

# Implementing Logic Spreadsheets in LESS

Andre Valente<sup>1</sup>, David Van Brackle<sup>2</sup>, Hans Chalupsky<sup>3</sup> and Gary Edwards<sup>2</sup>

<sup>1</sup> Alelo, Inc.

11965 Venice Blvd, Suite 402, Los Angeles, CA 90066  
avalente@alelo.com

<sup>2</sup> ISX Lab of Lockheed Martin Advanced Technology Labs  
760 Paseo Camarillo, Suite 401, Camarillo, CA 93010  
vanb@isx.com, gedwards@isx.com

USC Information Sciences Institute  
4676 Admiralty Way, Marina del Rey, CA 90292  
hans@isi.edu

## Abstract

Spreadsheets are a widespread tool for a variety of tasks, particularly in business settings. Spreadsheet users employ a form of programming that, while popular, has been shown to be highly error-prone and with limited expressiveness. An important approach to augment spreadsheets is to provide logic knowledge representation and reasoning (KR&R) functionality. In this paper we present LESS (Logic Embedded in SpreadSheets), a system that integrates PowerLoom, a highly expressive logic KR&R, with Microsoft Excel. LESS' design provides different tiers of functionality that explore different trade-offs between direct access to the underlying logic engine and user-friendly support for spreadsheets users.

## Introduction

Spreadsheets are a widespread tool for a variety of tasks, particularly in business settings. There are an estimated 55 million users of spreadsheets in 2005. Spreadsheets embed what is in practice a programming paradigm, based on the model of cells and cell variables. This means we are really talking about 55 million programmers – more than most current programming languages.

Despite their popularity and their image as easy-to-use, spreadsheets are highly error prone. For example, despite current spreadsheet implementations containing a variety of features to find and correct mistakes, Erwig and Burnett [1] estimate that up to 90% of spreadsheets contain errors. This makes clear the need for tools to improve on the spreadsheet paradigm.

A second problem with spreadsheets is their limitations in terms of the expressive power of their programming paradigm. Even some simple models can be are hard to

implement if they do not fit closely the spreadsheet paradigm. A result of these limitations is that many complex models implemented in systems like MS Excel rely on features that extend the spreadsheet paradigm, such as procedural scripts (e.g., in Visual Basic).

Our contention is that logic spreadsheets can be a way to extend the spreadsheet paradigm to both reduce errors and improve the expressiveness. Our implementation of the logic spreadsheet paradigm is the system LESS Logic Embedded in SpreadSheets (LESS). LESS combines the power of logic-based knowledge representation and reasoning with the familiar and easily-mastered user interface paradigm of a spreadsheet.

In this paper we present our work in designing and implementing LESS. We start by presenting the overall design of LESS. Then, we show through several detailed examples how LESS can overcome shortcomings in the spreadsheet approach. We discuss how we can introduce additional user interface elements that are intended to make LESS easier to use and more accessible to the average spreadsheet user. We briefly discuss the results of our implementation, and the insights we learned in the process. Finally, we present our conclusions.

## Architecture

LESS is built on top of Microsoft Excel, the most widely used spreadsheet system. We developed custom-built middleware to provide access to logic reasoning functionality, which is in turn provided by PowerLoom [2, 3]. PowerLoom is a logic-based knowledge representation and reasoning (KR&R) system developed over the last eight years by the Loom KR&R group at USC's Information Sciences Institute. PowerLoom is a maturing system that that has been used successfully in a variety of research programs from DARPA and NSF, and has been distributed to over 500 sites worldwide.

PowerLoom is one of the most expressive and powerful KR&R systems available today, and it has many characteristics that make it particularly adequate to integrate into a logic spreadsheet. First, its base language is the Knowledge Interchange Format (KIF) [4] which is more expressive than available the languages implemented in currently available KR systems such as description logics, logic programming and frame-based reasoners. Second, its engine has advanced inference capabilities such as complex rules, negation, equality reasoning, subsumption, and restricted forms of higher order reasoning. PowerLoom also has a classifier that is able to classify descriptions expressed in full first order predicate calculus. Third, PowerLoom adopts a unique approach that allows users to control how much of that inferential power to use, by implementing different inference levels and resource bounded inference. This way, a user only “pays” for the inference power it needs to use. Fourth, PowerLoom is available in libraries for C++, Java or CommonLisp. Despite of PowerLoom’s inference power and expressivity, the PowerLoom libraries are quite small. For instance, the C++ library that is being integrated with MS Excel takes about 7Mb. Fifth, PowerLoom’s architecture allows for modules to plug into and extend the main PowerLoom knowledge representation system. These facilities have been used to implement hybrid reasoners (e.g., partial inference, simple temporal reasoning, or rule learning), and specialized functions (e.g. units of measure), and to integrate PowerLoom with relational databases. These capabilities facilitate the work of integration with the spreadsheets.

