The Knowledge Engineering Review, Vol. 22:3, 237–253. © 2007, Cambridge University Press
doi:10.1017/S0269888907001178 Printed in the United Kingdom

Implementing logic spreadsheets in LESS

ANDRE VALENTE[1], DAVID VAN BRACKLE[2], HANS CHALUPSKY[3] and GARY EDWARDS[2]

[1]*Alelo, Inc., 11965 Venice Boulevard, Los Angeles, CA 90045, USA;*
e-mail: avalente@alelo.com;
[2]*Lockheed Martin Advanced Technology Laboratories, 3 Executive Campus, 6th Floor, Cherry Hill, NJ 08002, USA;*
e-mail: david.van.brackle@lmco.com, gedwards@atl.lmco.com;
[3]*USC Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292, USA;*
e-mail: 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, although popular, is highly error-prone and has limited expressiveness. A promising approach to overcome these shortcomings is to augment spreadsheets with logic-based knowledge representation and reasoning (KR&R) functionality. In this paper, we present Logic Embedded in SpreadSheets (LESS), a system which integrates PowerLoom, a highly expressive logic-based KR&R system, with Microsoft (MS) Excel. The design of LESS provides different tiers of functionality that explore trade-offs between direct access to the underlying logic engine and user-friendly support for spreadsheets users. A prototype of LESS was implemented as an MS Excel add-in.

1 Introduction

An estimated 55 million people used spreadsheets in 2005, evidence that spreadsheets are a widespread tool, particularly in the business environment, with more users than that of most current programming languages. Spreadsheets embed what is, in practice, a programming paradigm, based on the model of cells and cell variables.

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 (2002) estimate that up to 90% of spreadsheets contain errors. This makes clear the need for tools to improve on the spreadsheet paradigm.

Another problem with spreadsheets is their limitations in terms of the expressive power of their programming paradigm. Even some simple models can be difficult to implement if they do not closely fit 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 expressiveness. To prove this point we implemented a prototype system called LESS (for **L**ogic **E**mbedded in **S**pread **S**heets). The LESS system 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

https://doi.org/10.1017/S0269888907001178 Published online by Cambridge University Press

overcome shortcomings in the traditional spreadsheet approach. We discuss how we can introduce additional user interface elements that are intended to make LESS easy to use and more accessible to the ordinary spreadsheet user. We then briefly discuss the results of our implementation and the insights we gained in the process. Finally, we present our conclusions.

2 LESS architecture: best of breed

We built LESS 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 (MacGregor,1994; Chalupsky *et al.*, 2006). PowerLoom is a logic-based knowledge representation and reasoning (KR&R) system developed over the last 10 years by the Loom KR&R group at the University of Southern California's Information Sciences Institute. PowerLoom is a maturing prototype system that has been used successfully in a variety of research and application prototype programs and has been distributed to over 600 sites worldwide.

PowerLoom is one of the most expressive and powerful KR&R systems available today. First, its representation language is KIF (Knowledge Interchange Format) [Genesereth (1991)], which is more expressive than the languages implemented in currently available KR systems such as description logics, logic programming and frame-based reasoners. Second, its inference engine can handle recursive rules, negation, equality reasoning, subsumption, open and closed-world 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 that is actually needed. Fourth, PowerLoom is available in libraries for C++, Java or CommonLisp which, despite its significant complexity, are quite small. For instance, the C++ library that is being integrated with MS Excel is only about 8 MB. 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 specialized reasoners (e.g. abductive inference, simple temporal reasoning or rule learning), specialized functions (e.g. computing with units of measure) and to integrate PowerLoom with relational databases. All these capabilities greatly facilitate the integration of PowerLoom with spreadsheets.

3 LESS design: four tiers

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 of it is a logic-based KR&R system. A natural trade-off exists 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 is not 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 approach to solve this problem implements four tiers of functional complexity that provide different trade-offs between keeping close to the spreadsheet paradigm and providing more powerful models that exploit logic's capability for deeper reasoning. Some of the tiers imply additional user interfaces, but they are all additive to the existing spreadsheet. The idea is that a user

https://doi.org/10.1017/S0269888907001178 Published online by Cambridge University Press

can explore any of these tiers, more or less like making use of some advanced tools in spreadsheets today while also using the fundamentals. The four tiers are:

Tier 1: Cell-level spreadsheet functions give nearly full access to PowerLoom's ability to exploit concepts, relations, instances, facts and rules, but makes little use of the spreadsheet metaphor to help the user exploit this power. Straightforward assertion of facts and simple rules, augmented by rule-based conditional retrieval and Boolean evaluation of state are within the grasp of non-programmer users at this level of functionality. More complex exploitation of concepts, instances, relations, rules and specialized reasoning such as classification are available at this level, but may still require more sophisticated logic programming skills.

Tier 2: Tables of concepts and relationships exploit the spreadsheet matrix metaphor (and supporting functions) to provide a visual mechanism to more easily build and assert collections of knowledge, to retrieve and view asserted and inferred facts, instances and relationships; and thus to more easily view and understand the state of the knowledge base assertions and inferences. Tier 2 substantially enhances usability for a limited but powerful subset of logic functions (and can be augmented by the 'Power User' who selectively exploits Tier 1 functions).

Tier 3: Custom interfaces and wizards exploit a full integration of spreadsheet and logic functionality. Rules, concept hierarchies and more complex relational structures are made accessible to non-programmers by Tier 3 tools. These tools employ access to the functional features and current content of both the spreadsheet and logic environments, using click-and-choose interfaces to compose constructs that previously required experienced logic programmers.

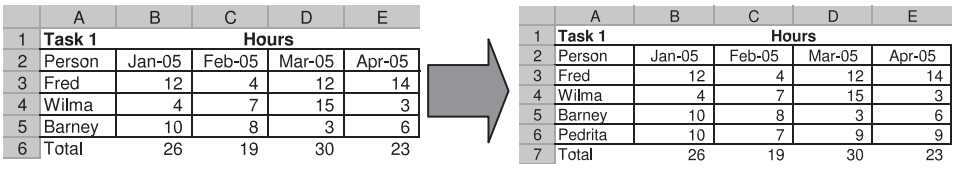
Tier 4: Real-world knowledge has structure; experienced knowledge engineers invest significant effort in engineering a domain model that reflects this inherent structure. Domain-specific ontologies are augmented by defining stereotypical rules whose structure is repeatedly used to capture sets of similar inference rules, and whose structure is designed to 'chain' rules to facilitate efficient evaluation of frequent queries. Combined with spreadsheet-based layouts of tables and supporting calculations, these structures form a template for implementing classes of solutions. These 'Domain Packages' (e.g. a 'Electric Motor Designer's Domain Pack') will provide sophisticated logic/spreadsheet applications ready to be populated with user-specific content and knowledge.

