PrediCalc: A Logical Spreadsheet Management System

Michael Kassoff Lee-Ming Zen Ankit Garg Michael Genesereth

Logic Group, Department of Computer Science, Stanford University
{mkassoff, leezen, ankitg, genesereth}@stanford.edu

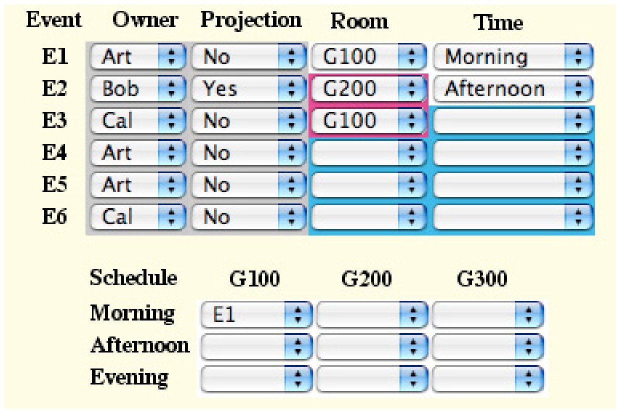
1 Introduction

Computerized spreadsheets are a great success. They are often touted in newspapers and magazine articles as the first "killer app" for personal computers. Over the years, they have proven their worth time and again. Today, they are used for managing enterprises of all sorts - from one-person projects to multi-institutional conglomerates.

The power of computerized spreadsheets derives in large part from two primary features - the automatic calculation of the values on spreadsheets and the use of mathematical formulas in specifying those calculations. The automatic calculation of values frees the user from the tedious task of doing those calculations manually. The support for mathematical formulas simplifies the task of setting up those calculations and makes spreadsheet technology accessible to a broad class of users, including those with no background in programming.

Despite their successes, computerized spreadsheet systems today have significant and unnecessary restrictions that limit their usefulness. One significant restriction is that the formulas used to specify calculations must be functions. Spreadsheet systems generally do not provide the ability to encode many-to-many relationships across cell values. Another superfluous restriction is that propagation can only occur in one direction in a traditional spreadsheet. For example, if one defines C = A + B, then one can specify values for A and B and obtain a value for C, but one cannot specify a value for C and A and obtain a value for B.

In this demonstration, we present PrediCalc, a spreadsheet system which allows for general logical constraints and omnidirectional propagation. PrediCalc provides greater benefits than traditional spreadsheets while preserving the key features of automatic calculation of values and ease of administration. PrediCalc has applications in data management, design, and configuration. Figure 1 shows part of a room management system created using PrediCalc.



Event	Owner	Projection	Room	Time
E1	Art	No	G100	Morning
E2	Bob	Yes	G200	Afternoon
E3	Cal	No	G100	
E4	Art	No		
E5	Art	No		
E6	Cal	No		

Schedule	G100	G200	G300
Morning	E1		
Afternoon			
Evening			

Figure 1: Part of a room management system created using PrediCalc

The primary feature of PrediCalc that we wish to demonstrate is its approach to update. PrediCalc allows for inconsistency between the value assignments and the constraints. This approach differs from the traditional consistency-maintaining techniques [Orm01, MT99]. In addition, PrediCalc shows the consequences of the value assignments, even when the assignments are inconsistent with the constraints. PrediCalc's notion of consequence differs from current notions based on minimal repairs [BC03].

Another important aspect of PrediCalc that we would like to demonstrate is its database-like organization of cells into tables. Because PrediCalc uses structured names for cells, these tables may be queried as if they were database tables, while maintaining the ability to refer to a cell individually by name. In this respect, PrediCalc bridges the gap between traditional databases and spreadsheets.

There have been several other systems that allow for mutidirectional, many-to-many constraints, including LogiCalc [Kri88], FINANZ [FR88], PERPLEX [SB89], Knowledgesheet [GA00] and CsSolver [FFJ+03]. None of these systems allow for propagation under inconsistency or use structured names for cells.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005

Our demonstration will proceed as follows. First, we describe the general user experience of building a spreadsheet in PrediCalc. We then proceed to illustrate the update capabilities of PrediCalc through an example. Finally, we focus on the formula language used by PrediCalc, describing how it allows for database-style querying.

2 The User Experience

The user interface of PrediCalc is somewhat different to that of traditional spreadsheets. A user creating a new spreadsheet document with PrediCalc is greeted with a blank canvas, a textual constraint editor, and a domain editor. The experience is similar to that of other canvas-based WYSIWYG tools, such as visual Web page designers. The user begins by placing cells and textual labels on the canvas. The user may also place static text onto the canvas, change the color scheme, etc.