The single biggest problem in building a logic spreadsheet is to balance how much of the integrated system is (or feels, works like) a spreadsheet and how much it is a logic-based KR&R system. There is a natural trade-off between keeping close to the spreadsheet paradigm and providing more powerful ways of using logic reasoning functionality. For example, spreadsheet users are accustomed to being able to access data in any cell, which assumes all cells either contain data or a formula that yields data. However, this turns out not to be the case in logic, where most of the action occurs indirectly, in side effects, and where the result may not be as interesting as data. For instance, the result of an assertion is not one of the values being asserted but rather a logic object that is not particularly useful for calculations. Further, spreadsheet users may balk at learning logic in depth, and thus it is critical to provide mechanisms to make it simpler to leverage the power of logic-based reasoning without requiring deep expertise in logic or knowledge representation.

Our strategy to face that challenge is to have four different integration tiers that provide different points in this space of trade-offs and offer alternatives that are likely to be valuable for different types of users.

**Tier 1:** In this tier, logic-based knowledge representation and reasoning systems allow users to ask questions about knowledge in a knowledge base. The LESS system provides an interface to allow users to make such queries

and use the results in a standard spreadsheet. Many such queries have YES or NO answers. With LESS, users will be able to use the results of such a query in any way they can use any boolean result – to alter the color of cells, to affect calculations, or in any other way now available. For example, in a spreadsheet of servicemen, a cell might be given a background of RED or GREEN based on a knowledge base query to determine suitability for a given task, or a calculation of pay may be affected by a knowledge base query concerning training level. Users will also be able to fill cells with the results of single-valued queries, such as “How many hours of training has this officer experienced in the area of Bomb damage Assessment?”

Users are able to create, destroy, or list knowledge entities such as concepts, instances, relations, rules or facts in the knowledge base through functions available in formulas. Virtually all of the functionality of PowerLoom, from the development of concepts to creation of instances to the specification of relations and the development of sets of rules, is available to the spreadsheet developer to use in formulas. For example, if an officer’s superior changes, a formula in a cell will be able to post this new fact in the knowledge base.

**Tier 2:** In this tier, LESS provides spreadsheet-based mechanisms for viewing and modifying such knowledge in the knowledge base, with rows representing concept instances and columns representing the values of relations. A simple but powerful query capability is also provided, so that tables may be populated in different ways with extant knowledge. Often, a user of a spreadsheet needs to build a new sheet from an old one – for example, selecting a collection of servicemen suitable for a given task into a new spreadsheet, to further determine who is best for the job. Using LESS, users are able to use knowledge in the knowledge base for classification and to extract the appropriate records into a new sheet.

**Tier 3:** At this tier, the spreadsheet is fully integrated with the knowledge base. Any changes made in the spreadsheet are immediately reflected in the knowledge base, and any changes made to the knowledge base are immediately reflected in the spreadsheet. All APIs of the logic system are available to the spreadsheet, and all spreadsheet functions are available to the logic system. Some knowledge has structure that is not natural for tabular representation. For this, LESS provides interfaces, or wizards, which are similar to spreadsheets and which are natural extensions to spreadsheets.

**Tier 4:** At this tier, LESS provides a mechanism for capturing and reusing domain-specific knowledge. Users are able to easily create and capture templates, containing the structure and basic relationships of their knowledge. They can then, perhaps at a later date, instantiate a template, and quickly and easily parameterize it for their current problem. Templates can even make abstract, declarative references describing the data they need, and can be immediately populated by automatic retrieval of

matching knowledge from the knowledge base. With templates, users will be able to describe the structure of their problem once and only once, and not have to expend valuable effort each time they need to solve such a problem with re-defining its structure. Rather, they can focus on parameterizing and solving it. The focus is to be able to develop and distribute precisely tailored “Domain Paks”. For instance, bankers can produce and use a Home Loan Domain Pak, which not only computes the loan amortization but also helps to answer questions regarding whether or not to allow a loan. With Domain Paks, an expert in a given field can capture the essence of a problem in that field, and users can utilize this problem-solving expertise without having to characterize the problem themselves.

