Enhancing Excel Business Tools with Additional Relational and Recursive Capabilities

Pedro Ramos[1*], Luís Botelho[2,] António Martins[3]

[1] pedro.ramos@iscte-iul.pt; Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR-IUL, IT-IUL, Lisboa, Portugal

[2] luis.botelho@iscte-iul.pt; Instituto Universitário de Lisboa (ISCTE-IUL), IT-IUL, Lisboa, Portugal

[3] antonio.martins@iscte-iul.pt; Instituto Universitário de Lisboa (ISCTE-IUL), Lisboa, Portugal

[*] Corresponding author. Postal address: Av. ª das Forças Armadas, 1649-026 Lisboa. Phone Nr.: +351 217 903 000.

Abstract

This paper presents a new plug-in that enriches spreadsheet capabilities mainly in what concerns its potential regarding relational queries and recursive computational processes. Currently some apparently trivial and useful queries can only be handled with the support of programming skills. Spreadsheet users with low computer science skills should have a natural and easy way to handle those queries within the spreadsheet, without relying on external programming (e.g., VBA). The tool we have developed can be used with Prolog technology, and provides those features to the most used professional spreadsheet: Microsoft Excel. Throughout the paper we explore the plug-in features with several business examples.

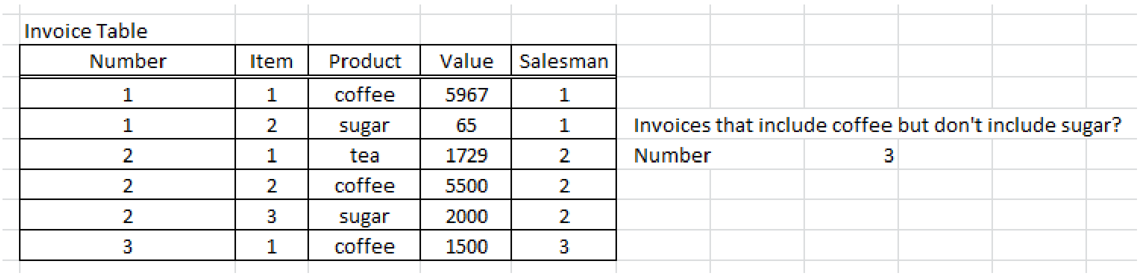
Keywords: Dedutive Spreadsheet, Business Tools, Excel, Prolog, Recursive Processing, Relational Data

1 Introduction

This paper presents a new plug-in that enriches spreadsheet capabilities mainly in what concerns its potential regarding relational queries and recursive computational processes. As we will discuss later, currently some apparently trivial and useful queries can only be performed with the support of programming skills. Spreadsheet users with low computer science skills should have a natural and easy way to perform those queries within the spreadsheet itself. The tool we have developed provides the most used professional spreadsheet, Microsoft Excel, with those features.

The paper presents examples of simple problems that can't be easily addressed using the usual Microsoft Excel resources. We consider different technological solutions for the exemplified classes of problems, namely Relational Database/SQL-based technology, and Prolog-based technology. While SQL-based technology would solve some of the presented problems, it can't solve them all. We then describe a plug-in for Microsoft Excel, through which MS Excel users can solve all the discussed classes of problems using Prolog technology integrated into the spreadsheet.

Microsoft Excel has powerful capabilities, which are strongly valued and used both by individuals and in every company for a large diversity of tasks. However, like all software tools, it has some limitations. One of those limitations pertains to relational queries, for example, queries whose filtering criteria concern more than one row in a table. When a user works with a table with several columns and rows, it is relatively simple to filter the rows using criteria whose scope is the row itself. For example, it is simple to filter only the invoices that sell a specific product (assuming that the product is one of the table columns). However, since one invoice can include more than one item, the user can be interested in retrieving invoices for a specific product that do not include another specific one (to assess the success of specific selling policies). It is not possible to ask that kind of query easily using Microsoft Excel features. Figure 1 illustrates the example (the expected answer is Invoice Number 3).


Invoice Table

Number	Item	Product	Value	Salesman
1	1	coffee	5967	1
1	2	sugar	65	1
2	1	tea	1729	2
2	2	coffee	5500	2
2	3	sugar	2000	2
3	1	coffee	1500	3

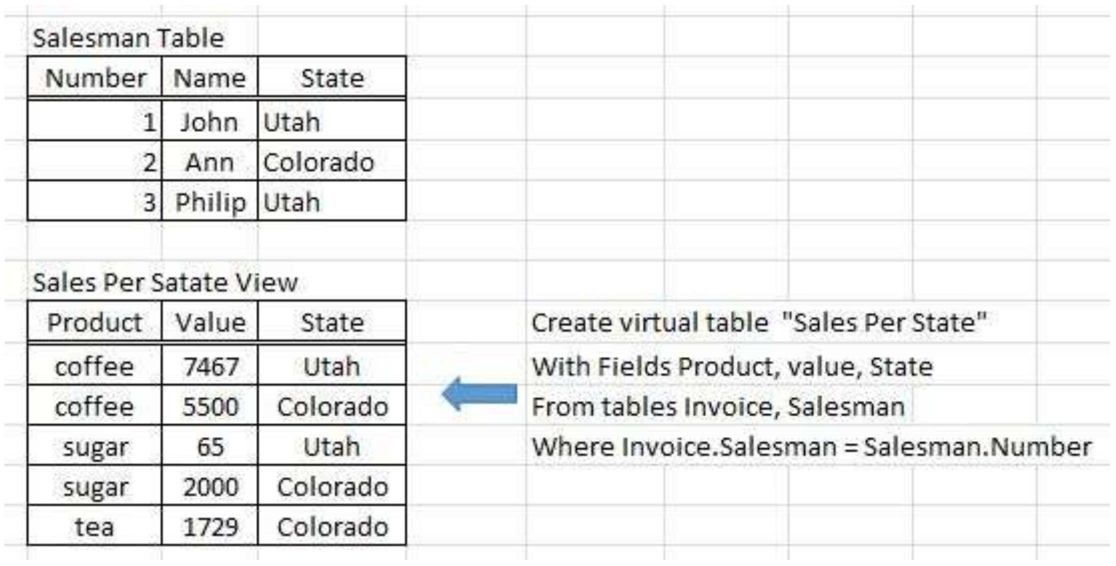
Invoices that include coffee but don't include sugar?

Number	3

Figure 1 - Example of a query whose criteria relate to more than one row

Another situation not covered in a satisfactory way by Microsoft Excel involves queries that need to relate data stored in more than one table. In order to deal with those kinds of queries users usually need to create a new table that combines data from the "base" tables. That solution is laborious and entails the creation of redundant data in the spreadsheet (with all the well-known associated disadvantages). In relational databases it is possible to create virtual tables (called Views) that combine data from several tables. Those tables are virtual in a sense that they do not actually store the information, but only its "signature" (the table name, the columns name, and the query that retrieves the data). However, users can retrieve data from the virtual table exactly the way they do with any other table. Microsoft Excel does not allow the definition of virtual tables.

In Figure 2 we illustrate the desired feature (i.e., creating a virtual table), which is not possible in Microsoft Excel.



Salesman Table

Number	Name	State
1	John	Utah
2	Ann	Colorado
3	Philip	Utah

Sales Per Satate View

Product	Value	State
coffee	7467	Utah
coffee	5500	Colorado
sugar	65	Utah
sugar	2000	Colorado
tea	1729	Colorado

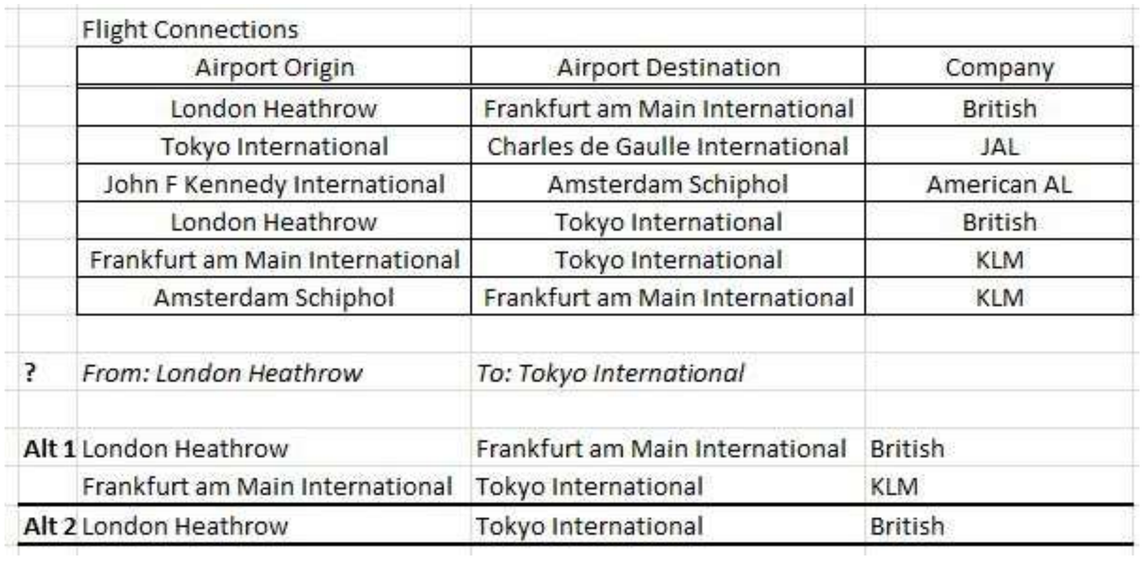
Create virtual table "Sales Per State"
With Fields Product, value, State
From tables Invoice, Salesman
Where Invoice.Salesman = Salesman.Number

Figure 2 – An example of a virtual table

Microsoft Excel does also not allow recursive processing, but recursive reasoning if often needed in data analysis.

