System Description: Embedding Verification into Microsoft Excel *

Graham Collins[1] and Louise A. Dennis[2]

[1] Department of Computing Science, University of Glasgow, G12 8QQ, UK
[2] Division of Informatics, University of Edinburgh, EH1 1HN, UK

Abstract. The aim of the PROSPER project is to allow the embedding of existing verification technology into applications in such a way that the theorem proving is hidden, or presented to the end user in a natural way. This paper describes a system built to test whether the PROSPER toolkit satisfied this aim. The system combines the toolkit with Microsoft Excel, a popular commercial spreadsheet application.

1 Introduction

The PROSPER project is researching and developing a toolkit [1] that allows an expert to easily and flexibly assemble *proof engines* from existing tools to provide embedded formal reasoning support inside applications. The ultimate goal is to make the reasoning and proof support invisible to the end-user—or at least, more realistically, to incorporate it securely within the interface and style of interaction to which they are already accustomed. Several large case studies are taking place within the project to investigate this.

This paper describes a preliminary case study embedding verification into Microsoft Excel without inventing or re-implementing any existing theorem proving techniques or mathematical decision procedures.[1] The primary aim was to show that the technology is effective when applied to real, standard applications not designed by project members. In addition we were interested in investigating a "lightweight" theorem proving approach where only a small amount of theorem proving functionality is added but it is completely hidden from the user.

This paper begins with a brief overview of the PROSPER toolkit (§2) and Excel (§3) followed by a discussion of the system developed.

2 Extending Applications with Custom Proof Engines

A central part of PROSPER's vision is the idea of a proof engine—a custom built verification engine which can be operated by another program through an Application Programming Interface (API). A proof engine can be built by a system developer using the toolkit provided by the project. A proof engine is based upon

* Work funded ESPRIT Framework IV Grant LTR 26241
[1] The case study is available from `http://www.collins-peak.net/p-excel/`

the functionality of a theorem prover with additional capabilities provided by 'plugins' formed from existing, off-the-shelf, tools. The toolkit includes a set of libraries based on a language-independent specification, the PROSPER Integration Interface (PII), for communication between components of a final system. The theorem prover's command language is treated as a kind of scripting or glue language for managing plugin components and orchestrating the proofs.

The PII consists of several parts. There is a datatype for communication of data between components of a system which includes the language of higher order logic used by the HOL system[2] and so any formula expressible in higher order logic can be passed between components. There is support for installing procedures in an API and calling them remotely. There are also parts for managing low level communication, which are largely invisible to an application developer. The PII is currently implemented in ML, C, Java, Python, λProlog and ADA.

Proof engines are constructed on top of a small subset of HOL, called the *Core Proof Engine*. This consists of theorems, inference rules for higher order logic and an ML implementation of the PII. A developer can write extensions to the Core Proof Engine and place them in an API to form a custom proof engine. When incorporating a proof engine into an application the developer calls the customised API through the PII.

3 Microsoft Excel

Excel is a spreadsheet package marketed by Microsoft [4]. Its basic constituents are rows and columns of cells into which either values or formulae may be entered. Formulae refer to other cells, which may contain either values or further formulae.

Users of Excel are likely to have no interest in using or guiding mathematical proof, but they do want to know that they have entered formulae correctly. They therefore have an interest in 'sanity checking functions' that they can use to reassure themselves of correctness. This made Excel suited as a case study since the users have a notion of formulae and correctness, all that needs to be hidden is the proof. Another advantage is that Excel was designed to allow new functionality to be added and although its developers were not concerned with verification there is support for calling external tools.

As a simple example, the authors undertook to incorporate a sanity checking function into Excel. We chose to implement an equality checking function which would take two cells containing formulae and attempt to determine whether these formulae were equal for all possible values of the cells to which they refer.

Simplifying assumptions were made for the case study. The most important were that cell values were only natural numbers or booleans and that only a small subset of the functions available in Excel (some simple arithmetical and logical functions) appeared in formulae. Given these assumptions, less than 150 lines of code were needed to produce a prototype. This prototype handled only a small range of formulae decidable by linear arithmetic or propositional logic decision procedures, but it demonstrated the basic functionality.

4 Architecture

The main difficulty in the system was that Excel is Windows based and expects Microsoft's Component Object Model (COM) to be used for communication between processes, whereas the Prosper toolkit had been developed for UNIX machines[2] and uses sockets for communications between components.

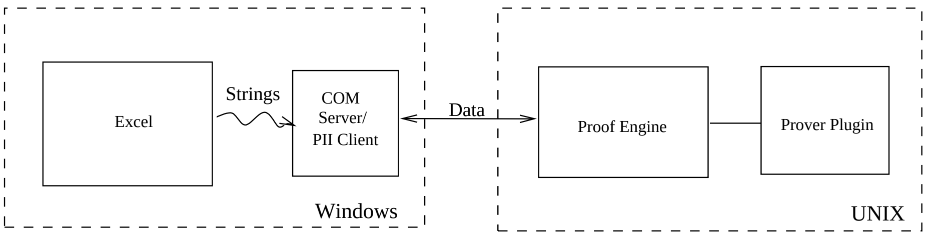
Several possible solutions to this problem were considered including implementing the PII in Visual Basic and using internet sockets to let Excel communicate with a proof engine. We did not take this approach because our aim was to show that theorem proving technology can be incorporated into applications in as natural a way as possible. For Excel this meant making the functionality of the Prosper tools available as a COM server.

The Prosper COM server was implemented in Python, a dynamically typed, object oriented scripting language which supports both COM and sockets. The server consists of two parts, the python implementation of the PII and the additional code described below which is specific to this example.

