The Knowledge Engineering Review, Vol. 22:3, 213–219. © 2007, Cambridge University Press
doi:10.1017/S0269888907001191 Printed in the United Kingdom

An introduction to logical spreadsheets

MICHAEL KASSOFF[1] and ANDRE VALENTE[2]

[1]*226 Gates Hall, Computer Science Department, Stanford University, Stanford, California 94305, USA;*
e-mail: mkassoff@stanford.edu;
[2]*Alelo, Inc., 11965 Venice Blvd, Los Angeles, California 90066, USA;*
e-mail: avalente@alelo.com

Abstract

A logical spreadsheet is a spreadsheet in which the formula language is composed of logical expressions. Logical spreadsheets were invented shortly after traditional electronic spreadsheets were introduced, but since then logical spreadsheet research has been somewhat sparse. Recently, however, there has been a resurgence in the interest of logical spreadsheets in the research community. In this article, we summarize logical spreadsheet research up to this point.

1 Introduction

The world's first electronic spreadsheet, VisiCalc, was created by Dan Bricklin and Bob Frankston in 1979. The spreadsheet was widely regarded as the 'killer app' for personal computers. It managed to hit a sweet spot between usability and functionality—millions of users with no formal training in programming were suddenly enabled to create custom applications of their own.

Today, the spreadsheet remains as popular as ever. Indeed, in 2001 there were an estimated 45 million end-users of spreadsheets or databases in the United States alone, representing 60% of the American workforce (Scaffidi *et al*., 2005), and this number is rising.

Spreadsheets do one thing and they do it well, which is to perform mathematical computations. While spreadsheets do have built-in Boolean functions and conditional functions, it is clear that they were not designed from the ground up to support logical reasoning. In modern commercial spreadsheets, logical functions have an inelegant syntax and inappropriate editing tools. Worse, the logical reasoning allowed is quite limited—there are no *n*-ary predicates, unification routines, negation-as-failure capabilities, and the like.

By extending a spreadsheet with proper logical reasoning capabilities, we allow them to be used for many additional applications, such as applying business rules, performing symbolic what-if analyses, and transforming data from one representation to another.

By incorporating further enhancements such as generalizing formulas to constraints, spreadsheets can be further used for applications such as data entry and validation, enterprise management, and constraint-solving.

In the sequel, we describe the work that has been done on logical spreadsheets up until today, and attempt to categorize the different logical spreadsheet systems that have been produced.

2 History

LogiCalc (Kriwaczek, 1988) was the first logical spreadsheet. It was developed by Frank Kriwazek at Imperial College in the early 1980s as part of his Master's thesis work. In LogiCalc, spreadsheet

https://doi.org/10.1017/S0269888907001191 Published online by Cambridge University Press

formulas were written using traditional spreadsheet syntax, but are translated internally to Prolog rules. For example, if C2 is defined as B2 + B3, then the following rule would be created:

has-definition ('C2', X)
if has-value ('B2', Y)
and has-value ('B7', Z)
and X = Y + Z

LogiCalc allowed regions of the spreadsheet to be saved as database relations and then queried. The queries were restricted to be conjunctive, e.g.:

Which [(YZ): father-of('Bob', Y) and friend-of(Y,Z)]

The user could either display all solutions to a query by filling a column of the spreadsheet or by displaying a single solution to the query in a cell and then cycling through the answers. Tuples were held in a single cell, as opposed to displaying each component of a tuple in a separate column.

LogiCalc also allowed database tables to define functions. For example, given facts of the form:

father-of ('Bob', 'Dan')
father-of ('Art', 'Cal')
father-of ('Art', 'Coe')

One could define a new function, @father-of, which mapped children to their fathers. These functions can be incorporated into spreadsheet formulae, such as:

A3 = @age-of(@father-of(B3))

The idea of using a spreadsheet as an interactive constraint solver dates back to LogiCalc. Users could specify constraints over cells, e.g.:

profession-of (A1, 'barber')
father-of (A1,A2)
father-of (A2,A3)
father-of (A3,A4)

Given no assignments to the cells A1–A4, LogiCalc would fill the cells with an appropriate barber–son–grandson–greatgrandchild relationship. The user could then choose to cycle through all such satisfying combinations of values, or he/she could stick with one or more of the values and cycle through the rest. The user also had the ability to declare that a cell either have some given value or that it is not allowed to take a given value.