Let us suppose that a certain user creates a table with direct flights connecting two airports: the origin and the destination airports. A trivial problem is to determine if there is a sequence of one or more flights connecting two airports (even if there is no direct flight between the two). A simple solution to this problem may be defined, using a banal example of recursive processing. In fact, it is possible to go from airport 1 to airport 2, if there is a direct flight between them, or if there is a direct flight from airport 1 to any other airport, say airport 3, and if it is possible to go from airport 3 to airport 2. This kind of reasoning is called recursive because the solution to the original problem (i.e., determining if "it is possible to go from airport 1 to airport 2") involves finding the solution to a similar problem (i.e., determining if "it is possible to go from airport 3 to airport 2"). Although simple and intuitive, this kind of solution cannot be implemented in Microsoft Excel spreadsheets.

With a set of filters and redundant data it is possible to implement it in a spreadsheet without using a programming language. Nevertheless this kind of solution is not easy and intuitive to users without sophisticated computer science skills. We claim that solutions to trivial problems should be easy and intuitive for all users. In Figure 3 we illustrate the desired spreadsheet feature.



Flight Connections

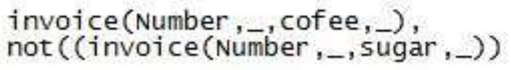
Airport Origin	Airport Destination	Company
London Heathrow	Frankfurt am Main International	British
Tokyo International	Charles de Gaulle International	JAL
John F Kennedy International	Amsterdam Schiphol	American AL
London Heathrow	Tokyo International	British
Frankfurt am Main International	Tokyo International	KLM
Amsterdam Schiphol	Frankfurt am Main International	KLM

?	*From: London Heathrow*	*To: Tokyo International*	
Alt 1	London Heathrow	Frankfurt am Main International	British
	Frankfurt am Main International	Tokyo International	KLM
Alt 2	London Heathrow	Tokyo International	British

Figure 3 – An example of a recursive query

In the example presented in Figure 3 the user wants to know if it is possible to fly from London to Tokyo. There is one alternative: a flight with a connection in Frankfurt. To automatically answer the question (when there aren't direct flights) the user must include in the spreadsheet a recursive rule that says something as "A flight is a direct flight, or a direct flight followed by a flight."

Relation manipulation and calculus are strongly developed in several languages. One of such examples is the database standard query language SQL (Structured Query Language, [Date and O'Reilly 2009]). Prolog Language [Clocksin and Mellish 2003] is another example of a powerful language suitable for relational and recursive processing, perhaps the most flexible and complete one. Prolog, for example, deals very naturally with recursive reasoning, contrary to SQL, which does not allow it at all. Prolog comes from a set of approaches based on logical representation and calculus. There are already some approaches trying to integrate logical mechanisms into spreadsheets, but we consider that all of them are far from achieving a satisfactory solution (in section 5 we summarize those approaches). However the underlying idea ("deductible spreadsheets") seemed to be adequate to our intents. The solution we present in this paper can be used with the Prolog language, but in such a way that integration is transparent to the users. From the users' perspective, what they have is just another Microsoft Excel function that allows very powerful queries, involving relational and recursive processing. Our new "function" has the same user interface as other Microsoft Excel functions. To illustrate Prolog potential, in Figure 4, Figure 5 and Figure 7 we present examples of queries that will solve the problems illustrated in examples presented in Figure 1, Figure 2 and Figure 3.



```
invoice(Number,_,cofee,_),
not((invoice(Number,_,sugar,_))
```

Figure 4 – Prolog Query that answers query presented in Figure 1

Informally, this query can be read as “What are the invoice numbers (variable Number) including coffee but not sugar?”. Using the same variable Number in the two parts of the query ensures that it refers to the same invoice that should satisfy the two conditions: including coffee but not sugar.

```
'Sales Per State'(Product, Value, State):-
    Invoice(Number,_,Product, Value),Salesman(Number,_,State)
```

Figure 5 – Prolog View equivalent to the virtual table of Figure 2

Figure 5 shows the Prolog definition of the virtual table described in Figure 2. The definition may have the following informal reading: “The specified Product, with the specified Value was sold in the specified State, if there is an invoice with the same product and value, sold by a salesman in the specified State”. The underscore in the second argument of the *Invoice* table means that the Item is not important. The underscore in the second argument of the *Salesman* table means that the name of the salesman is irrelevant.

If we define the virtual table ‘Sales Per State’ as per Figure 5, in order to obtain the sales from Utah we just need do execute the following query:

```
'Sales Per State'(Product, Value, utah)
```

Figure 6 – Querying a Prolog virtual table (sales from Utah)

The recursive rule of the flight example can be represented in Prolog as follows:

```
flight(Origin,Destination):-directFlight(Origin,Destination).

flight(Origin,Destination):-directFlight(Origin,Connection),flight(Connection,Destination).
```

Figure 7 – Recursive Flight Rule in Prolog

The flight definition presented in Figure 7 has two clauses. The first one says that there is a flight between two airports (Origin and Destination) if there is a direct flight between them.

The second part of the definition says that there is a flight between the Origin and the Destination airports if there is a direct flight from the Origin airport to another airport (Connection) and there is a flight from that airport (Connection) to the Destination.

Notice that the Prolog syntax is very similar to the SQL syntax that is taught to business students in several universities. Users with basic informatics skills can easily use SQL to retrieve information directly from a database without the support of dedicated forms. The SQL equivalent of the Prolog query presented in Figure 6 is the one depicted in Figure 8. Our point is that, if non-professional users can intuitively perform SQL queries, they will also be capable of doing the same in Prolog.

```
Select Product, value From "Sales Per State" where State = 'Utah'
```

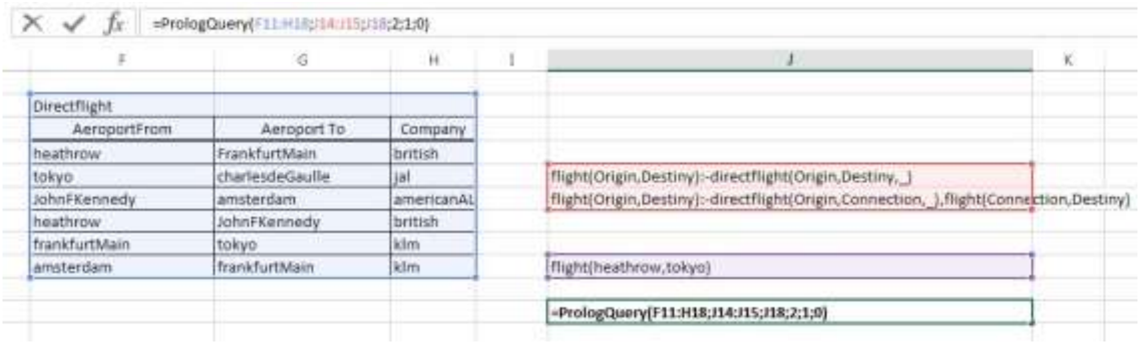
Figure 8 – Querying a View (sales from Utah)

Our tool integrates into Microsoft Excel spreadsheets exactly with the same features illustrated in Figure 4, Figure 5, Figure 6, and Figure 7. As we will see further down, we have managed to develop this functionality with the support of Swi-Prolog1 [Wielemaker et al 2012], an open source C implementation of Prolog.

The paper is organized as follows: Section 2 contains an overview of the features of our tool. Some technical details about the tool are presented in section 3. In section 4 comprehensible examples are used to illustrate how our tool exceeds Microsoft Excel limitations to support some basic queries. In section 5 we present related work, namely, other approaches to bringing together spreadsheets and the kind of features supported by our tool. In the last section we point out some directions for future developments.

2 Using Prolog within the Spreadsheet

In this section, we present Prolog Query, its features, and the way the user interacts with it. In Figure 9 we can see an example of a Prolog Query interaction. A query is executed through a function placed on a single cell (last row of column J in the example). The function returns the "answers" in the cells immediately below it. A query has several components (arguments), but the relevant ones are the **database** (columns F, G and H) where the data is placed, the **rules** that can provide new data (first two non blank rows of column J) and the **question** itself (third non blank row of column J). The function arguments are explained in detail throughout the present section.



=PrologQuery(F11:H18;J14:J15;J18;2;1;0)

Directflight		
AeroportFrom	Aeroport To	Company
heathrow	FrankfurtMain	british
tokyo	charlesdeGaulle	jal
JohnFKennedy	amsterdam	americanAL
heathrow	JohnFKennedy	british
frankfurtMain	tokyo	klm
amsterdam	frankfurtMain	klm

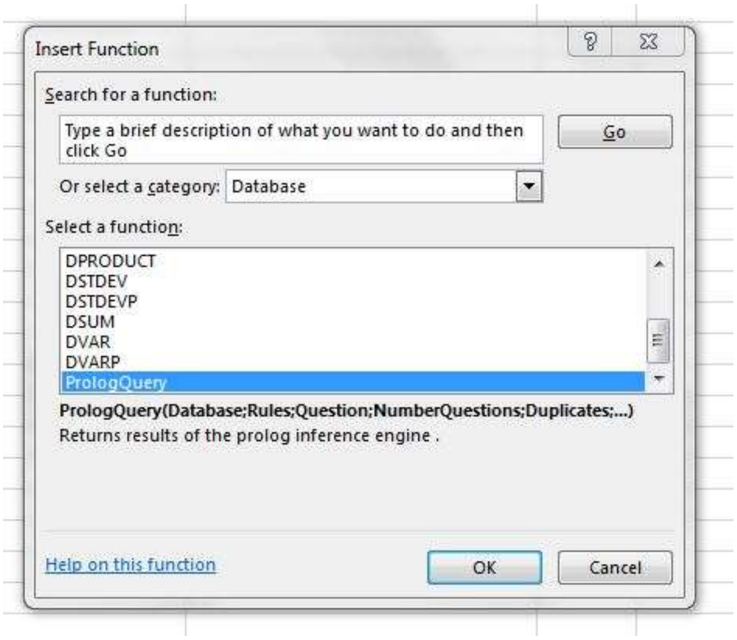
```
flight(Origin,Destiny):-directflight(Origin,Destiny,_)
flight(Origin,Destiny):-directflight(Origin,Connection,_),flight(Connection,Destiny)

flight(heathrow,tokyo)

=PrologQuery(F11:H18;J14:J15;J18;2;1;0)
```

Figure 9 – Prolog Query Function Example

Prolog Query is a user-defined function under the Database Functions Category (Figure 10).

