Prosper
An Investigation into Software Architecture for Embedded Proof Engines

Thomas F. Melham

Department of Computing Science
University of Glasgow
Glasgow, Scotland, G12 8QQ
tfm@dcs.gla.ac.uk

Abstract. Prosper is a recently-completed ESPRIT Framework IV research project that investigated software architectures for component-based, embedded formal verification tools. The aim of the project was to make mechanized formal analysis more accessible in practice by providing a framework for integrating formal proof tools inside other software applications. This paper is an extended abstract of an invited presentation on Prosper given at FroCoS 2002. It describes the vision of the Prosper project and provides a summary of the technical approach taken and some of the lessons learned.

Prosper [46] is a 24 person-year LTR project supported under the ESPRIT Framework IV programme and formally completed in May 2001. The project ran for three years and conducted a relatively large-scale research investigation into new software architectures for component-based, embedded formal verification tools.

The project was a collaboration between the Universities of Glasgow, Cambridge, Edinburgh, Tübingen and Karlsruhe, and the industrial partners IFAD and Prover Technology. Glasgow was the project Coordinator, as well as the main development site for the core Prosper software infrastructure.

1 Embedded, Component-Based Verification

The starting point for Prosper was the proposition that mechanized formal verification might be made more accessible to non-expert users by embedding it, indeed *hiding* it, as a component inside the software applications they use. Ideally, reasoning and proof support would be made available to the end-user by encapsulating it within the interaction model and interfaces they already know, rather than making them wrestle directly with theorem provers, model checkers, or other arcane software.

By contrast, the practical results of much current formal reasoning research are typically embodied in stand-alone tools that can be operated only by experts who have deep knowledge of the tool and its logical basis. Verification tools are therefore often not well integrated into established design flows—e.g. into the CAD or CASE tool environments currently used for hardware design or software development.

Good evidence that this imposes serious barriers to adoption of formal reasoning by industry can be found in the arena of formal verification for hardware design. By far the most successful method is formal equivalence checking (e.g. with BDDs), where the technology is relatively push-button and well integrated by electronics design tool vendors into their normal CAD tool flows [10]. On the other hand, only very few, well-resourced, early adopters have been making effective use of model checkers or theorem provers [19,43] and wider deployment of these in future is by no means certain.

But a developer who does wish to incorporate verification inside another application, for example a CAD or CASE tool, faces a difficult choice between creating a verification engine from scratch and adapting existing tools. Developing a new verification engine is time-consuming and means expensive re-implementation. But existing tools are rarely suitable as components that can be customized for a specific verification role and patched into other programs.

In summary, at the time the Prosper research programme was being devised (circa 1996) many promising formal reasoning tools existed, but these were typically

– not integrated into other applications,
– internally monolithic,
– driven through user-orientated interfaces, and
– operable only by expert users.

Prosper's idea was to experiment with a framework in which formal verification tools might instead be

– integrated as embedded 'proof engines' inside other applications,
– built from components,
– driven by other software through an API, and
– operable by ordinary application-domain users, by giving user-oriented guidance.

The Prosper project investigated this proposal by researching and developing a software *Toolkit* [21], centred around an open proof tool architecture, that allows an expert to easily and flexibly assemble customized software components to provide embedded formal reasoning support inside applications. The project originally had mainly CAD or CASE applications in mind, but its results were not really specialized to these. Indeed, one early experiment was to embed proof support within Microsoft Excel.

The primary concern of the project was *not* the logical aspects of integration or combining systems, and the Prosper Toolkit has no special mechanisms for ensuring the soundness of translations between the logical data of different tools. Of course this is important, but it was not emphasized in the project because other work, such as the OMRS project [29], was developing systematic frameworks for treating soundness.

The remainder of this abstract gives a sketch of the technical approach taken in Prosper and a brief account of some of the general lessons learned. Of course, other researchers have had similar ideas. There is a growing research literature on combinations of proof tools, tool integration architectures, and design tools with embedded verification—not least in the proceedings of the FroCoS workshop series [9,28,36]. A brief list of some related work is given in section 5

2 Technical Approach

Central to Prosper's vision is the idea of a *proof engine*, a custom-built verification software component which can be operated by another program through an Application Programming Interface (API). This allows the embedded formal reasoning capability to be operated in a machine-oriented rather than human-oriented fashion.

A proof engine is expected to be specially tuned to the verification needs of the application it supports. Application requirements in general are, of course, unpredictable and in any case will vary widely. Prosper therefore does not supply just one 'general-purpose' proof engine. Instead, the project has developed a collection of software libraries, called the Prosper *Toolkit*, that enables a system developer to build custom proof engines as required.