This interactive constraint-solving capability was improved upon by van Emden *et al.* (1986), whose system allowed for previous computation to be reused when a new constraint was incrementally added.

The constraints allowed in LogiCalc and van Emden's system were quite simple by Prolog standards. They consisted of a single conjunction of conditions, which is what can be expressed by a single Prolog rule. However, Prolog itself allows for a set of rules to be defined (which allows for disjunctive conditions), and furthermore, the rules may be recursive. PERPLEX (Spenke & Beilken, 1986) allowed for new constraints to be defined in terms of old ones, using Prolog rules.

Like its predecessors, PERPLEX allowed constraints to be defined using database relations. It also allowed for constraints based on built-in predicates, which could handle operations like arithmetic, string concatenation, and the like. Each built-in predicate had a set of legal input–output modes, which identified each argument of the predicate as either accepting input or producing output. For example, the add predicate [where add (A1 A2 A3) means that A1 + A2 = A3], had four input–output modes, namely (in in out), (in out in), (out in in), and (in in in). The way that constraint propagation was done was to find a constraint with all the input parameters for

https://doi.org/10.1017/S0269888907001191 Published online by Cambridge University Press

one input–output mode supplied, compute the outputs, and repeat. For user-defined predicates defined using Prolog rules, the set of legal input–output modes was automatically computed.

PERPLEX's handling of numerical constraints is simple, but incomplete. For example, consider the constraint +(A1 A1 2). Though A1 must equal 1, PERPLEX cannot determine this. Note that something more sophisticated than input–output modes is required to solve this constraint, since adding the input–output mode (out out in) will also trigger propagation when +(A1 B1 2) is given, which is not solvable. Similarly, consider constraints $A2 \geq 2$ and $A2 \leq 2$. In this case, it must be that $A2 = 2$, but the constraints must be considered together to determine this, which PERPLEX does not do.

A generalization of logic programming called constraint logic programming allows for sophisticated numerical and symbolic constraint-solving. While the most basic operation in logic programming is unification, in constraint logic programming, unification is replaced by constraint propagation, of which unification is a special case. There are many flavors of constraint propagation; in particular, there are versions that can handle linear constraints over real numbers, or over finite domains. Knowledgesheet (Gupta & Akhter, 2000) is a logical spreadsheet with a finite domain constraint logic programming engine as its constraint solver, developed in the late 1990s. In Knowledgesheet, the user assigns constraints over the cells, and associates each cell with a finite domain. The user may also assign values to cells. Once satisfied with the constraints, the user presses a 'solve' button, at which time the constraints are compiled into a constraint logic program and solved, resulting in the rest of the cells being assigned values if a solution exists. Note that this is different than the previously described systems in that propagation does not occur automatically as values are assigned to cells and constraints are created and removed. This behavior is desirable for large problems that may take minutes or longer to solve.

A logical spreadsheet system with similar capabilities to Knowledgesheet is CsSolver (Felfernig *et al.*, 2003). As opposed to using a logic programming engine to solve constraints, however, a constraint programming engine is used. In addition to finding a solution that simply satisfies the constraints, CsSolver also allows the user to specify an optimization function and find an optimal solution.

PrediCalc (Kassoff *et al.*, 2005) is a logical spreadsheet designed for interactive constraint-solving. Cells in PrediCalc are represented as single-valued, unary relation constants, i.e. $p(a)$ represents the fact that cell p contains the value a. To allow for a more compact representation of constraints, PrediCalc allows for structured cell names, i.e. $q(r,c)$ might represent the name of a cell in row r and column c of table q. Constraints in PrediCalc are written in universal first-order logic.

Unlike a traditional spreadsheet, PrediCalc's cells may be laid out arbitrarily on a canvas. This notion of 'liberated cells' is also found in FINANZ (Fischer & Rathke, 1988).