---

[1] http://www.swi-prolog.org


Insert Function

Search for a function:

Type a brief description of what you want to do and then click Go

Or select a category: Database

Select a function:

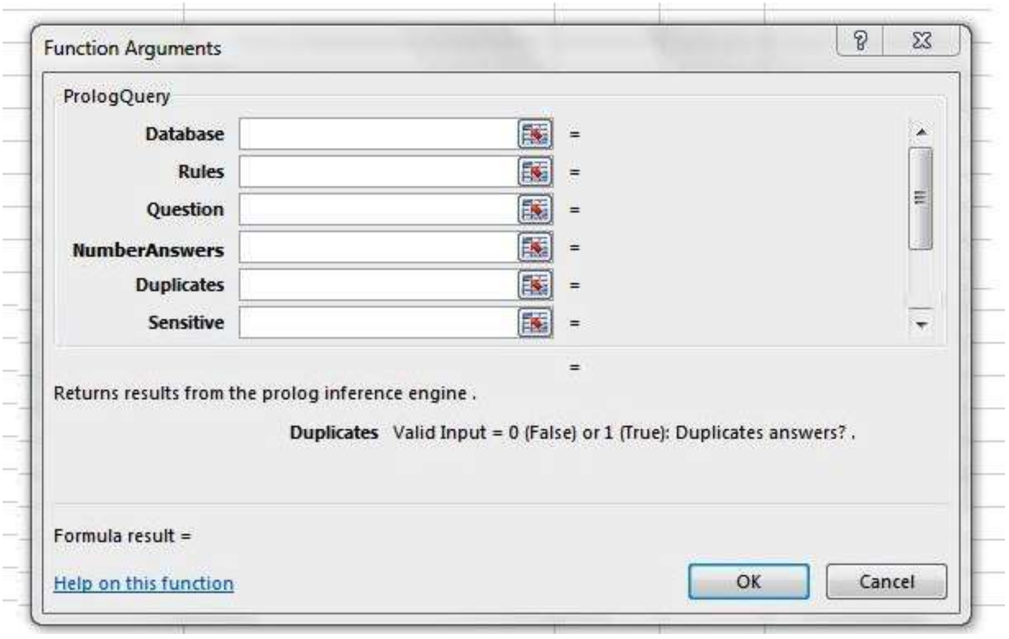
DPRODUCT
DSTDEV
DSTDEVP
DSUM
DVAR
DVARP
PrologQuery

PrologQuery(Database;Rules;Question;NumberQuestions;Duplicates;...)
Returns results of the prolog inference engine .

Figure 10 – *PrologQuery* as a Database Function

The PrologQuery function has six parameters that can be instantiated in the dialog box presented in Figure 11: database, rules, question, number of answers, permission to get duplicate answers, and case sensitiveness.

Database. The database (facts) containing the information that supports the queries. It is defined as a datasheet range that may not be empty. A database may contain several tables. Each table has a name (placed in the first row) and columns. The columns names must be placed in the second row of the table, as in the DirectFlight table presented in Figure 9. Tables may contain blank lines, but they will be omitted.


Function Arguments

PrologQuery

Database =
Rules =
Question =
NumberAnswers =
Duplicates =
Sensitive =

Returns results from the prolog inference engine .

Duplicates Valid Input = 0 (False) or 1 (True): Duplicates answers? .

Formula result =

Figure 11 – *PrologQuery* parameters dialog box

Rules. Rules are usually created in order to generate new facts based on the ones existing in the database. The "Virtual table" presented in Figure 2 and the query presented in Figure 1 do not require specific rules definition: these are examples of direct queries to the database. In the Figure 3 example the definition of a new rule is needed. The parameter is defined as a datasheet range that may be empty.

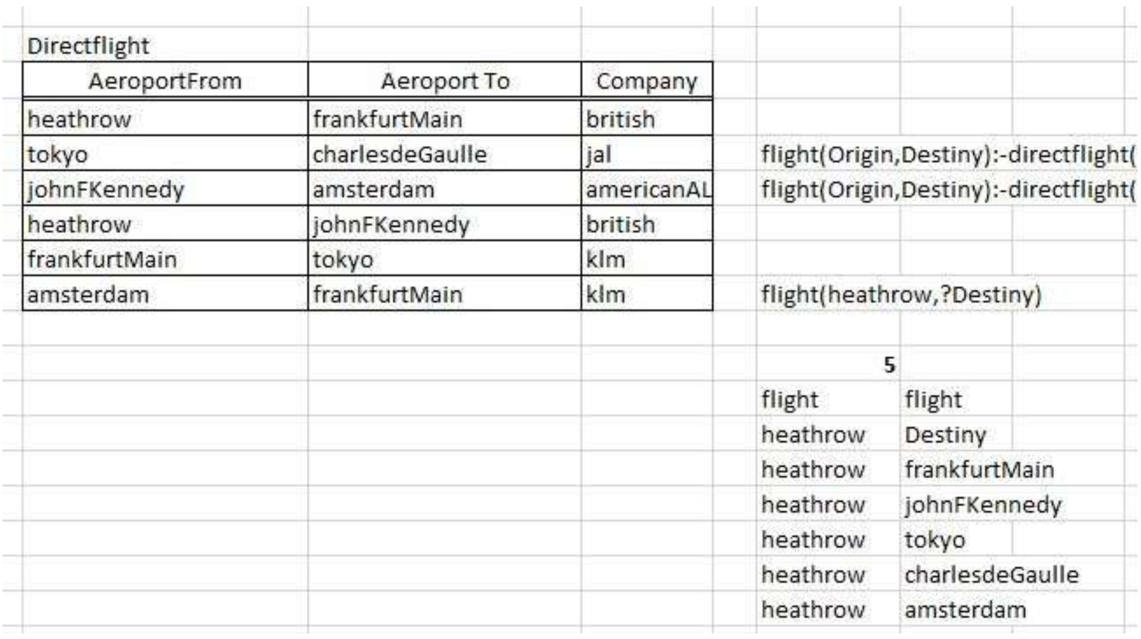
For example, the rule presented in the Figure 3 example (Flight) was created to represent information, which was not explicitly contained in the database: information about Flights. Flights are inferred based on a more specific concept already presented in the database: direct flights. The rule `flight(Origin, Destination):-directFlight(Origin, Destination)` means that if directFlight(Origin, Destination) is a fact (whatever the values for Origin and Destination, which work here as variables), then flight(Origin, Destination) is true. In the second rule, we can see that if directFlight(Origin, Connection) is a fact and flight(Connection, Destination) is true, then flight(Origin, Destination) is also true.

Question. The Question parameter is used to specify the query expression defined in the spreadsheet. The parameter is a cell range that may not be empty. Several query examples are presented throughout the paper. The query syntax is that of Prolog: it is a simple query (e.g., `'Sales per state'(Product, Value, utah))` or a compound query made up of conjunctions, disjunctions and negations of queries, in which simple queries have the form tablename<columns1, column2>), conjunctions are specified through the Prolog conjunction operator (`,` or *and*), disjunctions are specified through the Prolog disjunction operator (`;`) and negations are specified through the Prolog negation operators (`not` or `\+`). Constants are used to constrain (i.e., to filter) the answers. For example, `'Sales Per State'(Product, Value, Utah)` (Figure 6) is the same as asking "the columns Product and Value of all rows of table 'Sales Per State' where the State is 'utah'. The names of the predicate arguments do not need to be those of the table columns. Only their position in the predicate is relevant (the first argument corresponds to the first column, the second argument corresponds to the second column, and so on). The comma symbol (between predicates) represents columns, and the not operator returns true if the table inside is empty.

The other three arguments (NumberAnswers, Duplicates, and Sensitive) are optional and do not require much explanation. The first, NumberAnswers (integer data type) can be used to limit the number of rows retrieved by the query. The next one, Duplicates (logical data type), can be used to eliminate duplicate rows from the answers to be presented. The last one will be explained later in the paper (it has to do with the distinction between lower and uppercase letters).

In Figure 9 we simplified the name of the airports and removed all blank spaces for the sake of the explanation (as we will see later, we can use capital letters and blank spaces with some restrictions). The function returns **True** (not visible in the figure) since there is a flight between Heathrow and Tokyo (via Frankfurt).

In the next example, we *ask* where we can go if we leave from Heathrow. The question mark indicates that Destination is not a constant (a specific airport) but a variable that will become instantiated, in the answers, with all airports that match the query. The function returns the number five and presents the five flights. Notice that the query returns all flights (direct and indirect). The first row corresponds to the new table name (flight), the name of the columns comes in the second row, and the third row of the result is the first row retrieved from the database. For example, row <heathrow charlesdeGaulle> corresponds to a long flight with two connections: heathrow – frankfurt – tokyo – charlesdeGaulle.