A cell may have any number of modalities, such as a drop-down list or a type-in field. In addition, cells may be arranged into tables, complete with row and column names. This arrangement of cells into tables serves not only to visually organize cells, but also allows cells to be given names based on their rows and columns. Two tables created in PrediCalc are shown in Figure 1. The first table has six rows, representing events which need to be scheduled, and four columns, containing some properties of the events, namely their owner, whether a projector is required, their room, and their time. The second table represents the event schedule, where each cell contains the event scheduled in a given room at a given time.

Once the cells and tables are laid out, the user can create constraints that express relationships between cells. The constraints are written textually using a variant of first order logic (the formula language is explained in more detail in a subsequent section). The user can also create domains for cells using a textual editor and associate each cell with a domain. These domains are used to populate cell drop-down lists.

Finally, the user may proceed to use the newly created spreadsheet. As the user enters and deletes values from cells, the values in other cells are changed automatically. PrediCalc's approach to update is described in the next section.

3 Updates in PrediCalc

The primary feature of PrediCalc that we wish to demonstrate is how it handles updates. We begin by describing the update semantics abstractly, and then illustrate with a concrete example.

PrediCalc dynamically divides cells into "base cells" and "computed cells." The computed cells contain consequences of the base cells. After each update, the partitioning of base and computed cells changes and then the set of computed facts is recalculated. This process is described in more detail in the next several paragraphs.

In PrediCalc, a user can modify one cell at a time. The user can assign a value to an empty cell, change the value currently assigned to a cell, or empty a cell that currently has a value. When directly modified by the user, a cell automatically becomes a base cell. Next, PrediCalc determines a set of cells which are to lose their status as base cells and become computed cells. In the case of a new value assignment to a cell, these are the base cells with values that, together with the constraints, directly contradict the newly assigned value. When the user empties a cell, these are the cells with values that, together with the constraints, directly entail a value in the emptied cell.

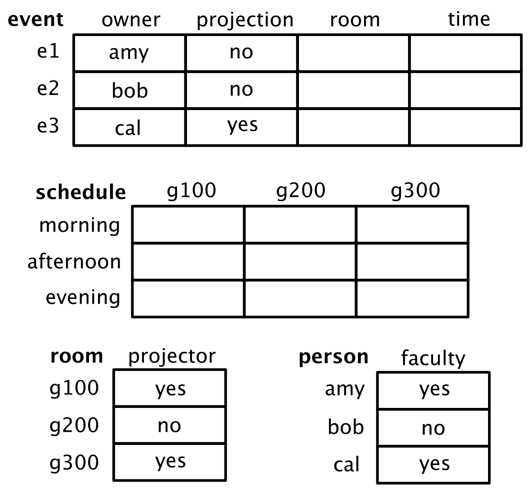
In the case that two or more base cells have values that together contradict the newly assigned value but none does individually, these cells are left as is and do not lose their status as base cells. This leads to inconsistency. Similarly, if two or more cells have values that together entail a value in the newly emptied cell but none does individually, these cells are left as is and remain base cells. This leads to the newly empty base cell having an entailed value. Note that since the newly emptied cell is now a base cell, the cell does not contain a computed value and remains empty.

PrediCalc then proceeds to modify the values in the computed cells. The computed cells contain consequences of the values in the base cells and the constraints. For this purpose PrediCalc uses a paraconsistent consequence relation called existential Ω-entailment. A set of value assignments Λ *existentially* Ω-*entails* a value assignment ϕ if and only if there is some subset of value assignments $\lambda \subseteq \Lambda$ consistent with a set of constraints Ω such that $\lambda \cup \Omega$ logically entails ϕ.

PrediCalc fills the computed cells with the existential Ω-consequences of the base values and the constraints. There is one complication, however, which is that a cell can hold at most one value. If more than one value is existentially Ω-entailed in a computed cell, then PrediCalc uses inertia as a tie-breaker: if the cell contained a value before the update and the value is still existentially Ω-entailed, then that value remains in the cell. If there are multiple existentially Ω-entailed values for a computed cell but none of these was in the cell before the update, the cell is left empty.

Unlike current approaches which define a consequence of an inconsistent database as a logical consequence of *all* minimal repairs of the database [BC03], in PrediCalc a consequence of an inconsistent set of assignments is defined as a logical consequence of *some* repair of the database.

