volume 1575 of LNCS, pages 198–213, March 1999.