Directflight

AeroportFrom	Aeroport To	Company
heathrow	frankfurtMain	british
tokyo	charlesdeGaulle	jal
johnFKennedy	amsterdam	americanAL
heathrow	johnFKennedy	british
frankfurtMain	tokyo	klm
amsterdam	frankfurtMain	klm

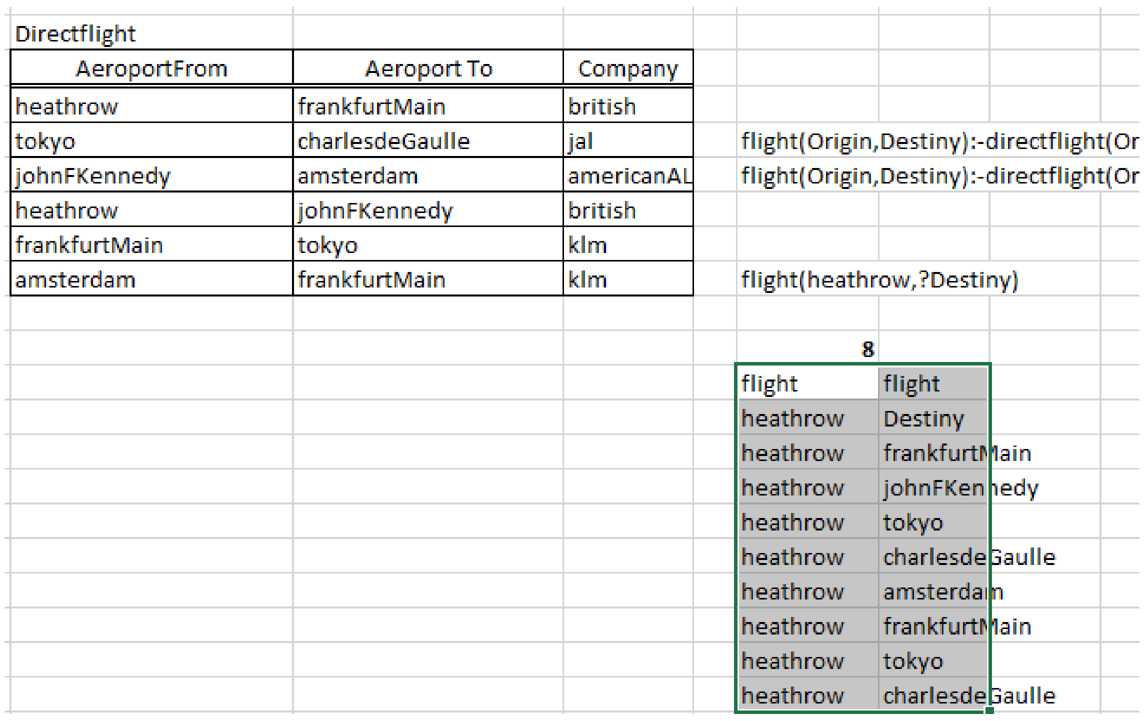
```
flight(Origin,Destiny):-directflight(
flight(Origin,Destiny):-directflight(

flight(heathrow,?Destiny)
```

	5
flight	flight
heathrow	Destiny
heathrow	frankfurtMain
heathrow	johnFKennedy
heathrow	tokyo
heathrow	charlesdeGaulle
heathrow	amsterdam

Figure 12 – Flight example without duplicates

Notice that the function in Figure 12 only returns five rows because the *Duplicates* parameter is set to false (no duplicates). Since there are two different alternatives to arrive to Frankfurt, the result will have duplicated rows. Figure 13 exemplifies this case, exactly with the same query but with the *Duplicates* parameter set to **true**. As we try to illustrate in the figure, after the query is executed, the returned rows are automatically highlighted.

It is important to understand that if one row, which affects the function is changed (cells from the database, the rules or the query), the results of the function are automatically recalculated. Notice that we also can have more than one PrologQuery function per worksheet.



Directflight

AeroportFrom	Aeroport To	Company
heathrow	frankfurtMain	british
tokyo	charlesdeGaulle	jal
johnFKennedy	amsterdam	americanAL
heathrow	johnFKennedy	british
frankfurtMain	tokyo	klm
amsterdam	frankfurtMain	klm

flight(Origin,Destiny):-directflight(Or

flight(Origin,Destiny):-directflight(Or

flight(heathrow,?Destiny)

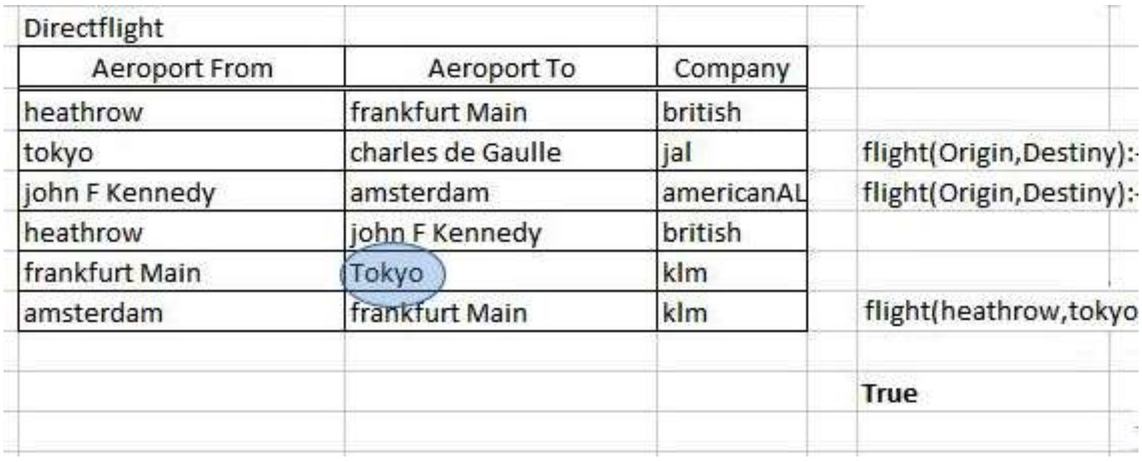
8

flight	flight
heathrow	Destiny
heathrow	frankfurtMain
heathrow	johnFKennedy
heathrow	tokyo
heathrow	charlesdeGaulle
heathrow	amsterdam
heathrow	frankfurtMain
heathrow	tokyo
heathrow	charlesdeGaulle

Figure 13 – Flight example with duplicates

Prolog makes a clear distinction between lower and upper case in what regards the first letter of a symbol (e.g., predicate name or argument). Uppercase first letter is used to represent variables; lower case first letter specifies an alphanumeric constant. Predicates names must start with a lowercase letter. In Prolog, if the user wants to have a constant that starts with a capital letter the constant must be enclosed within the single quote characters, 'England' for example. Those distinctions are odd to Microsoft Excel users, so we hid them from users. The last function parameter (*Sensitive*) may have two possible values: 0 and 1. The zero value means that data will be treated exactly as it was written in cells, which means that the strings starting with capital letters in the database will be treated as constants. Otherwise, before the query is processed by Prolog, all first letters will be replaced with lower case letters.

In Figure 14 the query returns True because the zero value was used for the parameter *Sensitive*. Consequently, in the internal Prolog database the fact stored is the following: directflight(frankfurtMain, tokyo, klm), and not as it is written in excel (directflight(frankfurtMain, Tokyo, klm). If the parameter had the value one the result would only continue to be **True** if the question were changed to flight(heathrow, Tokyo). In that scenario, if we want to use constants beginning with an uppercase letter in the rules, the variables must be preceded by the dollar ($) character ($Tokyo, for example).



Directflight

Aeroport From	Aeroport To	Company
heathrow	frankfurt Main	british
tokyo	charles de Gaulle	jal
john F Kennedy	amsterdam	americanAL
heathrow	john F Kennedy	british
frankfurt Main	Tokyo	klm
amsterdam	frankfurt Main	klm

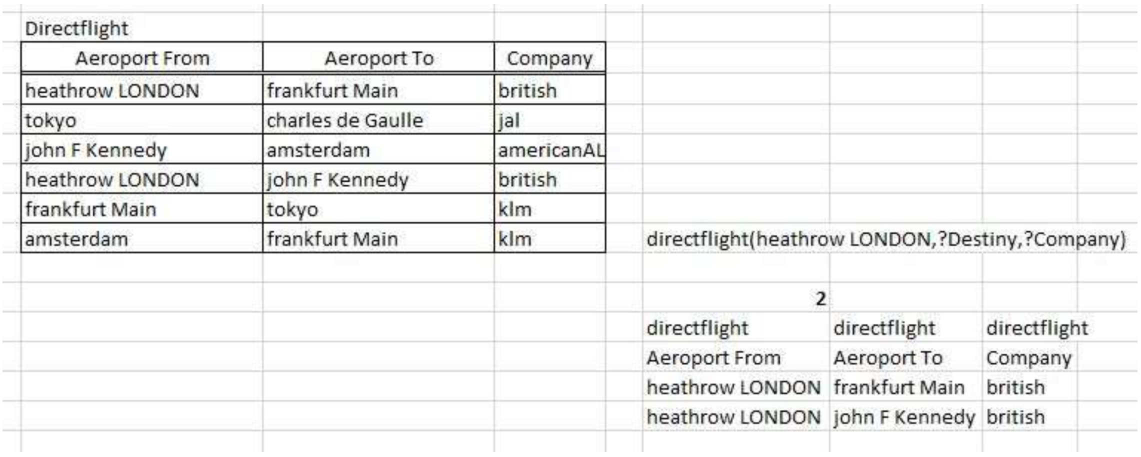
flight(Origin,Destiny):-

flight(Origin,Destiny):-

flight(heathrow,tokyo

True

Figure 14 – Flight example with capital letters



Directflight

Aeroport From	Aeroport To	Company
heathrow LONDON	frankfurt Main	british
tokyo	charles de Gaulle	jal
john F Kennedy	amsterdam	americanAL
heathrow LONDON	john F Kennedy	british
frankfurt Main	tokyo	klm
amsterdam	frankfurt Main	klm

directflight(heathrow LONDON,?Destiny,?Company)

2

directflight	directflight	directflight
Aeroport From	Aeroport To	Company
heathrow LONDON	frankfurt Main	british
heathrow LONDON	john F Kennedy	british

Figure 15 – Flight example with blank spaces

In Figure 15 one can see that blank spaces can be used in the database (in the facts as well as in the column names), and in the question. However, they cannot be used in the table name or in the predicates name.

3 System Architecture

Technically, Prolog Query is essentially a sophisticated Microsoft Excel interface to Swi-Prolog. All data required for a PrologQuery function invocation is collected from the spreadsheet. Prolog is only used to process the queries. Used data and definitions are not stored in Prolog internal database after the query has been processed and answered. When the PrologQuery function is executed in the spreadsheet the following steps take place (see Figure 16):

a) The facts, the rules and the query are converted to the Prolog syntax;
b) After the conversion, the facts, rules and query are written in a temporary file created by the plug-in (in the current user folder);
c) Swi-Prolog is automatically executed via command line (transparent to the user);
d) Swi-Prolog imports the temporary file and executes the query;

e) Swi-Prolog output is written in a temporary file in the same location;
f) Data is imported to excel and the temporary file is removed;
g) Data is displayed in the spreadsheet.