PrediCalc automatically resolves inconsistencies when it can; otherwise, it alerts the user to spreadsheet values that conflict with the constraints. Since the spreadsheet values may be inconsistent with the constraints, PrediCalc uses a paraconsistent entailment relation to determine the computed spreadsheet values.

In 2004, believing that logical spreadsheets should soon be a commercial reality, Defense Advanced Research Projects Agency (DARPA) put out a Small Business Innovation Research call for proposals on 'deductive spreadsheets' (Gunning, 2004). It read:

> 'One of the most useful and pervasive computing tools today is the spreadsheet. Novices and power users alike are able to use this flexible tool to do everything from performing simple arithmetic to programming complex financial analyses. We would like to develop an analogous tool for logical reasoning: a deductive spreadsheet that would allow users to define logic statements and inference rules for symbolic reasoning in the same way that current spreadsheet allow users to define mathematical formulae for numerical calculations. Such a tool could provide users with a powerful reasoning tool to capture decision rules, to perform what if analyses, and to translate and transform symbolic data from one form to another.'

https://doi.org/10.1017/S0269888907001191 Published online by Cambridge University Press

In the end, four projects were funded: NEXCEL, LESS (Logic Embedded in SpreadSheets), XcelLog, and the unnamed system of Waltzman *et al.* All four systems extend Microsoft Excel. The familiarity of most end-users with Excel was an important criterion for DARPA, since its envisioned military uses of logical spreadsheets required quick decisions to be made in time-critical situations.

In all four of these systems, the tabular format of the traditional spreadsheet, the immediate feedback that users receive when changing a value or a formula, and the familiar graphical user interface elements of current commercial spreadsheet systems are all preserved. Furthermore, in all cases the formula language is a conservative extension of the traditional language that users have come to expect from spreadsheet systems. Finally, in all cases the behavior of the spreadsheet is a simple extension of current behavior, namely that the formulas partition the cells into input cells and output cells, where the output cells are a function of the input cells.

In addition to these research systems, Amzi!, a company that developed and commercializes a Prolog system, developed a logical spreadsheet add-on to their product called ARulesXL[1]. ARulesXL came about as a support module to deal with the development of knowledge-based systems where spreadsheets were a natural candidate to allow end-users to enter rules—e.g. an online mortgage pricing application where pricing experts use familiar spreadsheets to enter rules. The system provides ways for users to specify and query logical rules in an Excel spreadsheet, and has some similarities with the approach used in LESS.

With this critical mass of recent work in logical spreadsheets, Mike Kassoff, Michael Genesereth, and Iliano Cervesato organized a workshop on Logical Spreadsheets at Stanford in late 1995. Almost all the research groups behind the recent systems were represented, and lively discussion ensued. A follow-up workshop was organized as part of the Association for the Advancement of Artificial Intelligence (AAAI) 2006 Fall Symposium series, and enhanced the scope of the debate to not only logical spreadsheets but also generally integrating reasoning into everyday applications (Bongard *et al.* 2006).

3 Articles in this special issue

The articles in this special issue present a selection of the best recent work in logical spreadsheets.

Iliano Cervesato presents NEXCEL, a logical spreadsheet implementation that allows users to specify regions of the spreadsheet as relational tables which can hold *n*-ary tuples. The tables are related using Datalog-style definitions, which use stratified negation semantics. NEXCEL allows for unsafe rules in certain situations, to allow for arithmetic spreadsheet formulas within the Datalog rules. An iteration bound is used to halt the otherwise potentially infinite computation of answers. NEXCEL also has facilities for explaining to the user why a certain tuple was or was not computed.

Andre Valente, David Van Brackle, Hans Chalupsky, and Gary Edwards present LESS, their logical spreadsheet implementation. LESS also allows users to specify regions of the spreadsheet as relational tables that can hold *n*-ary tuples. Indeed, LESS is quite flexible in its knowledge input mechanisms, allowing arbitrarily shaped regions of the spreadsheet to be translated into *n*-ary tuples or object definitions. LESS is built upon the PowerLoom knowledge representation and reasoning system (MacGregor *et al.*, 1997), which allows for expressive types of knowledge such as meta-level statements, functional terms, and rules with complex subgoals and consequents, and supports natural deduction, equality reasoning and limited higher-order reasoning.