Every Prosper proof engine is constructed from a (possibly quite minimal) deductive theorem prover, with additional capabilities provided by 'plugins' created from existing, off-the-shelf, tools such as model checkers or SAT solvers. The theorem prover's command language is regarded as a kind of scripting language for managing the plugin components and orchestrating proofs. The Toolkit includes libraries, currently supporting the C, ML and Java programming languages, for implementing data and control communications between the components of a final system. A standard for this is also documented in a language-independent specification, called the Prosper *Integration Interface* (PII), which could be implemented in other languages.

A theorem prover is placed at the centre of the architecture because this comes with ready-made concepts of logical term, theorem, and goal—essentially all the formal language infrastructure needed for managing verifications. A side benefit is that all the functionality in the theorem prover (libraries of procedures, tactics, logical theories, and so on) becomes available to a developer for inclusion in their custom proof engine. But this does not prevent the theorem proving part of a Prosper proof engine being very lightweight, if desired.

The Prosper Toolkit has been implemented around HOL98, a modern descendant of the HOL theorem prover [31]. HOL98 is highly modular, which suits the Prosper approach of building up a proof engine from components (be they HOL libraries or external plugins). It also contains numerous sophisticated automatic proof procedures. HOL's command language is the functional programming language ML [42], extended with datatypes for the abstract syntax of

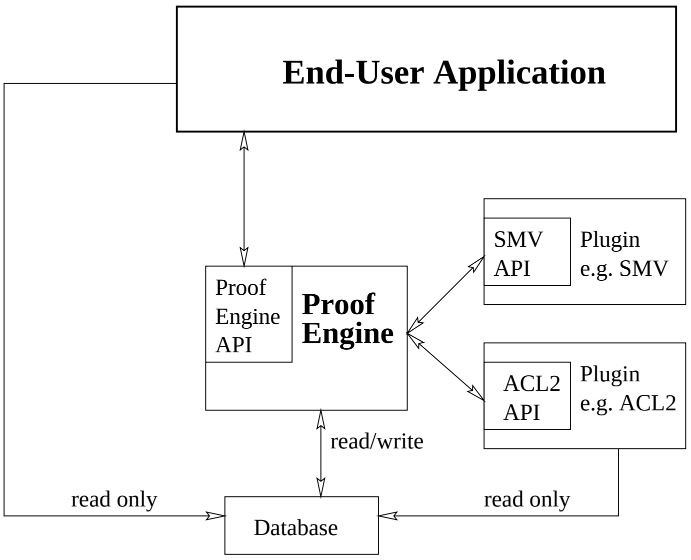

Fig. 1. A system built with the PROSPER Toolkit.

logical data and functions to support proof construction. This gives a developer a full programming language in which to create bespoke verification procedures. Proof procedures programmed in the proof engine are offered to client applications in an API.

The native formal logic of HOL is classical higher-order logic [15]. This is also supported by the PII communications infrastructure, so any formula expressible in higher-order logic can be passed between components. Many applications and plugins operate with logical data that is either already a subset of higher-order logic (e.g. first-order or propositional logic) or embeddable in it (e.g. CTL or other temporal logics [4,38]), so communication with these tools is directly supported.

The Toolkit provides code to construct several plugins based on particular external tools. These include a version of the SMV model checker [38], a version of the Prover Plug-In SAT solver of Prover Technology [50], and the circuit analysis tool AC/3 [33]. To complement these plugins, the PROSPER project provided a more tightly integrated BDD package in the implementation of ML used for proof engines [30]. Third party plugins have also been developed from ACL2 [35,51], Gandalf [34], and SVC [53]. Finally, the Toolkit includes a separate database component that duplicates the internal logical state of the theorem prover, so that plugins and applications can access theory-related data while the proof engine is busy.

The application, proof engine, database, and plugins are components integrated to produce the final system. A typical example is shown in Figure 1. In the current prototype, all these components are also separate processes that communicate in the uniform manner specified by the PII. The Prosper architecture also includes a separate *monitor* process not shown in the diagram. This allows components to interrupt each other and the output of servers to be redirected to be handled by clients. It also includes a *name server* facility whereby components may be requested by name instead of location. This is achieved by means of a configuration file that contains component names together with information about their initialization scripts and configurations.

For a fuller account of the technical approach taken in the Prosper Toolkit, including several illustrative examples, see [21]. Much more technical detail can also be found in the user guide [22].

3 Case Studies

Prosper researchers undertook three main case studies to demonstrate the concept of embedded proof engines and test the Toolkit. The first was an early experiment in integrating a function that used formal proof into Microsoft Excel [41]. The second was a much larger development to add verification capabilities to the VDM-SL Toolbox [24], a CASE tool for VDM specifications marketed by project partner IFAD. The third was a hardware verification tool, driven by a novel natural language and graphical interface, that allows specifications to be checked by a proof engine that incorporates a model checker.