Any user spreadsheet may contain any ammount of tables and definitions to be processed by Prolog. Users may submit all kinds of queries to Prolog, using the whole Prolog potential. Notice that, unlike the data in the database range, the data placed in the rules range is “sent” to Prolog without conversion. A complete Prolog program may be placed in the rules range. But the most important feature of our tool is the spreadsheet interface supplied to Excel users. They do not need to understand Prolog language in order to intuitively work with powerful queries.

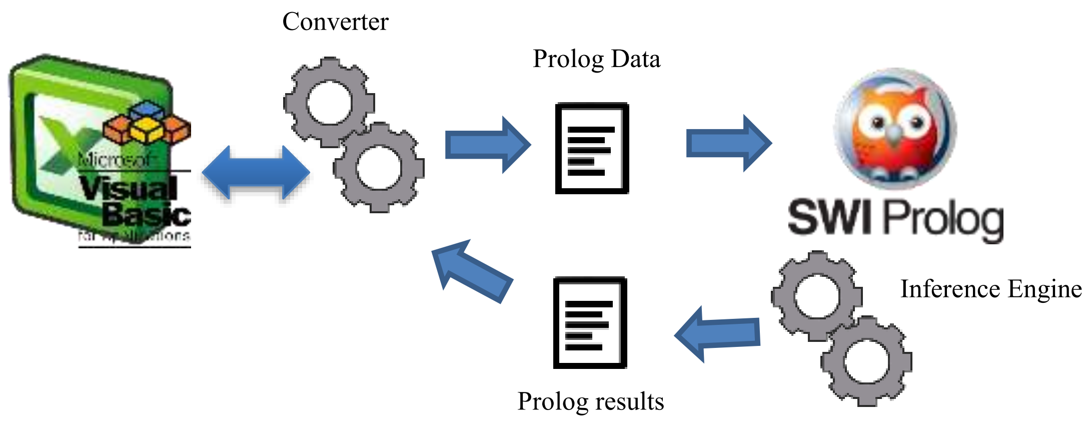


Figure 16 – Prolog Query Architecture

To work with the spreadsheet it is mandatory to have Swi-Prolog and Java (JRE) installed in the same machine. The tool only uses the current user folder and needs to access the global environment system variable PATH.

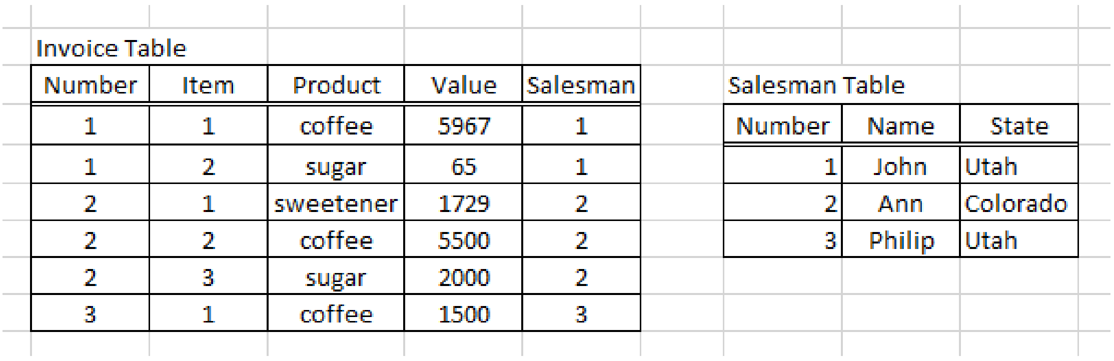
4 Exploring Prolog Query features with business examples

In the present section we illustrate how our tool exceeds Microsoft Excel limitations, providing easy spreadsheet support to some basic queries.

4.1 Product Sales Scenario

In Figure 17 we come back to the example presented in the first section, namely to queries whose filtering criteria concern more than one row in a table.

In the example, since one invoice can include more than one item, the user can be interested in retrieving invoices for a specific product that do not include other specified products (for instance, to assess the success of specific selling policies).



Invoice Table

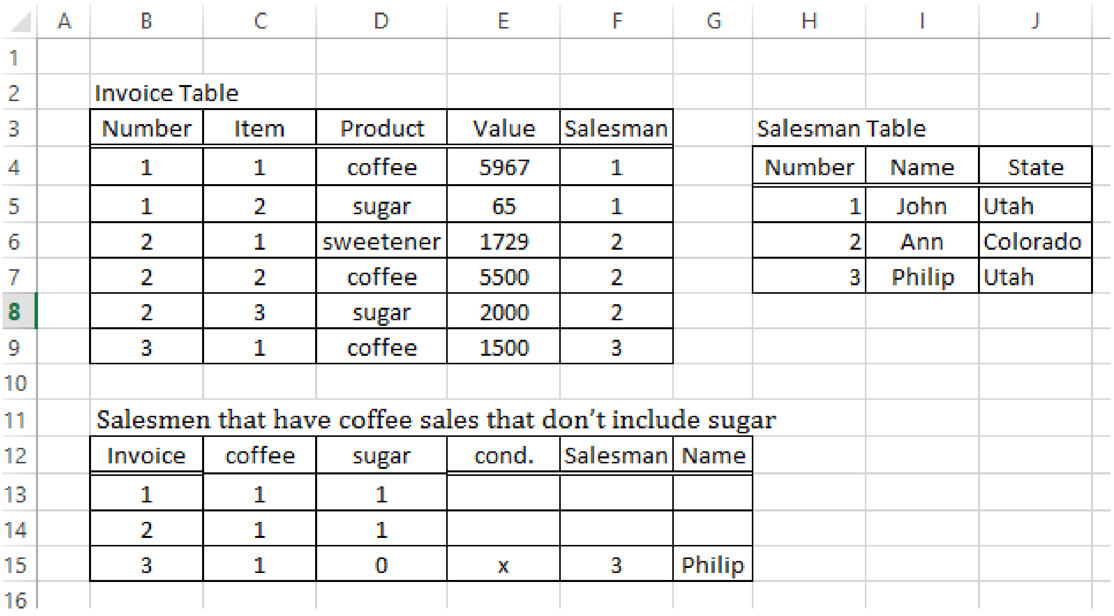
Number	Item	Product	Value	Salesman
1	1	coffee	5967	1
1	2	sugar	65	1
2	1	sweetener	1729	2
2	2	coffee	5500	2
2	3	sugar	2000	2
3	1	coffee	1500	3

Salesman Table

Number	Name	State
1	John	Utah
2	Ann	Colorado
3	Philip	Utah

Figure 17 - Product Sales Example

Considerer the following query: *who are the salesmen who have coffee sales that don't include sugar?* This kind of query is not easily made using Microsoft Excel features. The fact that one invoice is described in more than one row prevents a solution based on Pivot Tables or filtering. In the next figure we illustrate a possible implementation using standard Excel.



	A	B	C	D	E	F	G	H	I	J
1										
2		Invoice Table								
3		Number	Item	Product	Value	Salesman		Salesman Table		
4		1	1	coffee	5967	1		Number	Name	State
5		1	2	sugar	65	1		1	John	Utah
6		2	1	sweetener	1729	2		2	Ann	Colorado
7		2	2	coffee	5500	2		3	Philip	Utah
8		2	3	sugar	2000	2				
9		3	1	coffee	1500	3				
10										
11		Salesmen that have coffee sales that don't include sugar								
12		Invoice	coffee	sugar	cond.	Salesman	Name			
13		1	1	1						
14		2	1	1						
15		3	1	0	x	3	Philip			
16										

Figure 18- Product Sales Example with standard Excel

The bottom table (Salesmen who have coffee sales that don't include sugar) is a kind of temporary table built from the other two. Data in columns C, D, E, F and G is automatically filled based on the following formulas:

```
C13 -> =COUNTIFS(B$4:B$9;B13;D$4:D$9;$C$12)

D13 -> =COUNTIFS(B$4:B$9;B13;D$4:D$9;$D$12)

E13 -> =IF(AND(C13>0;D13=0);"x";"")

F13 -> =IF(E13="";"";VLOOKUP(B13;B$4:F$9;5;0))

G13 -> =IFERROR(VLOOKUP(F13;H$5:J$7;2;0);"")
```

The “cond.” column (E13) is used to mark the rows that satisfy the query (coffee without sugar) with an “x”. We may apply filters to the table in order to display only the desired rows (where cond. = “x”).

This solution is not flexible at all. Let us assume the user wants to have queries based on other products: the user should have a column for each product, which is impracticable. Users should have a practical way to quickly and easily obtain exploratory data analysis. For example, the user may be interested in knowing the total amount of invoices that sell only secondary products (sugar and sweetener). A new temporary table can be built, similar to the previous one, but this represents too much work for a simple query.

PrologQuery provides a much simpler solution, as illustrated in Figure 19.