4 A walk through LESS

In this section, example spreadsheets typical of business, military and educational users will be used to illuminate the limitations and issues faced by traditional spreadsheets, and illustrate how the logic-enabled features of LESS mitigate these shortcomings. Key issues include quality: mitigating the inherent spreadsheet difficulties of error-prone programming, maintenance, and change, and enhancing understandability of models through more explicit, declarative modeling of concepts, relations and business rules. Others address the richness of modeling available to users by placing the expressiveness of logic available within their reach while augmenting their ability to build larger, more complex models. The examples presented in this section will focus on LESS Tier 1.

Our examples will include a small business cost proposal spreadsheet using labor hours and pay rates; a classroom grading spreadsheet (which was also used as a class exercise in teaching spreadsheets); and a pilot training spreadsheet based on a spreadsheet used by the Air National Guard's 149th Tactical Fighter Squadron to track mission readiness, identify training requirements and plan training activities.

https://doi.org/10.1017/S0269888907001178 Published online by Cambridge University Press



	A	B	C	D	E
1	**Task 1**		**Hours**		
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

	A	B	C	D	E
1	**Task 1**		**Hours**		
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	Pedrita	10	7	9	9
7	Total	26	19	30	23

Figure 1 Errors are introduced by spreadsheet changes that impact formulaic dependency on layout

4.1 Explicit relationships

The basis for a cost proposal in a service contract is setting how many hours each member of the project team will work on each task. Figure 1 shows a simple table for such a purpose. This table shows a simple and common case of using spreadsheet formulas, where positional referencing is simple and most effective. The cells that contain totals per month in each individual table are calculated using a similar formula—for instance, cell B6 contains the formula = SUM(B3:B5).

However, even this simple example can get users into trouble if they attempt to perform a simple task such as adding a new project member. A user would attempt to introduce the new member by inserting an empty line and adding the values of hours for that person. Note, however, that the formula for the Total stayed the same, which now produces the incorrect results, adding only the hours of the original team members. This problem is common enough that Microsoft Excel normally tries to make an automatic correction[1]. However, if neither the user nor the program is able to correct the problem, the sheet will yield incorrect results.

The root of this problem is that the formula = SUM(B3:B5) is dependent on a table layout which only indirectly reflects the user's intent, which is to add the hours for all project members of the month. When the table dimensions changed, the consistency assumption between the calculation and the layout is violated. LESS allows us to avoid these errors 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 mark symbols denoting variables).

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 of doing this in LESS is to add a lifting formula on top of each column:[2]

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

This expands into the following series of PowerLoom assertions, for example:

```
(hours A3 B2 A1) = (hours Fred Jan-05 "Task 1") = 12 (hours A4 B2 A1) = (hours Wilma Jan-05
"Task 1") = 4 (hours A5 B2 A1) = (hours Barney Jan-05 "Task 1") = 10
```