Excel Example. Excel is a spreadsheet package marketed by Microsoft [41]. Ordinary users are unlikely to be directly interested in mathematical proof, but they do want to check they have entered their spreadsheet formulas correctly. As a simple case study, the Prosper Toolkit developers undertook to incorporate a 'sanity checking' function into Excel that users could employ to reassure themselves of correctness [18].

The function supplied tests the equality of the contents of two spreadsheet cells, not by comparing their current values, but by trying to verify that the two formulas underlying these cells are equal for all possible values of the input cells occurring in them. The idea is to provide ordinary users with a way of checking for errors in their spreadsheets, in cases where two cells calculated in different ways are expected always to produce the same value—for example, in financial bookkeeping applications.

When this operation is included in a spreadsheet formula, it invokes a simple proof engine to compute its result. The proof engine uses term-rewriting and a linear arithmetic decision procedure in HOL, as well as an external SAT solver to try to verify equivalence.

The prototype handled only very simple formulas, in which a small subset of the (natural number or Boolean) functions available in Excel could appear. But given this simplification, fewer than 150 lines of code and only a few days work were needed to produce a prototype, demonstrating that the basic functionality was achievable very easily using the Prosper Toolkit.

CASE Tool Application. The IFAD VDM-SL Toolbox [24] is a software design tool that supports the specification language VDM-SL. Its capabilities include syntax checking and type checking of specifications and code generation in C++ and Java. As a major case study within PROSPER, IFAD researchers worked together with other project partners to develop and integrate a proof engine for the VDM-SL Toolbox for discharging proof obligations generated by VDM-SL type invariants.

Type invariants are undecidable in general, but in practice many of them can be dismissed by simplification and term-rewriting in combination with bounded first-order logic decision procedures. A PROSPER proof engine that supplied this reasoning capability was developed by integrating theorem proving in HOL with BDDs and the Prover Plug-In SAT solver. The resulting heuristic was found by IFAD to perform well on realistic industrial test cases.

An additional requirement was if an automatic proof attempt fails, a user must be able to intervene and guide a proof by hand. The VDM-SL Toolbox application therefore also involved a fairly large-scale development to steer the proof engine through an interactive proof management interface in the VDM-SL Toolbox.

Hardware Verification Workbench Application. The second major PROSPER case study was a Hardware Verification Workbench that served as a research platform for developing and evaluating new methods in formal hardware verification. It was designed to use external verification back-ends, rather than implement all its own proof procedures. Communication between the Hardware Verification Workbench and the proof back-ends was achieved via the PROSPER Integration Interface (PII).

One experiment done with this platform was the development of a natural language interface [32] to the Hardware Verification Workbench. This translates statements about circuits, in the normal technical language of engineers, into temporal logic formulas that a model checking plugin can verify. Of course, natural language specifications may be ambiguous. To disambiguate, timing diagrams (waveforms) are generated from output of the model checker and presented back to the engineer. In keeping with the PROSPER aim of 'hiding' the proof tool, users never have to run the model checker directly, or even see temporal logic formulas—they work only with already familiar things, namely natural language and timing diagrams.

4 Some Conclusions from PROSPER

The premise of the PROSPER project was that mechanized formal analysis could be made more accessible to users by integrating it into applications. Moreover, embedded formal proof will gain widespread use, or at least be widely experimented with, only if verification tools can be easily customized, combined, and integrated.

The research results obtained support these propositions. An effective infrastructure for building and integrating embedded, component-based proof engines was found to be technically feasible; the Prosper Toolkit represents one possible prototype. And the general idea of embedded custom reasoning engines, giving ordinary users the power of proof, is promising enough to merit much more investigation on applications.

The experiment with Excel was very encouraging. It showed that infrastructure of the kind proposed can indeed make it easy to embed formal verification tools into other applications. The VDM-SL CASE tool example was also encouraging. The Prosper Toolkit gave a much more controllable and effective integration of verification into IFAD's existing CASE tool product than was found in the broadly negative experience of previous, ad-hoc experiments [3]. The Hardware Verification Workbench case study also provided an intriguing and novel example, showing that formal notations could be completely hidden behind a user-oriented interface.

Prosper aimed from the start for a single infrastructure covering both hardware and software design tool applications. In the end, the Toolkit is perhaps better suited to software design tools (CASE tools) and interactive general applications (like Excel) than the current generation of CAD tools. CAD tool flows are typically compilation or batch processing oriented, with large amounts of design data passing between tools through disk files in numerous different formats. The kind of fine-grained communications and interaction supported by the PII is less relevant here.