We illustrate the approach using the room management system of Figure 1 as an example. The room manager consists of four tables, shown schematically



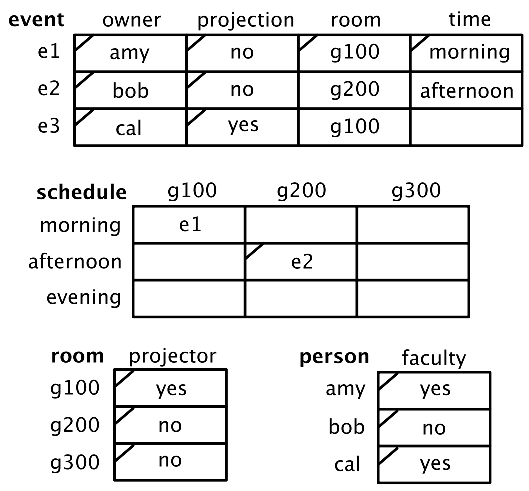
event	owner	projection	room	time
e1	amy	no		
e2	bob	no		
e3	cal	yes		

schedule	g100	g200	g300
morning			
afternoon			
evening			

room	projector
g100	yes
g200	no
g300	yes

person	faculty
amy	yes
bob	no
cal	yes

Figure 2: After creating three events



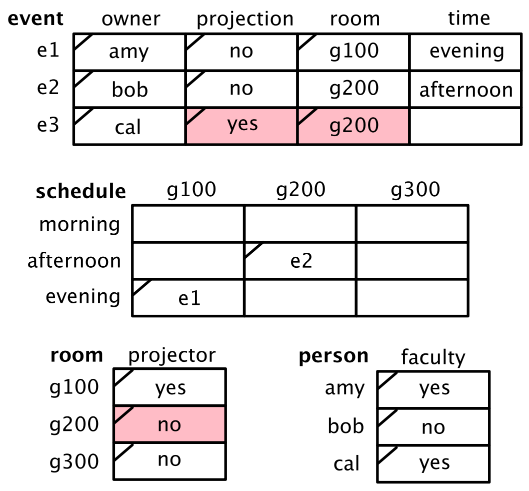
event	owner	projection	room	time
e1	◤ amy	◤ no	◤ g100	◤ morning
e2	◤ bob	◤ no	g200	afternoon
e3	◤ cal	◤ yes	g100	

schedule	g100	g200	g300
morning	e1		
afternoon		◤ e2	
evening			

room	projector
g100	◤ yes
g200	◤ no
g300	◤ no

person	faculty
amy	◤ yes
bob	◤ no
cal	◤ yes

Figure 3: Spreadsheet with e1 and e2 scheduled



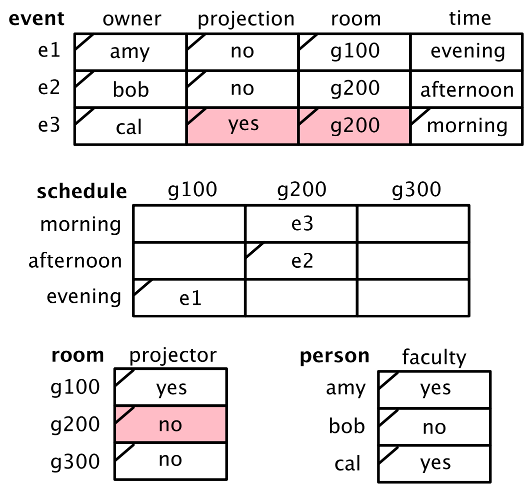
event	owner	projection	room	time
e1	◤ amy	◤ no	◤ g100	evening
e2	◤ bob	◤ no	g200	afternoon
e3	◤ cal	◤ yes	◤ g200	

schedule	g100	g200	g300
morning			
afternoon		◤ e2	
evening	◤ e1		

room	projector
g100	◤ yes
g200	◤ no
g300	◤ no

person	faculty
amy	◤ yes
bob	◤ no
cal	◤ yes

Figure 4: Spreadsheet with inconsistency



event	owner	projection	room	time
e1	◤ amy	◤ no	◤ g100	evening
e2	◤ bob	◤ no	g200	afternoon
e3	◤ cal	◤ yes	◤ g200	◤ morning

schedule	g100	g200	g300
morning		e3	
afternoon		◤ e2	
evening	◤ e1		

room	projector
g100	◤ yes
g200	◤ no
g300	◤ no

person	faculty
amy	◤ yes
bob	◤ no
cal	◤ yes

Figure 5: Showing consequences under inconsistency

in Figure 2. [1] The top table contains event requests, each of which has an owner, a specification of whether a projector is needed, a room, and a time. The center table contains a schedule of the events. The information is redundant with the first table but is useful because it offers a different view. The bottom-left table lists whether or not each room has a projector. The bottom-right table lists whether each person is a faculty member or not. Base cells contain a triangle in the upper left-hand corner of the cell, while computed cells do not.

We consider an administrator whose task is to assign three new events a room and a time. The administrator starts with empty schedule table and event table. She creates three new events in the event table and, for each, fills in the event owner's name and whether a projector is needed. As shown in Figure 2, the system responds by automatically filling in the room of event e3 as g100, since e3 requires a projector, and g100 is the only room with a projector.