At this point we have implemented Tier 1 and portions of Tier 2 and 3. The discussion below focuses on Tier 1. An upcoming article will discuss the other tiers in detail.

### Tier 1 Examples

In this section, we will show a few examples to illustrate how LESS works and how it can add power to standard spreadsheets. The example spreadsheets shown are based on real-life spreadsheets.

To facilitate exposition, the syntax used in some cases is slightly simplified from our current implementation; we are still adding syntactic sugar that will bridge that gap in due time.

#### Example 1: A budget spreadsheet

The first example is a budget spreadsheet used by a small business to write cost proposals using labor hours and pay rates. A typical section of such a spreadsheet (Figure 1) contains information about how many hours each member of the project team will work in each task.

	A	B	C	D	E
1	<b>Task 1</b>	<b>Hours</b>			
2	Person	Jan-05	Feb-05	Mar-05	Apr-05
3	Fred	12	4	12	14
4	Wilma	4	7	15	3
5	Barney	10	8	3	6
6	Total	26	19	30	23

Figure 1: Typical table in a budget spreadsheet.

Normally, the sum cells (B6 to E6) have a simple formula like `=SUM(B3:B5)`. The disadvantage of this approach is that it does not translate well the intent of the user, which is to add the hours for all project members of the month. LESS allows us to use a more declarative specification of that intent by expressing the relationships explicitly. We can use a logic function `(hours ?person ?month ?task)`, meaning the number of hours a ?person spends on a ?task in a ?month (we use KIF syntax, with question marks denoting variables).

In order to add relational information to a spreadsheet, we need to add a lifting formula that expresses how the table information corresponds to logic assertions. One way this can be done in LESS is to add a lifting formula on top of each column:

```
=ASSERTMULTI("hours", A3:A5, B2, $A$1)
```

This expands into and performs a series of assertions, e.g.:

```
(hours A3 B2 A1) = (hours "Fred" Jan-05 "Task 1") = 12
(hours A4 B2 A1) = (hours "Wilma" Jan-05 "Task 1") = 4
```

Once the information is lifted into PowerLoom, we can easily express a cell as “sum the Jan-05 hours in Task 1 for all persons” by setting cell B6 to the formula:

```
=SUM (RETRIEVEALL "(hours ?person Jan-05 "Task 1")")
```

Notice the mix of logic and spreadsheet functions in the formula: SUM is the Excel function; RETRIEVEALL is function implemented by calling the logic mechanisms to perform a PowerLoom RETRIEVE and transforming the results into the equivalent to an Excel cell range.

We also allow a style of parametrical formulas by using cell references in spreadsheet style, in this example:

```
=SUM (RETRIEVEALL "(hours ?person B2 A1)")
```

A situation in which this relational (as opposed to positional) approach shows its advantages is when the information has more than two dimensions. This is intuitively easy to see: since ranges cover a two-dimensional space (rows and columns), a spreadsheet user has trouble representing a relation with arity higher than 3. For instance, let us analyze an example where a user needs to calculate totals from information distributed across tables (Figure 2). It shows a more complex project sheet that contains four tables: three tables contain the hours worked by each project member in a specific task (Task 1, Task 2, Task 3) and the fourth contains aggregate numbers across all tasks. The construction of the aggregate table shows the limitations of positional referencing. Each cell is calculated by adding one cell from each of the cells above, e.g.: `B24=B3+B10+B17`

The problem is that since the information being added is not in a simple table, we can’t use ranges. Indeed, while such formula is easy to create for a small number of tasks, it can easily get cumbersome if there are many tasks. (The real-life example sheet from where this was abstracted sheet contained 15 tasks/tables with up to 20 people each, and it was constantly being enlarged.) Further, it is extremely hard to maintain and error prone – for instance, if a new task is added.