The Prosper architecture places a theorem prover, implemented in ML, at the centre of every proof engine. The reasons given for this in Section 2 are sound, but the architecture is in some respects also quite awkward. ML has all the usual advantages of a functional programming language—conciseness, strong typing, and so on—and this does make it well suited as a scripting language for *orchestrating verification strategies*. But the single-threaded nature of ML[1] is something of a disadvantage in a scripting language for *coordinating the invocation of plugins and communications between them*. For example, because ML (and hence HOL) is single-threaded, the architecture needed a separate database process to allow asynchronous access by plugins to logical data.

The separate 'monitor' process mentioned in section 2 is also related to this. Extra processes were added to the architecture mainly to support two seemingly elementary capabilities—namely, the ability to interrupt a proof engine and its plugins while they are working, and the ability to divert the output streams from plugins for handling by other processes. Providing these features, expressly requested by the VDM-SL application developers at IFAD, required a surprising amount of quite tricky distributed-systems programming.

A major outcome of the project is the language-independent specification of the Prosper Integration Interface and the implementations for ML, C, and Java. Alternative transport mechanisms were considered for logical data, such as the Extensible Markup Language (XML). Standard component architectures, such as CORBA [44], were also considered for low-level communications. Both were rejected early on in the project in favour of a custom solution, which was

[1] Or at least the ML underlying HOL98, namely Moscow ML.

felt in advance to be the more 'lightweight' approach. In retrospect, it may have been better to pursue standard solutions. On the other hand, it is by no means certain that much of the eventual Toolkit infrastructure would not have to have been built anyway.

Our experience with plugins was that many existing tools have command-line or user interfaces that are very difficult to separate from the underlying proof capabilities. This makes them hard to wrap up as PROSPER plugin components. It helps enormously if a tool already has a distinct and identifiable API—that is if its reasoning functionality is available through a well-documented and coherent set of entry points into the code. Good examples of such tools include the Prover Plug-In [50] and NuSMV [16].

On the logical side, the simply-typed higher-order logic implemented in HOL and supported by the PII was found to be adequate for the examples considered. This is not really a surprise—it has long been known how to embed a large range of other formalisms in this logic. Of course, PROSPER looked at only a limited range of plugins and applications; there are doubtless many other settings in which a more expressive type system (e.g. predicate subtypes as in PVS) would be strictly necessary.[2] But we would expect the infrastructure to extend quite naturally to this.

An important issue that was not investigated in depth was how to provide support for producing and presenting counter-examples in case of failed proof attempts. PROSPER focused on applications where automatic proof is possible, but this is rather idealistic. In many applications, failure of proof is likely to be frequent—not least while debugging specifications. The user then needs good feedback on why the proof attempt failed, in a form with which the user is familiar. A systematic treatment of this process and some general infrastructure support would be very valuable.

5 Related Work

Research related to PROSPER includes work on combining proof tools, tool integration architectures, and design tools with embedded verification. Only a brief list of pointers to some of the most relevant work is given here. An analysis of the relation of this research to PROSPER's approach and results can be found in an extended version of [21], to appear in the *International Journal on Software Tools for Technology Transfer.*

Combining Proof Tools. There is now a fairly large literature on combining decision procedures, and in particular model checkers, with theorem proving. The aim of this research is typically to increase the level of automation of theorem provers or extend the reach of model checkers in the face of fundamental capacity limits (or both). Early experiments include links to model-checking based on

[2] Partial functions in VDM seem a case in point. But these were avoided by doing some logical pre-processing within the CASE tool client itself—so HOL's logic of total functions was adequate for this example.

embeddings of the modal mu-calculus in the logics of the HOL and PVS theorem provers [4,47].

A notable example of current work is Intel's Forte system [1,2], which intimately combines Symbolic Trajectory Evaluation model checking [49] and theorem proving in a single framework. This has been used very effectively for industrial-scale formal hardware verification [45]. Another approach being investigated by Ken McMillan at Cadence Berkeley Labs is to extend the top-level of a model-checker with proof rules for abstraction and problem decomposition [39,40].

A useful summary of other recent work on combining model checking with theorem proving is given in Tomás Uribe's invited talk at FroCoS 2000 [54].

Integration Architectures. PROSPER focused more on infrastructure for tool integration in general than on developing particular tool combinations. (An exception is the work on linking BDDs into HOL98 [30].) Some other projects which also provide a generic framework for the integration of tools are listed below.

MATHWEB is a framework for distributed mathematical services provided by reasoning systems such as resolution theorem provers and computer algebra systems [25]. A special service for storing knowledge, MBASE [26], allows theory information to be shared between other services.