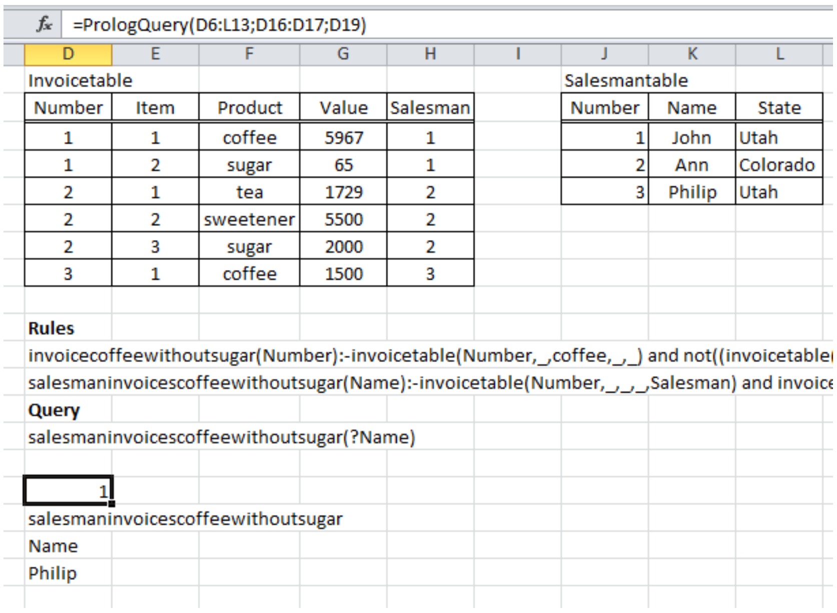
The user only needs to define two rules (we choose to use ‘and’ instead of ‘,’ to represent conjunctions, since it is more intuitive and both syntaxes are correct):

The following,

```
/* An invoice corresponds to a coffee sale that does not
include sugar if one of its rows corresponds to coffee and
none of its rows pertains to sugar. */
invoicecoffeewithoutsugar(Number):-
  invoicetable(Number,_,coffee,_,_) and
  not((invoicetable(Number,_,sugar,_,_)))
```

and a second one

```
/* A salesman has coffee sales that do not include sugar if
he is associated to an invoice that corresponds to a coffee
sale that does not include sugar */
salesmaninvoicescoffeewithoutsugar(Name):-
  invoicetable(Number,_,_,_,Salesman)and
  invoicecoffeewithoutsugar(Number)and
  salesmantable(Salesman,Name,_)
```



=PrologQuery(D6:L13;D16:D17;D19)

Invoicetable

Number	Item	Product	Value	Salesman
1	1	coffee	5967	1
1	2	sugar	65	1
2	1	tea	1729	2
2	2	sweetener	5500	2
2	3	sugar	2000	2
3	1	coffee	1500	3

Salesmantable

Number	Name	State
1	John	Utah
2	Ann	Colorado
3	Philip	Utah

Rules

invoicecoffeewithoutsugar(Number):-invoicetable(Number,_,coffee,_,_) and not((invoicetable(

salesmaninvoicescoffeewithoutsugar(Name):-invoicetable(Number,_,_,_,Salesman) and invoice

Query

salesmaninvoicescoffeewithoutsugar(?Name)

1

salesmaninvoicescoffeewithoutsugar

Name

Philip

Figure 19 - Product Sales Example using PrologQuery

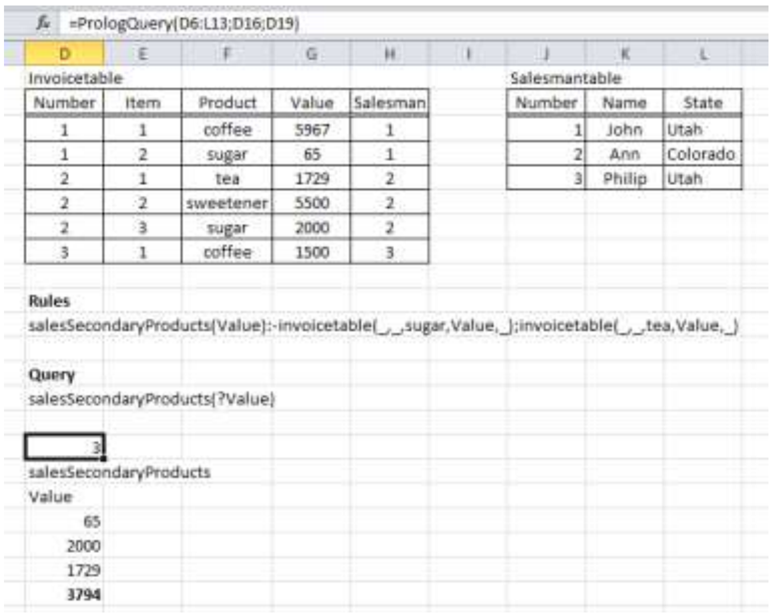
The query salesmaninvoicescoffeewithoutsugar(?Name) finds only one solution (1) and displays the corresponding name ‘Philip’.

In

Figure 20 we present the solution to the second query considered: the total amount of invoices that only sell secondary products.

The previous two rules are now replaced by a single one (‘;’ means OR):

```
salesSecondaryProducts(Value):-
invoicetable(_,_,sugar,Value,_);
invoicetable(_,_,tea,Value,_)
```



=PrologQuery(D6:L13;D16;D19)

Invoicetable

Number	Item	Product	Value	Salesman
1	1	coffee	5967	1
1	2	sugar	65	1
2	1	tea	1729	2
2	2	sweetener	5500	2
2	3	sugar	2000	2
3	1	coffee	1500	3

Salesmantable

Number	Name	State
1	John	Utah
2	Ann	Colorado
3	Philip	Utah

Rules

salesSecondaryProducts(Value):-invoicetable(_,_,sugar,Value,_);invoicetable(_,_,tea,Value,_)

Query

salesSecondaryProducts(?Value)

3

salesSecondaryProducts

Value

65

2000

1729

3794

Figure 20 - Product Sales Example with PrologQuery (cont.)

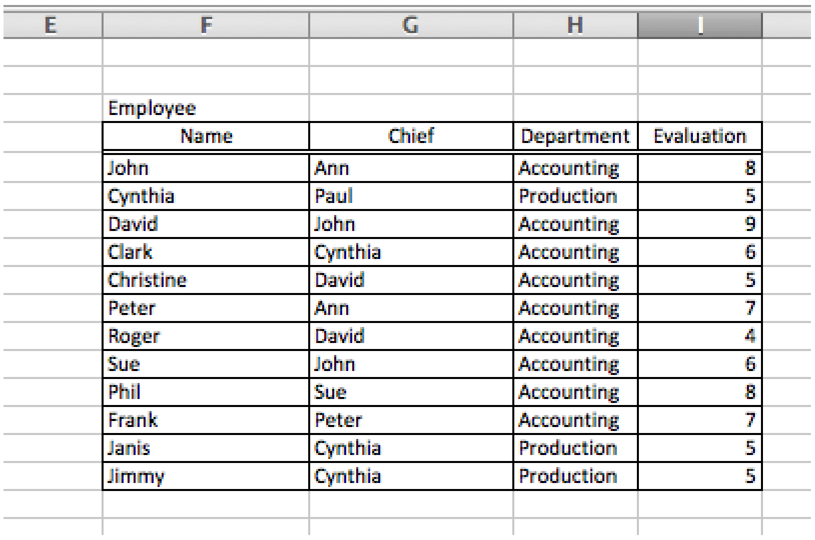
In order to obtain the total amount of sales it is sufficient to use a sum function to compute the three values, returning 3794. Users may have sheets with PrologQuery and original Excel formulas.

Notice that the Or (;) is optional; the same rule can be written in the following way (two expressions):

```
salesSecondaryProducts(Value):-
invoicetable(_,_,sugar,Value,_)
salesSecondaryProducts(Value):-
invoicetable(_,_,tea,Value,_)
```

4.2 Performance evaluation, a recursive scenario

In Figure 21 we can see some data from an employee table imported from a database. The first row should be read as follows: John is one of Ann's subordinates, he works in the accounting department and he has been evaluated with an eight (in a ten point scale). The data in the table reflects a hierarchical structure, like an organization chart, in which Ann and Paul are at the top of the tree.



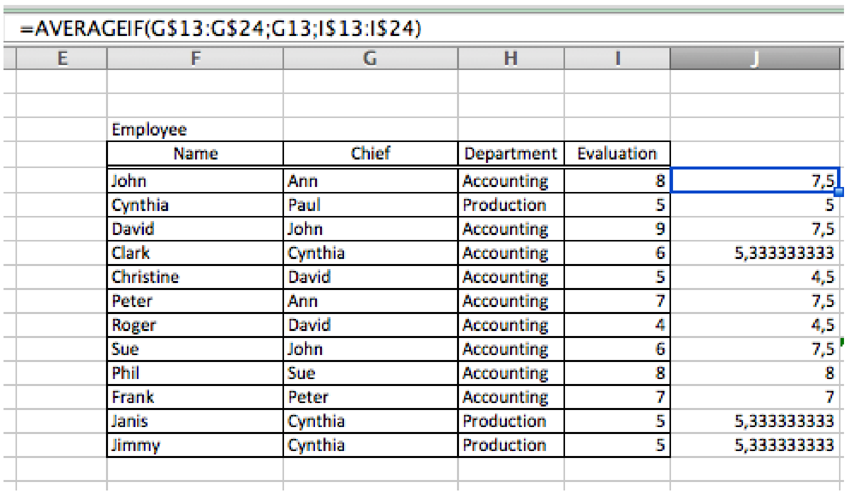
Employee

Name	Chief	Department	Evaluation
John	Ann	Accounting	8
Cynthia	Paul	Production	5
David	John	Accounting	9
Clark	Cynthia	Accounting	6
Christine	David	Accounting	5
Peter	Ann	Accounting	7
Roger	David	Accounting	4
Sue	John	Accounting	6
Phil	Sue	Accounting	8
Frank	Peter	Accounting	7
Janis	Cynthia	Production	5
Jimmy	Cynthia	Production	5

Figure 21 – Employee Table

Let us assume that the final Evaluation score of each employee is a combination of their own score (fourth column), with the average evaluation of all their subordinates (including the subordinates of their subordinates and so on).