Once the information is lifted into PowerLoom, we can explicitly specify our desired result ('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")"))
```

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

[1] Excel is able to make this work, for instance, if the new line was inserted in the middle of the table, for example, between lines 3 and 4.

[2] This formula also mentions the range explicitly and as such has similar difficulties than Excel in terms of update management. Tier 2 of LESS manages tables explicitly and would make sure the update would keep working as rows are added.

https://doi.org/10.1017/S0269888907001178 Published online by Cambridge University Press

	A	B	C	D	E
1	**Task 1**		**Hours**		
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 2 Table modeled with LESS

	A	B	C	D	E
1	**Task 1**		**Hours**		
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	**Task 2**		**Hours**		
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	**Task 3**		**Hours**		
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	**Total**		**Hours**		
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 3 Tables for specifying hours in different tasks in a budget spreadsheet

We can further generalize this formula by using cell references to allow automatic adaptation with cut and paste for other months:

$$= \text{SUM (RETRIEVEALL("(hours ?person B\$2 \$A\$1)"))}$$

Note that some of these problems are known to spreadsheet implementors, who in some cases produced mechanisms to try to overcome them. For instance, *lists* in Excel allow a user to mark an area of a spreadsheet to be interpreted and identified as a table. This type of mechanism helps in expressing the boundaries of the table, and avoiding some of the update problems we noted above. However, it still fails to make explicit the relationships expressed in the table contents, or to make it easier to query the table data as a logic approach does (Figure 2).

4.2 Multi-dimensional relations

Spreadsheet-style positional referencing works only when information has at most two independent dimensions. This limitation is frequently encountered in calculating totals from information distributed across tables. For instance, Figure 3 shows a more complex project sheet built as three tables of task-specific data, and a fourth that aggregates numbers across tasks using explicit positional references, for example, $B24 = B3 + B10 + B17$.

https://doi.org/10.1017/S0269888907001178 Published online by Cambridge University Press

The problem is that because the information is not in a single table, we cannot use ranges. Indeed, although such a formula is easy to create for a small number of tasks, it can become cumbersome if there are many tasks[3]. Moreover, it is difficult to maintain, and it is error-prone—for instance, if a new task is added, all aggregation cells must be re-specified; as in the previous section, the problem is that the formula B24 = B3 + B10 + B17 does not very well represent the intent of the user, which is to add the hours Fred worked in all tasks in Jan-05 (or, more generally, to add the hours worked by the project member specified in cell A24). As we can see, while the data in 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[4].

LESS can solve these limitations by resorting to relational referencing—indeed, the approach to model this problem is 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 be turned into a more general, spreadsheet-friendly formula by using cell variables instead of data:

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

4.3 Better, explicit rules

The main use of spreadsheets is to make calculations based on models of some business process or policy. Invariably, such models will involve one or more *business rules*, for instance, rules that state how to determine hours of vacation, or how to use indirect cost rates.

When creating a spreadsheet, business rules end up being encoded in spreadsheet formulas. This has several problems. First, rules encoded in formulas are hidden from view, which makes them opaque and harder to audit (and, thus, can easily lead to errors). Second, because spreadsheet formulas are repeated in different cells, business rules have to be repeated everywhere they are used. Again, this increases the possibility for errors, and makes the spreadsheet difficult to maintain. A modification in the business rule would require tracking all formulas where it was used, and then modifying each one accordingly. A third problem is that the spreadsheet language is very poor in expressing rules that are not strictly mathematical formulas. For example, there are no good mechanisms for complex conditions, which end up expressed as nested if—then formulas.

LESS improves on plain spreadsheets by allowing a user to model rules such as business rules explicitly. This is a better methodology (following a trend toward explicit business rules), makes the model easier to debug and maintain, improves visibility and facilitates communication and collaboration in developing or using a spreadsheet-implemented model.

Let us examine an example from a grading spreadsheet. Figure 4 shows a simplified calculation of grades for a fictitious class. The letter grade calculated in column H depends on the range in which the numeric grade in column G lies. For instance, if a numeric grade is lower than 70, the letter grade is D; if the numeric grade is at least 70 but lower than 75, the letter grade is C-,

[3] In fact, the real-life sheet from where this example was abstracted contained 15 task tables with up to 20 people each.

[4] Some spreadsheet implementations like Excel provide a mechanism called "pivot tables" that can handle some of these cases, but it also has limitations (e.g., it is limited to arity = 3). A really multidimensional spreadsheet (e.g. where the information was laid out in a cube instead of a plane) would extend the number of dimensions which the positional approach can handle well, but would introduce obvious usability problems.

https://doi.org/10.1017/S0269888907001178 Published online by Cambridge University Press

	A	B	C	D	E	F	G	H
1		Assignments		Tests		Exam	Final	Letter
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 4 Calculating grades for students in a class

	A	B	C	D	E	F	G
11	Classification rules - Version 1						
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 5 Classification rules version 1: explicit, individual business rules

etc. Spreadsheets like Excel allow such rules to be modeled via cascaded conditionals or 'cascaded ifs'. For example, a typical formula for cell H3 would look like this:

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 error-prone. Second, it is difficult 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 upper bound for a C-grade). This makes it difficult to perform consistent grade adjustments by modifying the ranges for each letter grade—a common practical need in real-life classes.

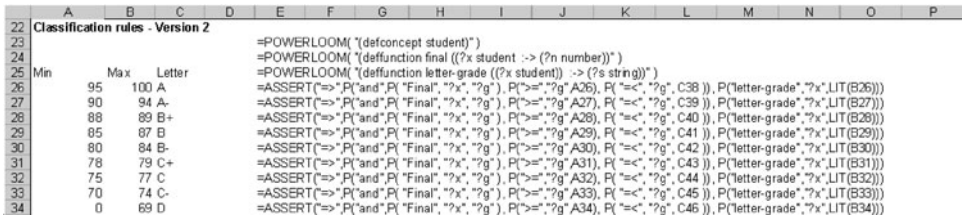
In LESS we improve on the spreadsheet model by allowing users to define rules explicitly and separately, and simply apply them when needed. Once the rules are defined, the formula to calculate the letter grades in this example (column H) becomes a simple query:

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

In LESS the required rules can be defined in several different ways. The first shown in Figure 5 is to write them with conditional antecedent and consequence terms using hardcoded range boundaries. This is easy for simple rules whose form repeats; users can 'copy' the rule cells to re-use the rule structure and then edit each rule appropriately. There is only one rule for each letter grade, which makes the overall rule set clearer and more modular for small rule sets. Editing a rule would automatically adjust all the grade sheets where the rule was used via a query as the one shown above.

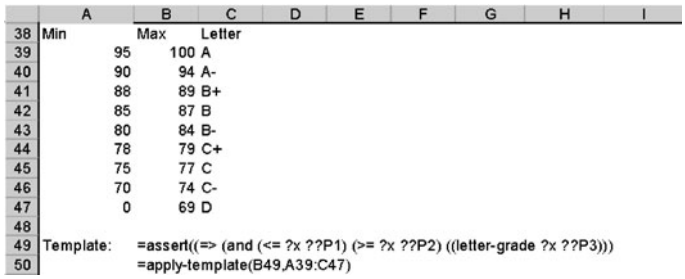
A more flexible modeling style—and one that fits nicely with the spreadsheet paradigm—is to generalize rules by using Excel cell references to parameterize the inference. A formulation along these lines is shown in Figure 6. We make use of a separate small table to represent the upper and lower bounds of each grade range and its associated letter grade (columns A, B and C). Now it suffices to write a single parametric rule and using Excel's 'copy' function to copy it to other rows

https://doi.org/10.1017/S0269888907001178 Published online by Cambridge University Press



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
22	**Classification rules - Version 2**															
23					=POWERLOOM("(defconcept student)")											
24					=POWERLOOM("(deffunction final ((?x student :-> (?n number))")											
25	Min		Max	Letter	=POWERLOOM("(deffunction letter-grade ((?x student)) :-> (?s string))")											
26		95	100	A	=ASSERT("=>",P("and",P("Final", "?x", "?g"), P(">=","?g",A26), P("=<", "?g", C38)), P("letter-grade","?x",LIT(B26)))											
27		90	94	A-	=ASSERT("=>",P("and",P("Final", "?x", "?g"), P(">=","?g",A27), P("=<", "?g", C39)), P("letter-grade","?x",LIT(B27)))											
28		88	89	B+	=ASSERT("=>",P("and",P("Final", "?x", "?g"), P(">=","?g",A28), P("=<", "?g", C40)), P("letter-grade","?x",LIT(B28)))											
29		85	87	B	=ASSERT("=>",P("and",P("Final", "?x", "?g"), P(">=","?g",A29), P("=<", "?g", C41)), P("letter-grade","?x",LIT(B29)))											
30		80	84	B-	=ASSERT("=>",P("and",P("Final", "?x", "?g"), P(">=","?g",A30), P("=<", "?g", C42)), P("letter-grade","?x",LIT(B30)))											
31		78	79	C+	=ASSERT("=>",P("and",P("Final", "?x", "?g"), P(">=","?g",A31), P("=<", "?g", C43)), P("letter-grade","?x",LIT(B31)))											
32		75	77	C	=ASSERT("=>",P("and",P("Final", "?x", "?g"), P(">=","?g",A32), P("=<", "?g", C44)), P("letter-grade","?x",LIT(B32)))											
33		70	74	C-	=ASSERT("=>",P("and",P("Final", "?x", "?g"), P(">=","?g",A33), P("=<", "?g", C45)), P("letter-grade","?x",LIT(B33)))											
34		0	69	D	=ASSERT("=>",P("and",P("Final", "?x", "?g"), P(">=","?g",A34), P("=<", "?g", C46)), P("letter-grade","?x",LIT(B34)))											

Figure 6 Classification rules version 2: parameterized business rules



	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 7 Classification rules version 3: defining rules using templates

(E27 to E34) where it automatically picks up the appropriate values from the grade definition table.

A final advantage of this approach is that it makes it easier to perform tasks that require iterative modification of rules, such as a simple manual form of 'curving' the grades. This involves changing the letter grade ranges to ensure that a certain number or percentage of students get a certain grade (e.g. only 10% of the class can get an A)[5]. These requirements are common in college departments, and can be seen as external rules (indeed, meta-rules). We can even represent these meta-rules explicitly (not shown) to calculate whether or not they are being obeyed by the actual grades given.

A third formulation that can provide an even more elegant solution is to use templates (which will be implemented in Phase II of the project). This is shown in Figure 7. Here, the rule template is defined based on abstract (non-cell) variables, and a separate formula indicates how to bind those logic variables to cells. As a result, we can formulate the rules with only two cells, achieving a maximum of flexibility and maintainability.

4.4 Template-based knowledge acquisition

A common problem in knowledge acquisition is that knowledge is modeled in very different ways by a subject-matter expert and by a knowledge engineer. Because of the precision requirements imposed by reasoning engines, each 'link' between two pieces of information has to be modeled in detail, usually depending on the type of reasoning needed. In contrast, subject-matter experts are able to provide concise information that is imprecise with respect to the relationships between the different pieces of information entered.

In a real-life example, Valente *et al.* (1999) developed an ontology of military aircraft as part of their work in the the Defense Advanced Research Projects Agency (DARPA) ARPI and JFACC Programs. As part of the knowledge acquisition process, the knowledge engineers would frequently receive textual or semi-structured information that would be expressed in tabular form.

[5] More complex forms of curving require statistical analysis.

https://doi.org/10.1017/S0269888907001178 Published online by Cambridge University Press

	A	C	D	F	G	H
1	**Aircraft**	**Maker**	**# Engines**	**Primary Fuel**	**Alternate Fuels**	**...**
2	F-15	McDonnel Douglas	2	JP-4	JP-5 JP-8 Jet-A Jet-A-1 Jet-B	...
3	F-16	General Dynamics	1	JP-4	JP-5 JP-8 Jet-A Jet-A-1 Jet-B	...
4	...	...	...	...	...	...

Figure 8 Information about aircraft specified by subject-matter experts

For example, subject-matter experts would produce a spreadsheet with properties of different aircraft that would look like the following:

Information provided this way would be translated into complex Loom definitions (the logic formalism used at the time—similar definitions can be done using PowerLoom), for example (somewhat simplified):