Marcelo Tallis, Rand Waltzman, and Bob Balzer present a logical spreadsheet that allows for users to specify regions of the spreadsheet as class tables, which allow for the natural representation of subject–predicate–object triples. This system is actually a framework that allows for any number of reasoners to be plugged in to it, to demonstrate its flexibility it has been successfully connected to both a Prolog reasoner and a Rete-based rule engine.

[1] http://www.arulesxl.com/

https://doi.org/10.1017/S0269888907001191 Published online by Cambridge University Press

C. R. Ramakrishnan, I. V. Ramakrishnan, and David Warren present XcelLog, which also uses class tables to represent subject–predicate–object triples. It is built on top of a tabled Prolog engine, though this fact is hidden from the user, who writes formulas using cell references only. The effect is a user interface which allows that the user to produce chain Datalog queries intermixed with traditional spreadsheet formulas.

Michael Kassoff, and Michael Genesereth present PrediCalc, a constraint-based logical spreadsheet that allows for inconsistency. PrediCalc is built upon the Epilog model elimination theorem-prover developed at the Stanford Logic Group. PrediCalc treats cells as single-valued unary relations and allows for clausal constraints to be placed across the cells. PrediCalc uses a paraconsistent entailment relation to compute the consequences of the spreadsheet values.

4 Dimensions of logical spreadsheets

Logical spreadsheet implementations differ in what benefits they intend to bring, and to whom. Some of the implementations discussed above are intended to extend or improve on spreadsheets; some are intended to extend or improve on logic systems. Some of these capabilities are evolutionary in the sense that they gently extend the spreadsheet paradigm, while others provide either a more radical departure from the spreadsheet paradigm or use spreadsheets more as an inspiration than a point of departure.

Some of the ways in which the applications above extend or improve on spreadsheets are:

- reduce errors in spreadsheet design (by checking errors or limiting the ways in which users make mistakes);
- make it easier to perform some common spreadsheet tasks (e.g. specifiy certain constraints or queries);
- allow users to specify what they intend more declaratively;
- provide a different/complementary abstraction;
- make it easier to perform certain tasks (e.g. query across tables);
- make it easier to express/use complex (business) rules, and make them more readable and maintainable;
- use logic to debug and reason about spreadsheet contents;
- add more powerful, expressive modeling capabilities through logic expressions;
- additional new reasoning capabilities (e.g. constraint reasoning);
- help users build models in spreadsheets.

Some of the ways in which the applications above extend or improve on logic systems are:

- using spreadsheets as a platform for deploying knowledge-based systems—reduce barriers to deployment by incorporating the knowledge-based systems into the users' existing tool set;
- using spreadsheets for input and/or output—providing users a familiar user interface to enter knowledge and data;
- reduce the learning curve for using logic systems by embedding logic into a familiar UI metaphor;
- using spreadsheets as a platform for knowledge acquisition—elicit knowledge from users by leveraging the spreadsheet's natural table format and its "read-write"' nature;
- using spreadsheets to provide additional financial/calculation capabilities not always available (or available with considerable effort) in logic systems.

A second dimension in which logical spreadsheets vary is the intended user. This is important because different target users are assumed to be more or less familiarity with spreadsheets, or with logic. For instance, many applications mentioned above target spreadsheet users (and in particular Microsoft Excel users); some made assumptions as to whether the user would be a beginner, intermediate, or expert. Other applications target programmers who build spreadsheet applications

https://doi.org/10.1017/S0269888907001191 Published online by Cambridge University Press

and assumed the user would be able to do tasks such as simple programming (e.g. in Visual Basic). Some assumed the user would be knowledgeable in some vertical application domain (e.g. logistics, scheduling), while others assumed the user is a business analyst.

Another dimension in which implementations differed was their interpretation of ''spreadsheet'' and ''logic''. Some research groups interpreted 'spreadsheet' as meaning only a spreadsheet-style user interface; others assumed spreadsheets to be any descendant of VisiCalc; still others concentrated on a specific system or implementation (most often Microsoft Excel, which is the most widely available implementation in the market). The meaning of 'logic' also varied. The type of logic system used included logic programming systems, constraint reasoners, first-order (or even higher-order) logic systems, and specialized relational database systems with deductive capabilities.