Ω-ANTS combines interactive and automated theorem provers using an agent-based approach [13]. Its blackboard architecture and other agent-oriented features provide flexible interaction between components.

ILF is a framework for integrating interactive and automated provers that places special emphasis on a good user interface for the automated provers [20]. The provers can be distributed and Prolog is used as a scripting language, much as ML is used in PROSPER.

TECHS is another framework which enables automated provers for first-order logic to cooperate by exchanging logical information [27].

ETI, the Electronic Tool Integration platform [52], aims to support easy and rapid comparison of tools that do similar jobs as well as rapid prototyping of combinations of tools. ETI has its own scripting and communication language, HLL, which acts much like PROSPER's combination of ML and the PII.

OMRS aims to develop an *open* architecture for integration of reasoning systems. The architecture covers three aspects: logic [29], control strategies [17], and interaction mechanisms [6].

LBA, the *Logic Broker Architecture* [7,8], is a CORBA-based infrastructure for the integration of reasoning systems. It provides location transparency, fowarding of requests to reasoning components via a registration/subscription mechanism, and provable soundness guarantees. The LBA was initially designed to be OMRS-based but has evolved into an independent entity.

SAL (Symbolic Analysis Laboratory) is a new collaborative effort that provides a framework for combining different tools to calculate properties of concurrent systems. One instance includes the PVS theorem prover as a major component [11].

Design Tools with Embedded Verification. Braun et al. argue that for formal techniques to be useful they must be integrated into the design process [14]. A primary aim of PROSPER was to support this by making it easier to link verification tools into the CASE and CAD tool environments for current software and hardware development processes. The two major PROSPER case studies described in section 3 looked at target applications of both kinds, embedding proof engines into both a commercial CASE tool and a formal hardware verification platform.

Some other projects also looking at linking formal techniques into design tools and the design process are listed below.

UniForM is a project that aims to encourage the development of reliable industrial software by enabling suitable tool-supported combinations of formal methods. The UniForM Workbench [37] is a generic framework that can be instantiated with specific tools. The project has produced a workbench for software design that gives access to the Isabelle theorem prover plus other verification tools through their command lines.

Extended Static Checking (ESC) is a software development tool using embedded verification developed at the Compaq Systems Research Center [23]. ESC uses cooperating decision procedure technology first developed in the early 1980s to analyse Java programs for static errors.

KeY is a relatively substantial research effort that aims to bridge the gap between object-oriented software engineering methods and deductive verification. The KeY system integrates a commercial CASE tool with an interactive verification system and automated deduction [5]. It aims to provide an industrial verification tool that benefits even software engineers who have little experience of formal methods.

InVeSt integrates the PVS theorem prover with the SMV model checker [12]. The combination is used as a ‘proof engine’, in the sense that it discharges verification conditions generated by another program external to the theorem prover.

As far as I am aware, no project other than PROSPER aims specifically to provide a general framework to support the integration of existing components with the view to producing an embeddable, customised proof engine. The closest in scope and aims is the Logic Broker Architecture.

Acknowledgments

I thank the organizers of FroCoS 2002 for their invitation to speak at the conference in Santa Margherita. Richard Boulton and Alessandro Armando kindly read and commented on an early draft of this abstract. Warm thanks are also due to the PROSPER Management Board and all the individual members of the PROSPER research team. The project could not have succeeded without their hard work and dedication to the project’s vision. PROSPER was supported in part by the Commission for European Communities as LTR Project 26241 under the ESPRIT Framework IV programme.