```
(defconcept F-15:
    is-primitive Fighter-Aircraft:
    implies (:and (:filled-by made-by McDonnell-Douglas)
                  (:all gun M61A1)
                  (:filled-by primary-fuel JP-4)
                  (:filled-by alternate-fuel JP-5 JP-8 Jet-A Jet-A-1 Jet-B))
    :defaults (:and (:filled-by ceiling 65000ft)
                  (:filled-by crew-size 1))
...)
```

Due to the complexity of the definition language, we created simpler *definition templates*, implemented as Lisp macros, which made it easier and simpler to enter the desired information. The advantage of these templates was that they abstracted the way information was used in the logic formalism. Instead, the code interpreting the template decided whether information was to be used as a slot filler, a type restriction, a default value, etc. A template would look like the following:

```
(def-aircraft-c F-15
    :types Fighter-Aircraft
    :made-by McDonnell-Douglas
    :default-ceiling 65000ft
    :default-crew-size 1
    :gun-type M61A1
    :primary-fuel JP-4
    :alternate-fuel (JP-5 JP-8 Jet-A Jet-A-1 Jet-B)
...)
```

Unfortunately, these templates were still hard for subject-matter experts to manipulate directly due to the use of parentheses, etc. With LESS we have an opportunity to further bridge this gap by allowing the use of spreadsheet tables as the vehicle for information input, which is the natural way in which domain experts expressed their knowledge. The provided information can then be lifted into a formal knowledge base by knowledge engineers directly in the spreadsheet. For instance, we can represent the template above based on the spreadsheet tables in Figure 8 using LESS templates as follows:

```
P2 = (defconcept ?v1 ((?x Fighter-Aircraft))
    :=> (AND (made-by ?v2)
        (= number-of-engines ?v3)
        (primary-fuel ?v4)
        (alternate-fuel ?v5)))
```

https://doi.org/10.1017/S0269888907001178 Published online by Cambridge University Press

	A	C	D	F	G	H
1	Aircraft	Maker	# Engines	Primary Fuel	Alternate Fuels	...
2	F-15	McDonnel Douglas	2	JP-4	JP-5	...
3					JP-8	...
4					Jet-A	...
5					Jet-A-1	...
6					Jet-B	...
7	F-16	General Dynamics	1	JP-4	JP-5	...
8					JP-8	...
9	...	...	...	...	...	...

Figure 9 Modified table with multiple values split into separate rows

This template can then be directly applied to the table in Figure 8 with a single formula (the first argument is the cell where the template is written and the second the range of cells where it is to be applied; here, notionally, the whole table with several aircraft):

$$\text{P3} = \text{APPLY-TEMPLATE(P2,A2:O35)}$$

An interesting issue in that kind of application in practice is that users tend to combine several values into a single cell (e.g. the several alternate fuel types in the table in Figure 8). This makes things a bit harder to parse, since spaces do not necessarily delineate value boundaries. One solution would be to adopt a special notational convention for such values, another is to use a sparse table where values without a new 'head' entry are interpreted as additional values for a slot. For example, the table shown earlier can be modified using the method described, resulting in the table shown in Figure 9.

4.5 Powerful information querying

Spreadsheets have several limitations on how to summarize information, particularly if the information is symbolic. The basic problem is that it is hard to select information from other parts of the spreadsheet. The problem becomes worse if information is in multiple tables and/or has more than three dimensions. Special wizards such as Excel pivot tables help overcome some of these deficiencies, but they still have restricted expressivity.

In this section, we present a very detailed example centered around an Air National Guard unit's need for tracking the training and certifications of its pilots. The example will show how we provide information synthesis and querying. It will also show how LESS can use logic reasoning to represent hierarchical (subsumption) relationships, and reiterate capabilities for using logic rules.

We will first examine information assertion and retrieval in LESS. In Figure 10, we show the Roster worksheet which is used for entering information about pilots into the knowledge base. The user may add pilots by name and callsign, and LESS asserts the appropriate facts into the knowledge base. Daily information about each pilot may be asserted on the Daily Template, which is copied for each day (not shown). The user can enter numbers in the body of the table, and these are interpreted and asserted as facts in the knowledge base. The information entered on each day's daily sheet can be retrieved and accumulated, such as on the Pilot Report By Skill sheet (also in Figure 10). The dates across the top are retrieved from the knowledge base, as well as the missions and codes, and the contents of the table.

Next (Figure 11), we will consider logic rules, and how they can be used. The Skills sheet is similar to the Roster sheet, in that it allows the user to enter a list of items, which then get asserted in the knowledge base—in this case, the mission skills that the pilots receive certifications for. In one column of this sheet, the user may enter a PowerLoom rule representing the conditions under which a pilot is considered certified for that skill on that date (in the case shown, if he has more than 10 sorties of that type within the last 90 days). On the Pilot Planner sheet, the user may plan out sorties for individual pilots by seeing which skills have expiring certifications. This uses the LESS ASK( ) macro to simply check certification for the given skill (row) on the given date (column). This is nested in the Excel IF macro, and Excel is used to set the background color.

The last phase of the example (Figure 12) illustrates hierarchical representation in PowerLoom and LESS. The Categories sheet is another information entry sheet like Roster and Skills—in this

https://doi.org/10.1017/S0269888907001178 Published online by Cambridge University Press

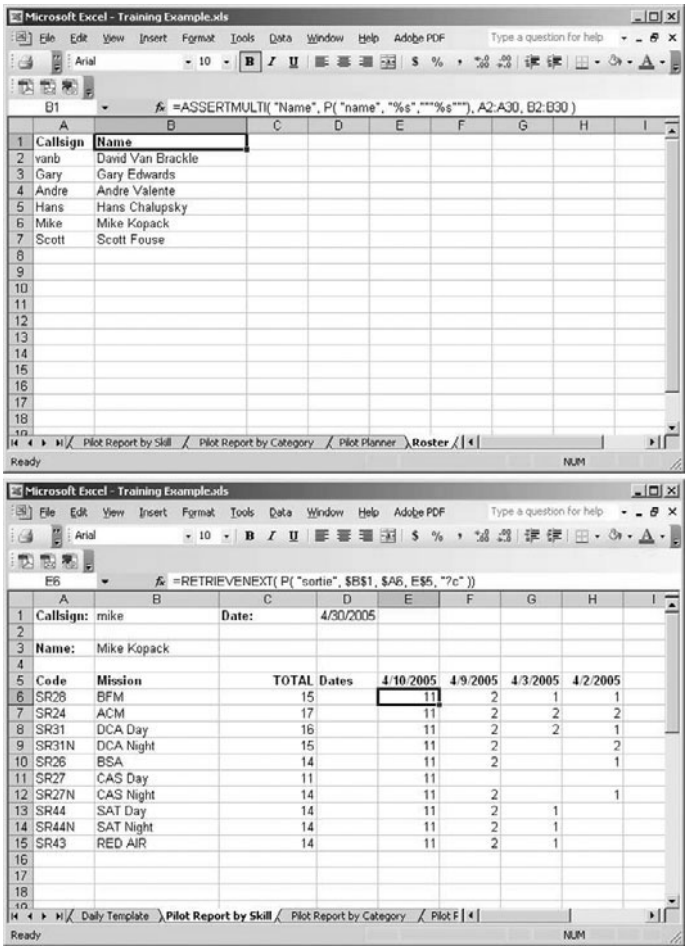


Figure 10 Entering information into the knowledge base and creating reports using LESS