Using Excel built in functions it is not possible to add a computed column that calculates the desired value. The AVERAGEIF function placed on a row can only calculate the average of the direct subordinates. For example, if the function is placed on the first row (=AVERAGEIF(G$13:G$24;G13;I$13:I$24)), it will return 7.5, the average of the evaluations of John and Peter (Figure 22)



```
=AVERAGEIF(G$13:G$24;G13;I$13:I$24)
```

Employee

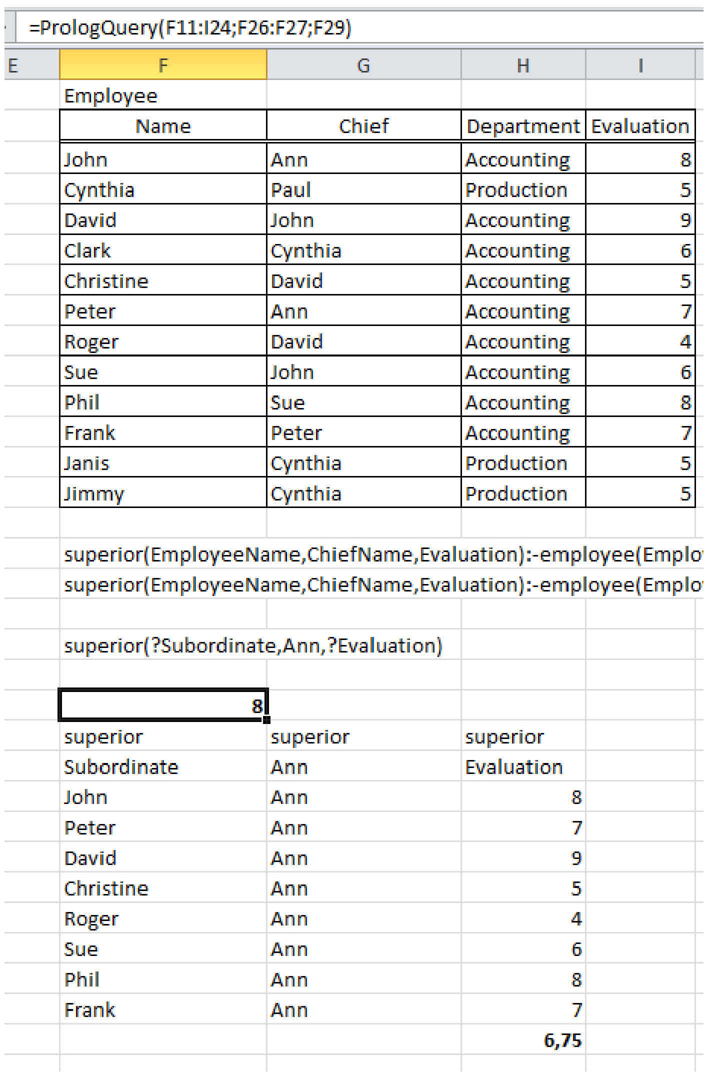
Name	Chief	Department	Evaluation	
John	Ann	Accounting	8	7,5
Cynthia	Paul	Production	5	5
David	John	Accounting	9	7,5
Clark	Cynthia	Accounting	6	5,333333333
Christine	David	Accounting	5	4,5
Peter	Ann	Accounting	7	7,5
Roger	David	Accounting	4	4,5
Sue	John	Accounting	6	7,5
Phil	Sue	Accounting	8	8
Frank	Peter	Accounting	7	7
Janis	Cynthia	Production	5	5,333333333
Jimmy	Cynthia	Production	5	5,333333333

Figure 22 – Performance Evaluation with Standard Excel

In order to get the average of all subordinates, it is necessary to extract them first, and then use the usual average function. In Figure 23 we exemplify how it can be done with PrologQuery.

In the solution we use a rule defined in two steps. Employee A is employee B's superior, in the hierarchical structure, if (i) A is B's direct superior (first part); or, A is a direct superior of another employee that is B's superior (second part). The rule is as follows:

```
superior(EmployeeName,ChiefName,Evaluation):-
  employee(EmployeeName,ChiefName,_,Evaluation)
superior(EmployeeName,ChiefName,Evaluation):-
  employee(EmployeeName,Person,_,_)and
  superior(Person,ChiefName,Evaluation)
```


=PrologQuery(F11:I24;F26:F27;F29)

E	F	G	H	I
	Employee			
	Name	Chief	Department	Evaluation
	John	Ann	Accounting	8
	Cynthia	Paul	Production	5
	David	John	Accounting	9
	Clark	Cynthia	Accounting	6
	Christine	David	Accounting	5
	Peter	Ann	Accounting	7
	Roger	David	Accounting	4
	Sue	John	Accounting	6
	Phil	Sue	Accounting	8
	Frank	Peter	Accounting	7
	Janis	Cynthia	Production	5
	Jimmy	Cynthia	Production	5

superior(EmployeeName,ChiefName,Evaluation):-employee(Emplo
superior(EmployeeName,ChiefName,Evaluation):-employee(Emplo

superior(?Subordinate,Ann,?Evaluation)

8			
superior	superior	superior	
Subordinate	Ann	Evaluation	
John	Ann	8	
Peter	Ann	7	
David	Ann	9	
Christine	Ann	5	
Roger	Ann	4	
Sue	Ann	6	
Phil	Ann	8	
Frank	Ann	7	
		6,75	

Figure 23 - Performance Evaluation using PrologQuery

The query (`superior(?Subordinate,Ann,Evaluation)`) returns eight rows. Having the results, we just need to add a trivial average function to get the desired result: 6.75.

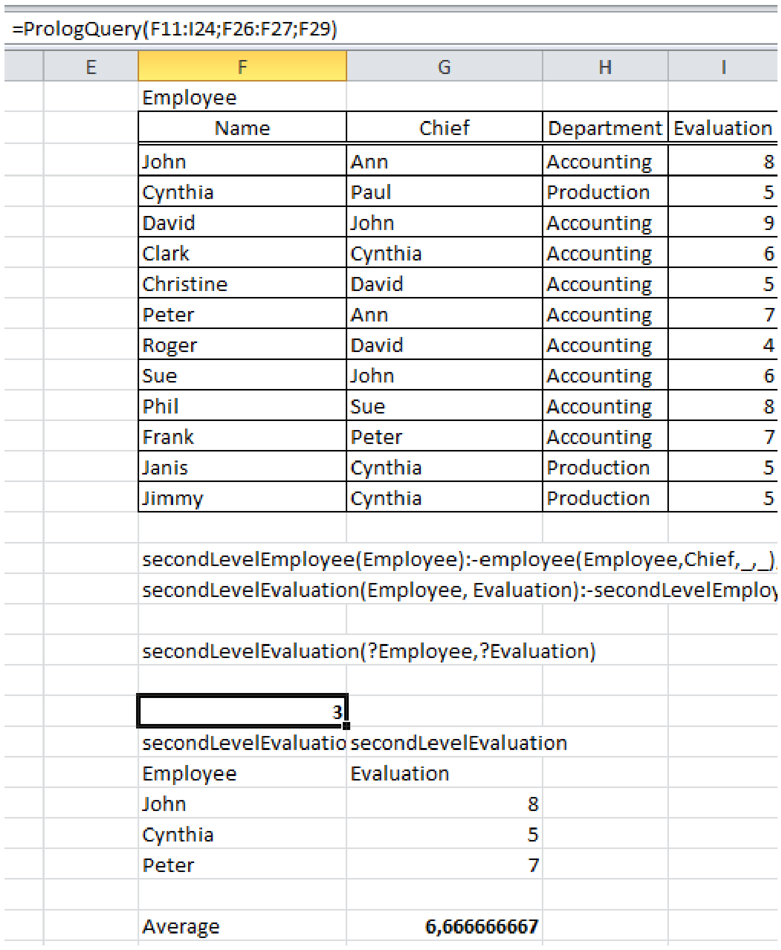
Let us consider a different but similar query: What is the average evaluation of the employees on the second level of the organizational structure?

In order to apply the average function we first need to define what second level employees are. The definition in PrologQuery is as follows:

```
secondLevelEmployee(Employee):-
    employee(Employee,Chief,_,_) and
    not(employee(Chief,UpperChief,_,_))
```

Informally, someone is a second level employee if they have a chief that does not have a chief, in turn. Having that rule we can have a query that returns all second level employees and their evaluation.

```
secondLevelEvaluation(Employee, Evaluation):-
    secondLevelEmployee(Employee)and
    employee(Employee,_,_,Evaluation)
```



=PrologQuery(F11:I24;F26:F27;F29)

Employee

Name	Chief	Department	Evaluation
John	Ann	Accounting	8
Cynthia	Paul	Production	5
David	John	Accounting	9
Clark	Cynthia	Accounting	6
Christine	David	Accounting	5
Peter	Ann	Accounting	7
Roger	David	Accounting	4
Sue	John	Accounting	6
Phil	Sue	Accounting	8
Frank	Peter	Accounting	7
Janis	Cynthia	Production	5
Jimmy	Cynthia	Production	5

secondLevelEmployee(Employee):-employee(Employee,Chief,_,_)
secondLevelEvaluation(Employee, Evaluation):-secondLevelEmploy

secondLevelEvaluation(?Employee,?Evaluation)

3	
secondLevelEvaluatio	secondLevelEvaluation
Employee	Evaluation
John	8
Cynthia	5
Peter	7
Average	**6,666666667**

Figure 24 - Performance Evaluation using PrologQuery (cont.)

This example can be generalized to all situations where we need to compute aggregate values, but the decomposition of rows is not explicitly represented. Another application example would be the computation of aggregate sales of geographical sales distribution (computation of values per region, states, cities,…).

5 Related Work

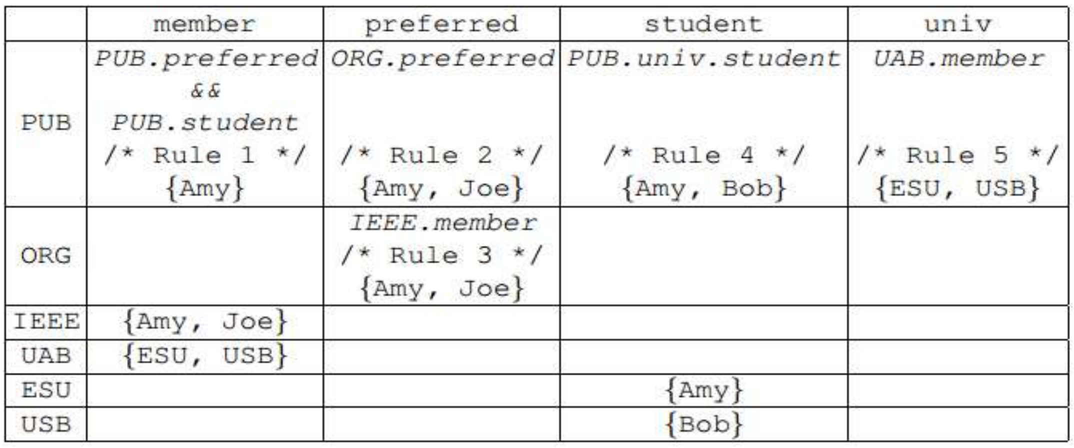
Since the late 1980s there have been several attempts to develop Logic (or Deductive) Spreadsheets that combine traditional features with reasoning capabilities.