References

1. M. D. Aagaard, R. B. Jones, and C.-J. H. Seger, 'Combining theorem proving and trajectory evaluation in an industrial environment,' in *ACM/IEEE Design Automation Conference, Proceedings* (1998), pp. 538–541.
2. M. D. Aagaard, R. B. Jones, and C.-J. H. Seger, 'Lifted-fl: A pragmatic implementation of combined model checking and theorem proving', in *Theorem Proving in Higher Order Logics*, edited by Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, Lecture Notes in Computer Science, vol. 1690 (Springer-Verlag, 1999), pp. 23–340.
3. S. Agerholm, 'Translating Specifications in VDM-SL to PVS', in *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLs'96: Turku, August 1996: Proceedings*, edited by J. von Wright, J. Grundy, and J. Harrison Lecture Notes in Computer Science, vol. 1690 (Springer-Verlag, 1999), pp. 1–16.
4. S. Agerholm and H. Skjødt, *Automating a model checker for recursive modal assertions in HOL.* Technical Report DAIMI IR-92, Computer Science Department (Aarhus University, 1990).
5. W. Ahrendt, T. Baar, B. Beckert, M. Giese, E. Habermalz, R. Hähnle, W. Menzel, and P. H. Schmitt, 'The KeY approach: Integrating object oriented design and formal verification', in *Proceedings of the 8th European Workshop on Logics in AI (JELIA)*, edited by M. Ojeda-Aciego, I. P. de Guzmán, G. Brewka, and L. M. Pereira (eds), Lecture Notes in Computer Science, vol. 1919 (Springer-Verlag, 2000), pp. 21–36,
6. A. Armando, M. Kohlhase, and S. Ranise, 'Communication protocols for mathematical services based on KQML and OMRS', in *Symbolic Computation and Automated Reasoning: The CALCULEMUS-2000 Symposium (Proceedings of the Eighth Symposium on the Integration of Symbolic Computation and Mechanized Reasoning)*, edited by M. Kerber and M. Kohlhase (A. K. Peters Ltd., 2001).
7. A. Armando and D. Zini, 'Interfacing computer algebra and deduction systems via the Logic Broker Architecture', in *Symbolic Computation and Automated Reasoning: The CALCULEMUS-2000 Symposium (Proceedings of the Eighth Symposium on the Integration of Symbolic Computation and Mechanized Reasoning)*, edited by M. Kerber and M. Kohlhase (A. K. Peters Ltd., 2001).
8. A. Armando and D. Zini, 'Towards Interoperable Mechanized Reasoning Systems: the Logic Broker Architecture', in *Proceedings of the AI*IA-TABOO Joint Workshop 'Dagli Oggetti agli Agenti: Tendenze Evolutive dei Sistemi Software'*, 29–30 May 2001, Parma, Italy (2001), pp. 70–75.
9. F. Baader and K. U. Schulz, editors, *Frontiers of Combining Systems: First International Workshop, Munich, March 1996*, Applied Logic Series, vol. 3 (Kluwer Academic Publishers, 1996).
10. L. Bening and H. Foster, *Principles of Verifiable RTL Design* (Kluwer, 2000).
11. S. Bensalem, V. Ganesh, Y. Lakhnech, C. Muñoz, S. Owre, H. Ruess, J. Rushby, V. Rusu, H. Saïdi, N. Shankar, E. Singerman, and A. Tiwari, 'An overview of SAL', in *Proceedings of the Fifth NASA Langley Formal Methods Workshop, June 2000* (Williamsburg, 2000).
12. S. Bensalem, Y. Lakhnech, and S. Owre, 'InVeSt: A tool for the verification of invariants', in *Proceedings of the 10th International Conference on Computer Aided Verification (CAV'98)*, edited by A. J. Hu and M. Y. Vardi, Lecture Notes in Computer Science, vol. 1427 (Springer-Verlag, 1998), pp. 505–510.