case, entering categories of skills and their parents. The PLOOM() macro allows the use of the PowerLoom defconcept command to define the concept and its parent. The final sheet, Pilot Report By Category, can list all of the skills in a given category that a pilot is not certified for. It uses the hierarchical relationships established on the Categories sheet. For example, DCA is a subcategory of RAP; when retrieving all relevant RAP missions, DCA missions are automatically, implicitly included, without any need for explicit specification.

4.6 Reasoning

A final way in which logic spreadsheets can improve over plain spreadsheets is that they can contain full-blown knowledge-based systems. Using spreadsheets as a vehicle for deploying knowledge-based systems has the advantage that information is usually already available in spreadsheets, and users find it easy to enter new information in spreadsheets.

For instance, it would be possible to re-implement a system like PROSPECT (Valente and Scacchi; 1999) using LESS. PROSPECT is a knowledge-based system that critiques business processes. PROSPECT's knowledge base contains a list of types of mistakes and problems in process models. The system was used to critique processes from the accounting department from a large corporation in Southern California. The input to the system is a list of (instances of) business process models; the output is an agenda of problems to evaluate.

An implementation of PROSPECT in LESS can use tables and templates to express input process models, which is a natural representation that was indeed used in practice (the users were business people). The output agenda of problems in the process models can be implemented

https://doi.org/10.1017/S0269888907001178 Published online by Cambridge University Press

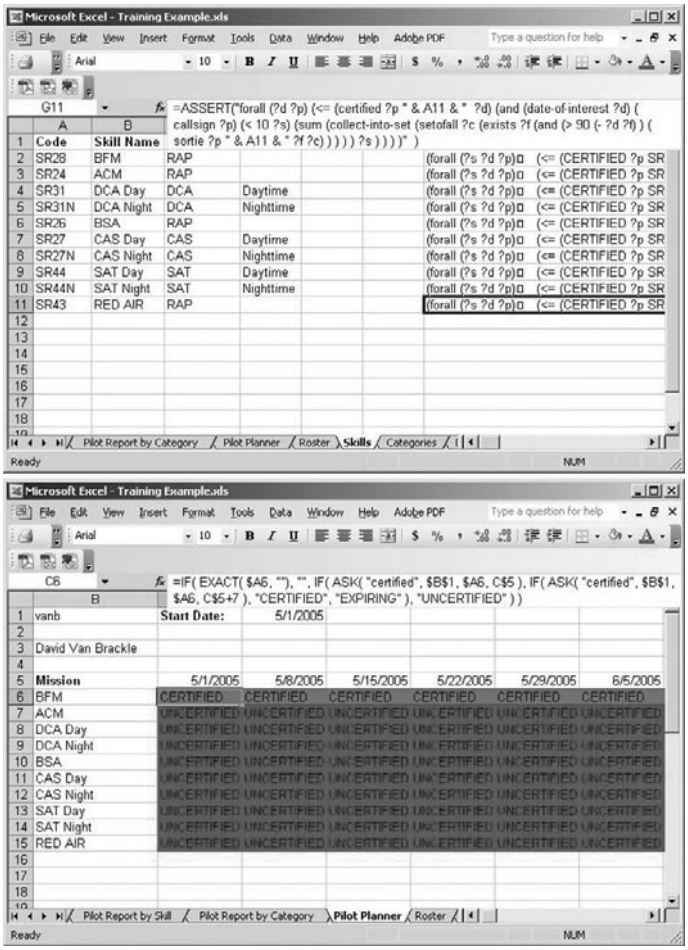


Figure 11 Using logic rules for reasoning and querying in LESS

as an instance table (Tier 3), and the knowledge base can be provided explicitly or as a domain pak (Tier 4).

5 Making LESS easier: higher tiers

All of the functionality discussed in the previous session was shown as used in Tier 1, that is, by directly creating formulas that call the logic engine. The idea of the higher tiers in LESS is to introduce additional user interfaces that make it easier to employ that underlying power without forcing the user to know too much about logic reasoning. These user interfaces were developed using Microsoft Visual Basic for Applications, which allows a tighter integration with Excel's inner structures.

5.1 Tier 2

An initial implementation of Tier 2 includes functionality such as adding a class and adding an instance, and changing instance property values. For example, a menu item was added to the **File** menu: **Add Instance** Table. When chosen, this action would invoke a pop-up dialog, which allows the user to enter a name of a class, and the names of several properties. If the user clicks OK, a new worksheet is added, and given the same name as the user chose for the class. The created worksheet is specially tailored to its task. The first row holds the names of the properties as column headers, and the first column holds computer-generated unique Ids for the

https://doi.org/10.1017/S0269888907001178 Published online by Cambridge University Press

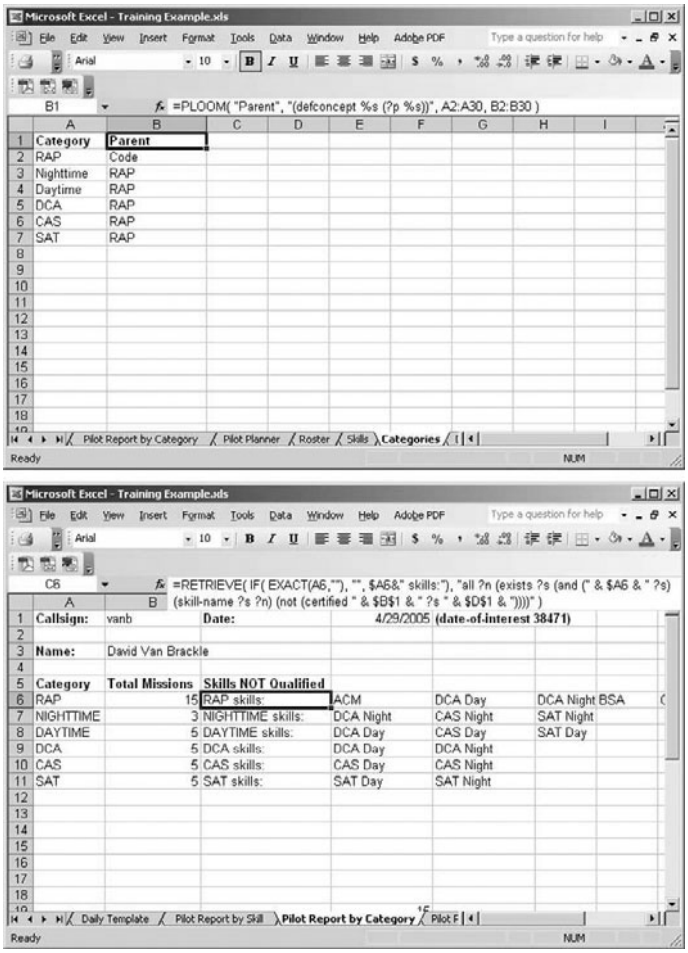


Figure 12 Representing and reasoning with class hierarchies in LESS

instances. Adding a row means adding an instance, and a unique ID is automatically generated. The user may not move the cursor except into editable fields.

Below is an example of using Tier 2. We will employ the pilot training spreadsheet example discussed in the previous section. Using Tier 2 functionality, a user can create a **Pilot** table, with fields such as **CallSign** and **Platform** (**Platform** represents the type of plane that pilot flies), and populate it with a few instances. Further, a user can create a **Mechanic** table, with fields such as **Name** and **Primary** (**Primary** represents the primary type of plane that mechanic works on, and has the same kinds of values as **Platform** for a **Pilot**), and populate it with a few instances. When the sheets are created, a user can easily produce sophisticated queries. For example, one can query for the names of all mechanics who work on the same plane as flown by a specified pilot. The results of the query are automatically updated when the user changes the pilot's callsign in a spreadsheet cell. The query is as follows, assuming that the pilot's callsign is in cell A1, and allowing for linebreaks to make the code clearer:

```
= RETRIEVEALL( "all ?n (exists ?p ?m ?x (and
    (pilot ?p)
    (callsign ?p " \& A1 \& " )
    (mechanic ?m)
    (name ?m ?n)
    (platform ?p ?x)
    (primary ?m ?x)))" )
```

https://doi.org/10.1017/S0269888907001178 Published online by Cambridge University Press

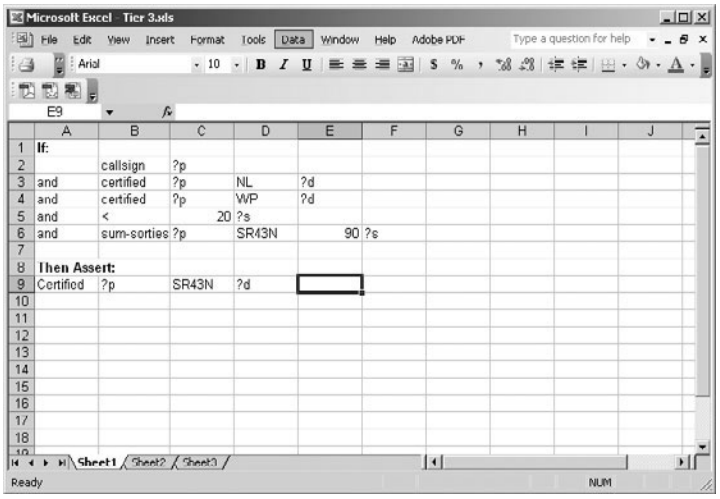


Figure 13 Mockup of a Tier 3 tool to create rules that uses the physical positioning of cells in the spreadsheet

The idea in Tier 2 is that a query is associated automatically with a table (in the current version, one table per worksheet) so that on the top you write the query and the table is automatically populated with the results. Each line of the table has one solution for the query, and the columns of the table represent the variables in the query (in the example: ?n ?p ?m ?x). Note that the user cannot edit the table, only access the contents by using the usual cell references. This is intentional; we take away control but try to make querying easier.

5.2 Tier 3

For Tier 3, two different paradigms of rule entry were prototyped: a *spreadsheet mockup* and an *active templates mockup*. In each case, the strength and weaknesses of the proposed approach became apparent.

5.2.1 Spreadsheet mockup

In the Spreadsheet mockup (displayed in Figure 13), we used the structure of the spreadsheet to represent rules. There are two sections: the IF clause, and the THEN clause. In the IF clause, each row represents a fact that must be in evidence for the rule to fire. The first column has a pull-down menu with a conjunction, one of AND, OR, AND NOT, OR NOT. The rest of the columns hold contents of the tuples. In the THEN clause, the rows simply represent facts to be asserted if the IF clause is true.

This methodology presents a spreadsheet-based representation of a rule which is easy to understand and manipulate. Rules can be entered within the context of the spreadsheet itself, which is a powerful advantage. It has disadvantages, however. The split between IF and THEN clauses is arbitrary, and will certainly need to be moved in some cases. This can be accomplished simply by adding rows to the IF clause, but this can still seem clumsy. This representation also puts specific requirements on the interpretation of the spreadsheet, which may be difficult to maintain. Its most negative aspect, however, is that it handles hierarchy poorly, if at all. Nested logical conditions are very difficult to represent; they must be flattened out to a single level. Nested facts are likewise impossible to represent.

5.2.2 Active templates jumpstart mockup

This mockup is based on earlier work on the Defense Advanced Research Projects Agency (DARPA) Active Templates project, which built spreadsheet-like tools to manage and process

https://doi.org/10.1017/S0269888907001178 Published online by Cambridge University Press

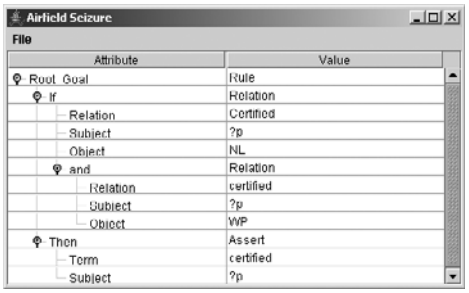


Figure 14 Mockup of a Tier 3 tool to create rules that uses a separate graphical, tree-based widget

information. Specifically, some of the authors built an experimental lightweight planning system based on a spreadsheet GUI paradigm.

In this mockup, displayed in Figure 14, rules are represented using a TreeTable user interface widget, with two columns: Attribute and Value.

This methodology represents hierarchy well, even allowing large sections to be exposed or hidden. The specifics of the Active Templates solution are, however, ill-suited to the problem at hand. Adding a clause is somewhat clumsy, and the two-column paradigm does not handle higher arity tuples well.

Although neither of these experiments yielded a final solution, ideas from both could be used to create a worthwhile GUI. Combining the tree view of the ATJ mockup with the multi-column table of the Spreadsheet mockup could yield a serviceable solution. Another possibility is to abandon the graphical paradigm, and build a syntax-directed or language-assisted editor for a simple rule specification language.

5.3 Tier 4

The entire Tier 1 demonstration serves as an example of a Tier 4 mockup. It could be a Domain Pak, built by experts and distributed to all Air National Guard (ANG) units to monitor their training and certifications. LESS would need to provide ways of making this example easier to deploy, such as

- Easy ingest—a way to acquire the names of pilots from the ANG existing source, perhaps a database or spreadsheet
- Protection of logic—a way to protect key cells from accidental modification by end users
- Better persistence mechanism—The knowledge base in LESS, as it stands, maintains all of its knowledge persistently. It would need to be more selective, and allow the end user to partition knowledge of one application from that of another.

In addition, tools would need to be built to help an expert author a Domain Pak.

6 Implementing LESS

The LESS system was implemented as an Excel add-in that includes the PowerLoom C++ libraries. The whole file takes less than 4 MB, and it is quick enough that we have not noticeable slowed 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, the existing application programming interfaces (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.

https://doi.org/10.1017/S0269888907001178 Published online by Cambridge University Press

PowerLoom's ease of integration was also extremely important in allowing us to experiment with different ideas.

An interesting (and still open) challenge was the synchronization between the data in the spreadsheet and its corresponding (lifted) PowerLoom knowledge base. A logic engine typically has facts asserted, and queries requested, in a fairly ordered sequence. In contrast, a spreadsheet refreshes itself frequently, often re-evaluating the same macros multiple times, and with the only promised order being explicitly stated in cell relationships. This can lead to several difficulties in interfacing. For example, queries usually do not explicitly reference the cells that compose their results, and as a result it is difficult to determine when to re-evaluate a query based on changes in the cells that would constitute its result. The current prototype implements a simple strategy in which logic formulas are reevaluated frequently and repeatedly. Although this has not yet caused serious performance problems (PowerLoom is reasonably smart in recognizing repeated assertions), an effective solution requires the development of tailored mechanisms to track dependencies between elements in the knowledge base and elements of the spreadsheet so that we can control in finer grain what to re-evaluate and what/when to synchronize.

7 Related work

The idea of marrying spreadsheets and logic is not really new. For example, Spenke and Beilken (1989) produced an early implementation of a logic spreadsheet much like LESS on a Symbolics Lisp machine.

The interest in spreadsheets and their problems has increased 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 [e.g. Erwig and Burnett (2002)] was dedicated to the identification of spreadsheets as a form of programming and some of its shortcomings. Later, Abraham & Erwig (2004) 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. A similar strategy was used by Paine (2004), who developed a system that uses logic to discover the underlying structure of a model represented in a spreadsheet. Note that logic here is not actually integrated in the spreadsheet, but instead used at the meta-level to reason about the spreadsheet. This mirrors one of the strategies we intend to employ in developing Tiers 2 and 3, namely to create a knowledge base about how models are laid out in spreadsheets that can be used intelligently to discover the underlying structure of an instance spreadsheet.

The latest work Erwig's group is Gencel (Erwig *et al.*, 2005), an extension to Excel based on the concept of a spreadsheet template. Gencel templates avoid a number of problems such as reference, range and type errors. Gencel templates have many similarities to Tier 2 instance tables, but they have additional functionality we have not developed so far. It is an interesting (and open) question how easily we can replicate that additional functionality in LESS.

Some similarities exist between the way we lift Excel information into n-arity relations and the Query by Excel application developed by a group at Oracle (Witkowski *et al.* 2003). A key difference is that the information once lifted is interpreted through relational database queries, while we use first-order logic.

Of course, the closest related work to the one presented here is the implementations of logic spreadsheets discussed in the Workshop on Logical Spreadsheets (WOLS'05)—and in this special issue of KER. The differences between these implementations and LESS are discussed in detail in the introductory article to this special issue.

8 Conclusions

The LESS system 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

https://doi.org/10.1017/S0269888907001178 Published online by Cambridge University Press

spreadsheets, as well as open the door to additional applications for which spreadsheets alone are not well-suited.

Designing LESS posed many significant challenges. Spreadsheets and Logic each have their own language, with quite different syntactic conventions. This is perfectly natural—they were designed for very different purposes. The solution we used for treating logic statements and logic API calls (PowerLoom-specific syntax) as string parameters to LESS functions which are presented as spreadsheet macros in Excel proved functional, and made available the full strength of PowerLoom to users well-versed in both domains; however, this convention is not necessarily easy for a novice. Products in higher tiers, such as rule-construction wizards, will help mitigate this, but will not be suitable for highly complex constructs. A carefully designed combined syntax that balances both concerns remains a key challenge for logic spreadsheets.

Acknowledgements

The research reported here was sponsored by the Active Templates, the DARPA Small Business Innovation Research program and Dr. David Gunning of the DARPA IPTO office. Distribution Statement 'A' (Approved for Public Release, Distributed Unlimited).

References

Abraham, R. and Erwig, M. 2004 Header and unit inference for spreadsheets through spatial analyses. In *IEEE International Symposium on Visual Languages and Human-Centric Computing*, pp. 165–172.

Chalupsky, H., MacGregor, R. and Russ, T. 2006 *PowerLoom Manual*, USC Information Sciences Institute. Available at `http://www.isi.edu/isd/LOOM/PowerLoom`.

Erwig, M., Abraham, R., Cooperstein, I. and Kollmansberger, S. 2005 Automatic generation and maintenance of correct spreadsheets. In *27th IEEE International Conference on Software Engineering*, pp. 136–145.

Erwig, M. and Burnett, M. 2002 Adding apples and oranges. *Practical Aspects of Declarative Languages, 4th International Symposium, PADL 2002*, Vol. 2257 of LNCS, Springer, pp. 173–191.

Genesereth, M. 1991 Knowledge interchange format. In J. Allen, R. Fikes and E. Sandewall (eds.), *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, Cambridge, MA, pp. 599–600.

MacGregor, R. 1994 A description classifier for the predicate calculus. In Proceedings of the Twelfth National Conference on Artificial Intelligence, pp. 213–220.

Paine, J. 2004 Spreadsheet structure discovery with logic programming. In *Proceedings of EuSpRIG 2004*, Klagenfurt, Austria.

Spenke, M. and Beilken, C. 1989 A spreadsheet interface for logic programming In K. Bice and C. Lewis, (eds.), *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Wings For the Mind (CHI '89)*, ACM Press, New York, NY, pp. 75–80.

Valente, A., Russ, T., MacGregor, R. and Swartout, W. 1999 Building and (re)using an ontology of air campaign planning. *IEEE Intelligent Systems*, **14**(1): 27–36.

Valente, A. and Scacchi, W. 1999 Developing a knowledge web for business process redesign, In *Proceedings of the IJCAI'99 Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business*.

Witkowski, A., Bellamkonda, S., Bozkaya, T., Dorman, G., Folkert, N., Gupta, A., Shen, L. and Subramanian, S. (2003). Spreadsheets in rdbms for olap. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, ACM Press, pp. 52–63.

https://doi.org/10.1017/S0269888907001178 Published online by Cambridge University Press