One of the oldest, LogiCalc [Kriwaszek, 1988], follows a similar approach to the one presented in this paper: an attempt to integrate Prolog power into Microsoft Excel. However LogiCalc has some limitations, like the impossibility to have disjunctions and a weak and nonintuitive interface with users (tuples were held in a single cell, as opposed to displaying each component of a tuple in a separate column).

In 2004 DARPA (Defense Advanced Research Projects Agency, EUA) promoted an event (Small Business Innovation Research) in which several deductive spreadsheets were presented. As a consequence, four projects appeared: NEXCEL [Cervesato, 2007], LESS [Valente et al., 2007], XcelLog [Ramaskrishnan et al., 2007] and [Tallis et al., 2007]. All those systems aimed at enriching Microsoft Excel features. DARPA wanted to use those spreadsheet tools for military purposes, namely, to support quick decisions in critical situations.

It is important to refer that none of these is available. We exchanged emails with the authors of three of them, but we only had access to one demo (the demo runs over Google Spreadsheet so it's far from having Microsoft Excel functionalities). All we know about them is what is described in the papers presented in 2004.

XcelLog represents rules with Datalog [Maier and Warren 1988], a declarative logic programming language similar to Prolog (syntactically, it is a subset of Prolog, with no functional symbols). Excel works as a front end for writing rules and data. The evaluation of the queries is made through XSB in the backend. XSB (Extended Stony Brook)[2] is a Logic Programming and Deductive Database system for Unix and Windows, whose development started in the Stony Brook University [Sagonas, Swift and Warren 1994]. The way the users write the rules in Excel is exemplified in the following illustration.



	member	preferred	student	univ
PUB	*PUB.preferred* && *PUB.student* /* Rule 1 */ {Amy}	*ORG.preferred* /* Rule 2 */ {Amy, Joe}	*PUB.univ.student* /* Rule 4 */ {Amy, Bob}	*UAB.member* /* Rule 5 */ {ESU, USB}
ORG		*IEEE.member* /* Rule 3 */ {Amy, Joe}		
IEEE	{Amy, Joe}			
UAB	{ESU, USB}			
ESU			{Amy}	
USB			{Bob}	

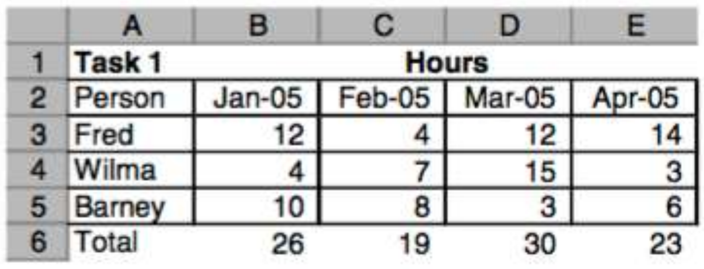
Figure 25 - An XcelLog example [Ramaskrishnan et al. 2007]

The expressions in italics within the cells represent users' intents (what users want, the "rules"), and the expressions in brackets represent the results of the "rule". For example, rule number 3 says that all IEEE members are *preferred* clients of ORG. Amy and Joe is the result of the Rule. XcelLog uses those special cells where users can "write" the rule and, simultaneously, get the results.

[2] http://xsb.sourceforge.net/index.html

Nexcel also relies on Datalog to support the deductive engine. The authors developed a specific Datalog adaption. Nexcel has an add-on developed in Visual Basic for applications that connect Microsoft Excel with the Deductive Engine. The authors gave special attention to the ability to supply explanations of the inference results to the user. The prototype we had access to is so limited (based on a Google Drive Spreadsheet tool) that it is not possible to analyze the user interface and functionalities. Once again, Microsoft Excel wasn't the chosen spreadsheet.

LESS is built on top of Microsoft Excel. A custom-built middleware was developed to provide access to logic reasoning functionality supported by PowerLoom[3]. PowerLoom is a logic-based knowledge representation and reasoning system based on a variant of KIF (Knowledge Interchange Format). Users can add formulas to the spreadsheet in order to express relational information. Those relations are stored in PowerLoom as a set of assertions that can be furthermore retrieved by other formulas. Consider the example presented in the following figure.



	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 26 - Time Schedule Example

In Microsoft Excel, the Total formula (line 6) is calculated by a SUM function (`=SUM(B3:B5)`). With LESS the user can place the following formula at the top of each column: `=ASSERTMULTI("hours", $A$3: $A$5, B2, $A$1)` and, for the given example, the following assertions will be created:

```
(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
```

Furthermore, the following formula can be placed anywhere in the spreadsheet:

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

In [Tallis et al., 2007] a different approach is proposed, using, alternatively, Prolog, Jess (a rule engine that implements the Rete algorithm), or KAON2 (an inference engine that manages Ontologies). We haven't found any examples or applications of this approach,

[3] http://www.isi.edu/isd/LOOM/PowerLoom/

but according to the authors, one of its major strengths is the ability to use a semantic representation (OWL, for instance) of the knowledge base.

Other approaches have been proposed, namely PERPLEX [Spenke and Beiken 1989], Knowledgesheet [Gupta and Akhter, 2000], CSSOLVER [Felfernig et al., 2003] and PrediCalc [Kassoff et al., 2005]. None of them is based on Microsoft Excel or in Prolog.

We haven't found, in the existing literature, any ready to use solutions that integrate Microsoft Excel with Prolog Language. As we briefly presented in this section, there have been several attempts to integrate both technologies, some of them proposing very powerful features, but none of them present a natural and easy way to realize those queries within the spreadsheet itself

We found a vba Module, created by Dipak Audy, that connects Swi-Prolog and Excel, (https://friendpaste.com/3aNELWiC1ed4bzApDhYIPl)). We tried to use it but it has too many limitations in what regards the Excel usability (namely the impossibility to use of copy/paste, to define column names). It wasn't exactly an integrated approach, but more a technical way to connect and transport data between Excel and Swi-Prolog.

6 Future Directions

In this paper we present the first version of the Prolog Query Tool. There are improvements still to be done. It is important that the tool is used and tested by Microsoft Excel advanced users. Their feedback is important since they are our priority users.

The current version has one limitation that we intend to address in the next version: lists haven't yet been properly treated in the spreadsheet (when Prolog returns a list of values, the result is placed in only one cell).

Using a file for data communication works for now, but it is not the best solution. We intend to use a DLL for a direct communication.

7 References

[Cervesato 2007] Cervesato, Iliano. 2007. NEXCEL, A deductive spreadsheet. In The Knowledge Engineering Review (2007), 22:221-236 Cambridge University Press Copyright © Cambridge University Press 2007 doi:10.1017/S0269888907001142

[Clocksin and Mellish 2003] Clocksin, W.F.; and Mellish, C.S. 2003. *Programming in Prolog. Using the ISO Standard*. Springer-Verlag. ISBN 3-540-00678-8

[Date and O'Reilly 2009] Date C. J.; and O'Reilly. 2009. SQL and Relational Theory. Publisher: O'Reilly Media . IBSN: 978-0-596-52306-0

[Felfernig et al. 2003] Felfernig, A; Friedrich, G.; Jannach D; Russ C.; and Zanker M. Developing Constraint-Based Applications with SpreadsheetsIn Developments. in

Applied Artificial Intelligence: 16th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2003 Loughborough, UK, June 23–26, 2003 Proceeding, Springer Berlin Heidelberg

[Gupta and Akhter 2000] 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 Practical Aspects of Declarative Languages, Springer Verlag LNCS 1753, Boston, MA.

[Kassoff et al., 2005] M. Kassoff, L.-M. Zen, A. Garg; and Genesereth M. 2005. Predicalc: A logical spreadsheet management system. In 31st International Conference on Very Large Databases (VLDB)

[Kriwaszek 1988] Kriwaszek, F. 1988. LogiCalc – A PROLOG Spreadsheet. in D. Michie and J. Hayes, Machine Intelligence 11, 1988.

[Maier and Warren 1988] Maier, D.; and Warren, D.S. 1988. Computing with Logic. Benjamin/Cummings. Menlo Park, California. ISBN10: 0805366814

[Ramaskrishnan et al., 2007] Ramakrishnan, C. R.; Ramakrishnan, I. V.; and Warren, D. S. 2007. XcelLog: A deductive spreadsheet system. The Knowledge Engineering Review, 22(03), 269–279.

[Sagonas, Swift and Warren 1994] Sagonas, K.; Swift, T.; and Warren, D.S. 1994. XSB as a Deductive Database. In Proceedings of the SIGMOD '94 ACM International Conference on Management of Data. ACM New York, NY, USA. DOI: 10.1145/191839.191970

[Spenke and Beiken 1989] Spenke, M.; and Beilken, C. 1989. A spreadsheet interface for logic programming, in ‘CHI ’89: Proceedings of the SIGCHI conference on Human Factors in Computing Systems’, ACM Press, pp. 75–80.

[Tallis et al. 2007] Tallis, M.; Waltzman, R.; and Balzer, R. 2007. Adding deductive logic to a COTS spreadsheet. The Knowledge Engineering Review, 22(03), 255–268.

[Valente et al. 2007] Valente, A.; van Brackle, D.; Chalupsky, H.; andEdwards, G. 2007. Implementing logic spreadsheets in LESS. The Knowledge Engineering Review, 22(03), 237–253.

[Wielemaker et al 2012] Wielemaker, J.; Schrijvers, T.; Triska, M.; and Lager, T. 2012. SWI-Prolog. Theory and Practice of Logic Programming 12(1-2):67-96. DOI: http://dx.doi.org/10.1017/S1471068411000494