The administrator then selects g100 as the room for event e1 and morning as the time in the event table, causing e1 to show up in the corresponding cell in the schedule table. The administrator then assigns e2 to g200 in the afternoon by modifying the schedule table directly, causing the corresponding values to appear in the event table. This leads to the state shown in Figure 3. This illustrates PrediCalc's ability to do propagation in multiple directions.

Next, the administrator moves e1 to g100 in the evening by modifying the schedule table, resulting in e1's time being changed to the evening in the event table. This illustrates how the update algorithm deals

[1] While this illustrative example is small, PrediCalc can handle large numbers of cells and complex constraints.

```
exists(R, event[E,room](R))
exists(T, event[E,time](T))
schedule[T,R](E) <=> event[E,time](T) and event[E,room](R)
event[E,projection](yes) and event[E,room](R) => room[R,projector](yes)
event[E,owner](P) and person[P,faculty](no) => not event[E,room](g100)
```

Figure 6: Constraints for the room manager. The first two constraints dictate that every event has a room and a time. The third constraint relates the schedule table to the event table. The fourth states that events that require a projector must be scheduled in a rooms with a projector. The fifth states that only faculty members can reserve room g100. Note that free variables are considered to be universally quantified.

with direct conflicts. The administrator then changes the room assignment for e3 to g200. Since e3 requires a projector but g200 lacks a projector, this leads to a conflict. As shown in Figure 4, PrediCalc marks the conflicting cells in red. This shows how PrediCalc deals with conflicts caused by multiple cells.

The administrator does not have to resolve the conflict immediately. In fact, she instead proceeds to set the time of e3 to the morning. The event e3 then appears in the schedule table, as shown in Figure 5. This demonstrates PrediCalc's use of existential Ω-entailment to show the consequences of the (inconsistent) base assignments. Finally, the administrator moves the projector from g100 into g200, removing the conflict and resulting in a complete assignment to all events (not shown).

4 Formula Language

As mentioned above, PrediCalc allows for structured names for cells. For example, the structured name schedule[morning,g100] refers to the cell in the schedule table in the morning row and the g100 column. This sort of structured name allows rows and columns to be quantified over. PrediCalc structured names are similar to the structured predicate names allowed in the logic programming language HiLog [CKW93].

In addition to improving the user experience by reducing the replication of typically required in a traditional spreadsheet, structured names allow tables to be queried in a manner similar to database tables. Indeed, since all rows in a PrediCalc table are named, one can either treat a row as tuple with attributes named by the columns, or treat a column as a tuples with attributes named by the rows.

Formulas in PrediCalc can be built up from these structured names and the usual logical connectives and, or, not, =>, <=, and <=> and the quantifiers forall and exists. There are no restrictions on these formulas; since everything is reducible to unary relations, satisfiability is decidable [BJ02]. For convenience, PrediCalc also allows users to define new n-ary relations using <=> and use these in an unrestricted manner. Again, decidability is preserved since these n-ary relations are reducible to unary ones. Figure 6 shows the set of constraints for the room manager.

5 PrediCalc and Databases

A PrediCalc spreadsheet is a special kinds of database, in particular, a database with unary, single-valued relations. However, the techniques employed by PrediCalc could be generalized to general databases. In particular, its approach to dealing with ambiguity and inconsistency could be applied to databases with integrity constraints.

References

[BC03] L. Bertossi and J. Chomicki. Query answering in inconsistent databases. In *Logics for Emerging Applications of Databases*, 2003.

[BJ02] G. Boolos and R. Jeffrey. *Computability and Logic*. Cambridge University Press, 2002.

[CKW93] W. Chen, M. Kifer, and D. Warren. Hilog: A foundation for higher-order logic programming. *J. Log. Program.*, 15(3), 1993.

[FFJ$^+$03] A. Felfernig, G. Friedrich, D. Jannach, C. Russ, and M. Zanker. Developing constraint-based applications with spreadsheets. *IEA/AIE*, pages 197–207, 2003.

[FR88] G. Fischer and C. Rathke. Knowledge-based spreadsheets. In *AAAI*, 1988.

[GA00] G. Gupta and S. Akhter. Knowledgesheet: A graphical spreadsheet interface for interactively developing a class of constraint programs. In *PADL*, pages 308–323, 2000.

[Kri88] F. Kriwaczek. Logicalc: a prolog spreadsheet. *Machine Intelligence 11*, 1988.

[MT99] E. Mayol and E. Teniente. A survey of current methods for integrity constraint maintenance and view updating. In *ER Workshops*, pages 62–73, 1999.

[Orm01] L. Orman. Transaction repair for integrity enforcement. *TKDE*, 13(6):996–1009, 2001.

[SB89] M. Spenke and C. Beilken. A spreadsheet interface for logic programming. In *Proc. of CHI-89*, pages 75–80, 1989.