	A	B	C	D	E
1	<b>Task 1 Hours</b>				
2	Person	Jan-05	Feb-05	Mar-05	Apr-05
3	Fred	12	4	12	14
4	Wilma	4	7	15	3
5	Barney	10	8	3	6
6	Total	26	19	30	23
7					
8	<b>Task 2 Hours</b>				
9	Person	Jan-05	Feb-05	Mar-05	Apr-05
10	Fred	4	7	4	7
11	Wilma	12	12	4	14
12	Barney	10	8	10	8
13	Total	26	27	18	29
14					
15	<b>Task 3 Hours</b>				
16	Person	Jan-05	Feb-05	Mar-05	Apr-05
17	Fred	12	12	4	14
18	Wilma	4	4	7	3
19	Barney	10	10	8	6
20	Total	26	26	19	23
21					
22	<b>Total Hours</b>				
23	Person	Jan-05	Feb-05	Mar-05	Apr-05
24	Fred	28	23	20	35
25	Wilma	20	23	26	20
26	Barney	30	26	21	20
27	Total	78	72	67	75

Figure 2: Tables for specifying hours in different tasks in a budget spreadsheet.

As before, the problem is that the formula  $B_{24} = B_3 + B_{10} + B_{17}$  does not translate well the intent of the user, which is to add the hours Fred worked in all tasks in Jan-05 (or, to be more relative, to add the hours worked by project member specified in cell B23). As we can see, while the data each Task table can be seen as a model with two variables (person and monthly hours), the overall set reflects indexing in three dimensions – person, month and task. When information needs to be aggregated across more than two dimensions, the spreadsheet model suffers.

LESS can solve these limitations by resorting to relational referencing – indeed, the approach to model this problem is very similar to the approach used for the previous example. Assuming the same lifting described earlier for each task table, we can easily express “sum the hours in Jan-05 for Fred in all tasks” as:

```
=SUM (RETRIEVEALL ("(hours Fred Jan-05 ?task)"))
```

This can also easily be turned in to a more spreadsheet-like formula by using cell variables instead of data:

```
=SUM (RETRIEVEALL ("(hours A24 B$23 ?task)"))
```

### Example 2: A grading spreadsheet

This example is based on a grading spreadsheet used to calculate grades for students in a class, and which was also used as a class exercise in teaching spreadsheets (Figure 3).

	A	B	C	D	E	F	G	H
1		<b>Assignments</b>		<b>Tests</b>		<b>Exam</b>	<b>Final</b>	<b>Letter</b>
2	Student	1	2	1	2	Grade	Grade	Grade
3	Parker	56	84	89	93	90	81	B-
4	May	95	92	79	83	95	90	A-
5	Watson	89	87	98	78	80	86	B
6	Jameson	66	95	99	69	69	79	C+
7	Osborn	88	51	79	63	79	72	C-

Figure 3: Grading spreadsheet example.

The letter grade calculated in column H depends of where the numeric grade in column G lies in a number of ranges. For instance, if a numeric grade is lower than 70, the letter grade is D; if the numeric grade is higher than 70 but lower than 75, the letter grade is C-, etc.

Spreadsheets like Excel allow conditionals that can be cascaded, so such rules can be modeled with “cascades ifs”. For example, a typical formula for cell H3 would look like:

```
IF($H4<70,"D", IF($H4<75,"C", IF($H4<78,"C-", IF($H4<80,"C+", IF($H4<85,"B-", IF($H4<88,"B", IF($H4<90,"B+", IF($H4<95,"A-", "A"))))))))
```

This formula has three major problems. First, it is all but unreadable, which makes it highly error-prone. Second, it is very hard to write. When a similar example was used in an assignment for business students at a class given by one of the authors, 60% of the students were unable to come up with the correct formula. Third, the formula is completely hardwired in terms of the ranges used (e.g., 74 being the higher bound for a C- grade). This makes it hard to perform consistent grade adjustments by modifying the ranges for each letter grade – a common practical need in real-life classes.

In LESS we improve on the spreadsheet model by allowing users to define rules explicitly and separately, and apply the rules when needed. This helps with maintenance problems when it is necessary to modify the rules.

In the same example, once the rules are defined, the formula to calculate the grade letter (column H) would become a simple query:

```
RETRIEVE("1 (letter-grade ", A3, " ?x) " )
```

In LESS, rules can be defined in several different ways. The first way is to write them explicitly as shown in Figure 4. A notable difference is that there is one rule for each letter grade, which makes the overall rule set more clear and modular. Each cell contains one rule; the formula in the cell is used to assert the rule in PowerLoom. In this case all elements of the rule are instantiated. This has the advantage that the rules are self-contained, but it makes them also less flexible and parameterizable.

A more flexible modeling style – and one that fits well the spreadsheet paradigm – is to use *rule templates* that are defined on Excel data. For instance, Figure 5 shows how a rule template can be defined based on abstract (non-cell) variables, and a separate formula indicates how to bind those logic variables to cells.