13. C. Benzmüller and V. Sorge, '*Ω*-ANTS — An open approach at combining interactive and automated theorem proving' in *Symbolic Computation and Automated Reasoning: The CALCULEMUS-2000 Symposium (Proceedings of the Eighth Symposium on the Integration of Symbolic Computation and Mechanized Reasoning)*, edited by M. Kerber and M. Kohlhase (A. K. Peters Ltd., 2001).
14. P. Braun, H. Lötzbeyer, B. Schätz, and O. Slotosch, 'Consistent integration of formal methods', in *Tools and Algorithms for the Construction and Analysis of Systems: 6th International Conference, TACAS 2000: Berlin, March/April 2000: Proceedings*, edited by S. Graf and M. Schwartzbach, Lecture Notes in Computer Science, vol. 1785 (Springer-Verlag, 2000), pp. 48–62.
15. A. Church, 'A Formulation of the Simple Theory of Types', *Journal of Symbolic Logic*, vol. 5 (1940), pp. 56–68.
16. A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri, 'NUSMV: a new Symbolic Model Verifier', in *Proceedings of the Eleventh Conference on Computer-Aided Verification (CAV'99)*, edited by N. Halbwachs and D. Peled, Lecture Notes in Computer Science, vol. 1633 (Springer-Verlag, 1999), pp. 495–499.
17. A. Coglio, 'The control component of OMRS: NQTHM as a case study', in *Proceedings of the First Workshop on Abstraction, Analogy and Metareasoning*, IRST (Trento, 1996), pp. 65–71.
18. G. Collins and L. A. Dennis, 'System description: Embedding verification into Microsoft Excel', in *Proceedings of the 17th International Conference on Automated Deduction: CADE-17*, edited by D. McAllester, Lecture Notes in Artificial Intelligence, vol. 1831 (Springer-Verlag, 2000), pp. 497–501.
19. B. Colwell and B. Brennan, 'Intel's Formal Verification Experience on the Willamette Development', in *Theorem Proving in Higher Order Logics: 13th International Conference, TPHOLs 2000: Portland, August 2000: Proceedings*, edited by M. Aagaard and J. Harrision, Lecture Notes in Computer Science, vol. 1869 (Springer-Verlag, 2000), pp. 106–107.
20. B. I. Dahn, J. Gehne, T. Honigmann, and A. Wolf, 'Integration of automated and interactive theorem proving in ILF', in *Proceedings of the 14th International Conference on Automated Deduction (CADE-14)*, edited by W. McCune, Lecture Notes in Artificial Intelligence, vol. 1249 (Springer-Verlag, 1997), pp. 57–60.
21. L. A. Dennis, G. Collins, M. Norrish, R. Boulton, K. Slind, G. Robinson, M. Gordon, and T. Melham, 'The PROSPER Toolkit', in *Tools and Algorithms for the Construction and Analysis of Systems: 6th International Conference, TACAS 2000: Berlin, March/April 2000: Proceedings*, edited by S. Graf and M. Schwartzbach, Lecture Notes in Computer Science, vol. 1785 (Springer-Verlag, 2000), pp. 78–92. An extended version of this paper is to appear in the *International Journal on Software Tools for Technology Transfer.*
22. L. Dennis, G. Collins, R. Boulton, G. Robinson, M. Norrish, and K. Slind, *The PROSPER Toolkit, version 1.4*, Part of deliverable D3.5, ESPRIT LTR Project PROSPER (26241), Department of Computing Science, University of Glasgow (April, 2001). Available as `prosper1-4.ps.gz` at `www.dcs.gla.ac.uk/prosper/toolkit/`.
23. D. L. Detlefs, K. R. M. Leino, G. Nelson, and J. B. Saxe, *Extended static checking*, Research Report 159, Compaq Systems Research Center, Palo Alto (December, 1998).
24. J. Fitzgerald and P. G. Larsen, *Modelling Systems: Practical Tools and Techniques in Software Development* (Cambridge University Press, 1998).

25. A. Franke, S. M. Hess, C. G. Jung, M. Kohlhase, and V. Sorge, 'Agent-Oriented Integration of Distributed Mathematical Services', *Journal of Universal Computer Science*, vol. 5 (1999), pp. 156–187.
26. A. Franke and M. Kohlhase, 'System description: MBASE, an open mathematical knowledge base', in *Proceedings of the 17th International Conference on Automated Deduction: CADE-17*, edited by D. McAllester, Lecture Notes in Artificial Intelligence, vol. 1831 (Springer-Verlag, 2000), pp. 455–459.
27. D. Fuchs and J. Denzinger, *Knowledge-based cooperation between theorem provers by TECHS*, SEKI-Report SR-97-11 (University of Kaiserslautern, 1997).
28. D. M. Gabbay and M. de Rijke, editors, *Frontiers of Combining Systems 2*, Studies in Logic and Computation Series (Research Studies Press, 2000).
29. F. Giunchiglia, P. Pecchiari, and C. Talcott, 'Reasoning theories: Towards an architecture for open mechanized reasoning systems', in *Frontiers of Combining Systems: First International Workshop, Munich, March 1996*, edited by F. Baader and K. U. Schulz, Applied Logic Series, vol. 3 (Kluwer Academic Publishers, 1996), pp. 157–174.
30. M. Gordon and K. F. Larsen, *Combining the Hol98 proof assistant with the BuDDy BDD package*, Technical Report 481, University of Cambridge Computer Laboratory (December, 1999).
31. M. J. C. Gordon and T. F. Melham, editors, *Introduction to HOL: A theorem proving environment for higher order logic* (Cambridge University Press, 1993).
32. A. Holt, E. Klein, and C. Grover, 'Natural language specifications for hardware verification', *Language and Computation*, vol. 1 (2000), pp. 275–282. Special issue on ICoS-1.
33. D. W. Hoffmann and T. Kropf, 'Automatic error correction of large circuits using boolean decomposition and abstraction', in *Correct Hardware Design and Verification Methods: 10th IFIP WG10.5 Advanced Research Working Conference: Bad Herrenalb, September 1999: Proceedings*, edited by L. Pierre and T. Kropf, Lecture Notes in Computer Science, vol. 1703 (Springer-Verlag, 1999), pp. 157–171.
34. J. Hurd, 'Integrating Gandalf and HOL', in *Theorem Proving in Higher Order Logics*, edited by Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, Lecture Notes in Computer Science, vol. 1690 (Springer-Verlag, 1999), pp. 311–321.
35. M. Kaufmann, P. Manolios, and J S. Moore, *Computer-Aided Reasoning: An Approach* (Kluwer Academic Publishers, 2000).
36. H. Kirchner and C. Ringeissen, editors, *Frontiers of Combining Systems: Third International Workshop, FroCoS 2000: Nancy, March 2000: Proceedings*, Lecture Notes in Artificial Intelligence, vol. 1794 (Springer-Verlag, 2000).
37. B. Krieg-Brückner, J. Peleska, E.-R. Olderog, and A. Baer, 'The UniForM Work-Bench, a universal development environment for formal methods', in *FM'99—Formal Methods*, edited by J. M. Wing, J. Woodcock, and J. Davies, vol. 2, Lecture Notes in Computer Science, vol. 1709 (Springer-Verlag, 1999), pp. 1186–1205.
38. K. L. McMillan, *Symbolic Model Checking* (Kluwer Academic Publishers, 1993).
39. K. L. McMillan, 'Verification of infinite state systems by compositional model checking', in *Correct Hardware Design and Verification Methods*, edited by L. Pierre and T. Kropf, Lecture Notes in Computer Science, vol. 1703 (Springer-Verlag, 1999), pp. 219–233.
40. K. L. McMillan, 'Parameterized Verification of the FLASH Cache Coherence Protocol by Compositional Model Checking', in *Correct Hardware Design and Verification Methods: 11th IFIP WG10.5 Advanced Research Working Conference, CHARME 2001: Livingston, Scotland, UK, September 4–7 2001: Proceed-*