The remaining decision was where to convert Excel's formulae, which we access as strings, into terms. This requires some type inference but is simple to do and could have been written in either the Python or Visual Basic components. This was done in Python since it was the preferred language of the authors.

From the Excel side the Python component is a COM server which makes available a small number of functions that Excel can call. The use of a UNIX based theorem prover is not visible to Excel. From the proof engine side the Python component behaves like any other application calling the proof engine using the PII. The use of Excel is not visible to the theorem prover.

A view of the current (2 operating system) architecture is shown below.



5 Custom Proof Engine

The initial custom proof procedure is very simple-minded. It uses a linear arithmetic decision procedure provided by HOL and a propositional logic plugin (based on Prover Technology's proof tool [6, 5]) to decide the truth of formulae. While the approach is not especially robust, it is strong enough to handle many formulae.

[2] It is expected that a future version will be ported to Windows.

The additional code required to create this custom proof procedure is very small (approx. 45 extra lines of ML were needed). All the verification code used already existed either in HOL or the plugin, the new code concentrated on gluing together the decision procedures and deciding which should be used.

A proof engine which could handle a wider range of formulae would require more work. It is possible that more decision procedures could be used to provide this, for instance we could exploit HOL's simplifier. Alternatively it might prove necessary to implement some specialised theorem proving algorithms. This would also be possible using the PROSPER toolkit.

6 Python COM Component

The main piece of code developed for this system is the Python implementation of the PII. This was simple to write, partly since the structure is similar to the existing Java PII, and partly because this is the sort of application for which Python was designed. The code makes use of dynamic typing and other features of the language to provide a compact and natural implementation of the PII. Although written for this one application, the Python implementation makes available the objects of the PII, and hence the functionality of the PROSPER tools to any language that supports COM.

In addition to the PII implementation the COM component contains some additional code specific to this example. This first parses the strings to logical terms. This assumes that the semantics of the operators is the same in Excel and HOL. The terms are then passed on to the proof engine. It returns the result of the proof attempt as true, false, or 'unable to decide', which is displayed in the cell containing the `ISEQUAL` formula. This result can be used by other cells and will be automatically recomputed if necessary.

7 Excel Macro

We wrote a visual basic function, `ISEQUAL`, using Excel's macro editor. Once written, it automatically appears in Excel's function list as a User Defined Function and can be used in a spreadsheet like any other function. `ISEQUAL` takes two cell references as arguments. It recursively extracts the formulae contained in the cells as strings (support for this already exists in Excel) and passes them on to the Python object. The macro consists of about 30 lines of Visual Basic code.

8 Conclusions

There are numerous Add-Ins to Excel many of which, unsurprisingly, extend its mathematical ability. The Maple 6 Add-In provides computer algebra techniques to Excel spreadsheets. Interval Solver [3] extends Excel with Interval Constraint Solving to allow spreadsheet users to reason with incomplete and uncertain information. We believe that theorem proving could also have a role to play in this field. We have demonstrated that the PROSPER approach provides a framework in which this could be done.

We were surprised and pleased with the ease that a very basic prototype of verification support for Excel could be produced. It took two programmers, neither of whom had any experience with Visual Basic, Python or COM only 48 hours to get to the point where Excel was able to prove the commutativity of plus. While this may seem uninteresting, the reordering of the mathematical operators in large formulae is exactly the kind of lightweight sanity check that may appeal to users. Extending the system to handle more arithmetic and logical operators was easy and the system has been tested on a range of linear arithmetic and spreadsheet style examples. The system could be extended further and more complex and interesting proof strategies could be programmed.

The system is a proof of concept of the claim made by the PROSPER project that their toolkit would enable the embedding of verification into applications not designed with it specifically in mind. The only significant piece of new code is the Python port of the PII which is a general purpose component that could be used for other systems.

Adding even limited theorem proving functionality by programming a procedure from scratch instead of using existing tools would have taken much longer, as would interfacing to a theorem prover without using the PROSPER tools.

The use of two operating systems is not ideal but could be removed if the PROSPER tools were ported to Windows. The current setup would be reasonable in a networked setting with many copies of Excel accessing one proof engine.

The embedding of verification into Excel also serves as an example of the concepts of "lightweight" theorem proving and the "invisible" use of verification. Here all the infrastructure is invisible to the user who simply gets an extra function available in Excel.

References

1. L. A. Dennis, G. Collins, M. Norrish, R. Boulton, K. Slind, G. Robinson, M. Gordon and T. Melham, The PROSPER Toolkit, *TACAS 2000*, to appear. 2000.
2. M. J. C. Gordon and T. F. Melham (eds), *Introduction to HOL: A theorem proving environment for higher order logic*, Cambridge University Press, 1993.
3. E. Hyvö nen and S. De Pascale, A New Basis for Spreadsheet Computing: Interval Solver^{TM} for Microsoft Excel. *Proceedings of 16th National Conference on Artificial Intelligence and 11th Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI-99)*, AAAI Press / The MIT Press, pp. 799–806, 1999.
4. Microsoft Corporation, *Microsoft Excel*, `http://www.microsoft.com/excel`.
5. M. Sheeran and G. Stålmarck, A tutorial on Stålmarck's proof procedure for propositional logic. *The Second International Conference on Formal Methods in Computer-Aided Design*, Lecture Notes in Computer Science 1522, Springer-Verlag, pp. 82–99, 1998.
6. G. Stålmarck and M. Säflund, Modelling and Verifying Systems and Software in Propositional Logic. *Proceedings of SAFECOMP '90*, Pergamon Press, pp. 31–36, 1990.