	A	B	C	D	E	F	G
11	<b>Classification rules - Version 1</b>						
12	!=assert(=> (>= ?x 95) (letter-grade ?x "A"))						
13	!=assert(=> (and (< ?x 95) (>= ?x 90))(letter-grade ?x "A-"))						
14	!=assert(=> (and (< ?x 90) (>= ?x 88))(letter-grade ?x "B+"))						
15	!=assert(=> (and (< ?x 88) (>= ?x 85))(letter-grade ?x "B"))						
16	!=assert(=> (and (< ?x 85) (>= ?x 80))(letter-grade ?x "B-"))						
17	!=assert(=> (and (< ?x 80) (>= ?x 78))(letter-grade ?x "C+"))						
18	!=assert(=> (and (< ?x 78) (>= ?x 75))(letter-grade ?x "C"))						
19	!=assert(=> (and (< ?x 75) (>= ?x 70))(letter-grade ?x "C-"))						
20	!=assert(=> (< ?x 70) (letter-grade ?x "D"))						

Figure 4: Representing individual rules in LESS.

	A	B	C	D	E	F	G	H	I
38	Min	Max	Letter						
39		95	100 A						
40		90	94 A-						
41		88	89 B+						
42		85	87 B						
43		80	84 B-						
44		78	79 C+						
45		75	77 C						
46		70	74 C-						
47		0	69 D						
48									
49	Template:	=assert(=> (and (<= ?x ??P1) (>= ?x ??P2) ((letter-grade ?x ??P3)))							
50		=apply-template(B49,A39:C47)							

Figure 5: Representing parameterized rules in LESS.

## Implementation

LESS was implemented as an Excel add-in that includes the PowerLoom C++ libraries. The whole file takes less than 4Mb, and it is quick enough that we have not yet to noticeably slow down Excel in the examples we have built (some of them sizeable spreadsheets). This proved one of our original claims, that one of the advantages of PowerLoom is its small footprint and high speed.

Another important property of PowerLoom that proved relevant was the ease of integration and extension. When we started designing the integration and implementing a prototype, it turned out that the existing APIs available in PowerLoom were not only particularly relevant (built precisely for this type of integration) but also easy and quick to extend. For instance, we were able to create a simple Excel C++ wrapper for PowerLoom in days. PowerLoom's ease of integration was also extremely important in allowing us to experiment with different ideas.

## Related Work

The idea of marrying spreadsheets and logic is not really new. The interest in spreadsheets and their problems has increase in the last few years in part fueled by the seminal work in the analysis of spreadsheets and their problems by Martin Erwig and his colleagues at Oregon State University. Their initial work [1] was dedicated to the identification of spreadsheets as a form of programming and some of its shortcomings. Later, Abraham and Erwig [5] proposed a system to uncover errors in spreadsheets by

defining a ``unit system''. The units in their work are similar to the structure of relations to represent information in tables in LESS.

The closest related work to the one presented here are the implementations of logic spreadsheets discussed in the Workshop on Logical Spreadsheets (WOLS'05). A detailed comparison between these implementations will be discussed in detail in a forthcoming article

## Conclusions

LESS is a practical, real-life implementation of the concept of logic spreadsheets. Our experience showed that logic spreadsheets can overcome some of the limitations of spreadsheets, as well as open the door to additional applications for which spreadsheets alone are not well-suited.

## References

- [1] Adding Apples and Oranges, Martin Erwig and Margaret M. Burnett, 4th Int. Symp. on Practical Aspects of Declarative Languages, LNCS 2257, 173-191, 2002
- [2] R. MacGregor, H. Chalupsky, and E. Melz. PowerLoom manual. University of Southern California, 1997. Available online at <http://www.isi.edu/isd/LOOM/PowerLoom/documentation/manual.html>.
- [3] MacGregor, Robert M. ``A Description Classifier for the Predicate Calculus'', in Proceedings Twelfth National Conference on Artificial Intelligence, 1994.
- [4] M. R. Genesereth. Knowledge Interchange Format. Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference, Cambridge, MA, pages 599-600. Morgan Kaufmann, 1991.
- [5] Header and Unit Inference for Spreadsheets Through Spatial Analyses, Robin Abraham and Martin Erwig, IEEE Int. Symp. on Visual Languages and Human-Centric Computing, 165-172, 2004.

## Acknowledgements

The research reported here was sponsored by the DARPA Small Business Innovative Research program and Dr. David Gunning of the DARPA IPTO office.