ings, edited by T. Margaria and T. Melham, Lecture Notes in Computer Science, vol. 2144, (Springer-Verlag, 2001), pp. 179–195.

41. Microsoft Corporation, *Microsoft Excel*, `www.microsoft.com/excel`.
42. R. Milner, M. Tofte, R. Harper, and D. MacQueen, *The Definition of Standard ML*, revised edition (MIT Press, 1997).
43. N. Mokhoff, 'Intel, Motorola report formal verification gains', *The EE Times Online*, `www.eetimes.com/story/OEG20010621S0080`.
44. Object Management Group, *The Common Object Request Broker: Architecture and Specification*, OMG Technical Report (July 1995).
45. J. O'Leary, X. Zhao, R. Gerth, and C.-J. H. Seger, 'Formally verifying IEEE compliance of floating-point hardware', *Intel Technology Journal* (First Quarter, 1999). Available online at `developer.intel.com/technology/itj/`.
46. Prosper: Proof and Specification Assisted Design Environments, ESPRIT Framework IV LTR Project 26241, `www.dcs.gla.ac.uk/prosper/`.
47. S. Rajan, N. Shankar, and M. Srivas, 'An integration of model checking and automated proof checking', in *Proceedings of the International Conference on Computer-Aided Verification*, Lecture Notes in Computer Science, vol. 939 (Springer-Verlag, 1995), pp. 84–97.
48. K. Schneider and D. W. Hoffmann, 'A HOL Conversion for Translating Linear Time Temporal Logic to ω-Automata', in *Proceedings of the 12th International Conference on Theorem Proving in Higher Order Logics, Nice, 14–17 September, 1999*, Lecture Notes in Computer Science, vol. 1690 (Springer-Verlag, 1999), pp. 255–272.
49. C.-J. H. Seger and R. E. Bryant, 'Formal verification by symbolic evaluation of partially-ordered trajectories', *Formal Methods in System Design*, vol. 6 (1995), pp. 147–189.
50. M. Sheeran and G. Stålmarck, 'A tutorial on Stålmarck's proof procedure for propositional logic', *Formal Methods in System Design*, vol. 16, no. 1 (2000), pp. 23–58.
51. M. Staples, *Linking ACL2 and HOL*, Technical Report 476, University of Cambridge Computer Laboratory (1999).
52. B. Steffen, T. Margaria, and V. Braun, 'The Electronic Tool Integration Platform: concepts and design', *International Journal on Software Tools for Technology Transfer*, vol. 1 (1997), pp. 9–30.
53. A. Stevenson and L. A. Dennis, 'Integrating SVC and HOL with the Prosper Toolkit', in *Supplemental Proceedings of the 13th International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2000)*, OGI Technical Report CSE 00-009, Oregon Graduate Institute (August, 2000), pp. 199–206.
54. T. E. Uribe, 'Combinations of Model Checking and Theorem Proving', in *Frontiers of Combining Systems: Third International Workshop, FroCoS 2000: Nancy, March 2000: Proceedings*, edited by H. Kirchner and C. Ringeissen, Lecture Notes in Artificial Intelligence, vol. 1794 (Springer-Verlag, 2000), pp. 151–170.