All of the above choices lead to other choices in user interface, and more specifically in how much to depart from (and integrate with) the spreadsheet paradigm. In some cases, implementers chose to make the logic functionality seamless with the spreadsheet implementation, e.g. by making it available through spreadsheet function libraries. Some chose to add user interface elements (menu items, keystrokes) that do not change the underlying functions. A popular choice was to add wizards and debugging tools, reflecting the assumption that logic would be too hard for spreadsheet users and needed to be hidden or carefully wrapped. Some implementations chose to take over the semantics and behavior of cells, for instance, making cells contain lists, logic objects, etc. A few implementations decided to take over the semantics and behavior of worksheets, e.g. by introducing a special type of worksheets. Finally, some implementations decided to take over the semantics and behavior of formulas, or introduce entirely new graphical user interface elements.

5 Conclusion

Logical spreadsheets remain an interesting and fruitful area of research and development. Much work remains in making logical spreadsheets easy to use; in particular, writing and debugging complex logical rules can be a daunting task for the non-expert. Nonetheless, the systems detailed in this special issue show much promise in this regard, and if coupled with an appropriate level of marketing and business sophistication should find their 'killer app' soon.

References

Bongard, J. *et al*. 2007 Reports on the 2006 AAAI Fall Symposia. *AI Magazine* **28**(1), 88–92.

Felfernig, A., Friedrich, G., Jannach, D., Russ, C. and Zanker, M. 2003 developing constraint-based applications with spreadsheets. In *Proceedings of the 16th International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE 2003)*. Loughborough, UK, 23–26 June 2003, pp. 197–207.

Fischer, G. and Rathke, C. 1988 Knowledge-based spreadsheets. In *Proceedings of the 7th National Conference on Artificial Intelligence*, *St. Paul, Minnesota, 21–26 August 1988*. AAAI press, Menlo Park, CA pp. 802–807.

Gunning, D. 2004 Deductive spreadsheets. Defense Advanced Research Projects Agency Small Business Innovation Research 2004.3-Topic SB043–040.

Gupta, G. and Akhter, S. 2000 Knowledgesheet: A graphical spreadsheet interface for interactively developing a class of constraint programs. In *Proceedings of the Second International Workshop on Practical Aspects of Declarative Languages (PADL'00)*, *Boston, Massachusetts, 17–18 January 2000*. Springer, Berlin, Germany pp. 308–323.

Kassoff, M., Zen, L., Garg, A. and Genesereth, M. R. 2005 PrediCalc: a logical spreadsheet management system. In *Proceedings of the 31st International Conference on Very Large Data Bases*, *Trondheim, Norway, 30 August–2 September 2005*. ACM, New York, NY pp. 1247–1250.

Kriwaczek, F. 1988 LogiCalc: a Prolog spreadsheet. *Machine Intelligence*, **11**, 193–208.

MacGregor, R., Chalupsky, H. and Melz, E. 1997 *PowerLoom manual*. University of Southern California. Available online at `http://www.isi.edu/isd/LOOM/PowerLoom/documentation/manual.html`.

https://doi.org/10.1017/S0269888907001191 Published online by Cambridge University Press

Scaffidi, C., Shaw, M. and Myers, B. 2005 Estimating the numbers of end users and end user programmers. In *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE Computer Society, Los Alamitos, CA pp. 207–214.

Spenke, M. and Beilken, C. 1989 A spreadsheet interface for logic programming. In Bice, K. and Lewis, C. H. (eds.), *Proceedings of the ACM CHI 89 Human Factors in Computing Systems*, *Austin, Texas, 30 April–4 June 1989*. ACM press, New York, NY pp. 75–80.

van Emden, M., Ohki, M. and Takeuchi, A. 1986 Spreadsheets with incremental queries as a user interface for logic programming. *New Generation Computing* **4**(3), 287–304.

https://doi.org/10.1017/S0269888907001191 Published online by Cambridge University Press