Theorem Proving for Verification
(Invited Tutorial)

John Harrison

Intel Corporation, JF1-13
2111 NE 25th Avenue, Hillsboro OR 97124, USA
johnh@ichips.intel.com

1 The Scope of Automation

There are numerous verification techniques in active use. Traditional testing and simulation usually only provide a limited guarantee, since they can seldom exercise all possible situations. Methods based on abstraction consciously simplify the problem to make its complete analysis tractable, but still do not normally completely verify the ultimate target. We will confine ourselves here to full formal verification techniques that can be used to prove complete correctness of a (model of a) system with respect to a formal specification. Roughly speaking, these methods model the system and specification in a logical formalism and then apply general methods to determine whether the formal expressions are valid, indicating correctness of the model with respect to the specification. Typical formalisms include:

- Propositional logic, a.k.a. Boolean algebra
- Temporal logic (CTL, LTL etc.)
- Quantifier-free combinations of first-order theories
- Full first-order logic
- Higher-order logic or first-order logic with arithmetic or set theory

This list is organized approximately in order of increasing logical generality, with formalisms later in the list often subsuming earlier ones. But there is a price to be paid for this generality: deciding validity in the formalisms becomes successively more difficult.

Testing validity (tautology) for propositional logic is just [the dual of] the well-studied propositional satisfiability problem (SAT), and even though the problem is known to be [co-]NP-complete [15], there are many reasonable algorithms [4,17,33,59] and some of these are implemented in practical SAT solvers with very good performance on typical problems, e.g. [22,26,46]. Many temporal logics, even quite rich ones, permit efficient 'model checking' algorithms based on explicit-state reachability analysis [13,52], its symbolic refinement [7] or automata-related techniques [68]. Algorithms for testing validity of formulas without quantifiers in combinations of common first-order theories like linear arithmetic and lists, originating in [47] and further studied in many recent papers [48,40], have been implemented in a series of tools now commonly known as SMT (Satisfiability Modulo Theories) solvers [2].

A. Gupta and S. Malik (Eds.): CAV 2008, LNCS 5123, pp. 11–18, 2008.

Once we reach full first-order logic, where we allow arbitrary use of the quantifiers 'for all' and 'there exists' over domain objects, the validity problem actually becomes undecidable [12,66]. However it is at least semidecidable, and there has been an extensive line of research in developing first-order provers, going right back to [18,24] and leading to several effective search algorithms incorporating unification, most famously resolution [54]. There are many good modern implementations of first-order proof search, and they have even occasionally answered significant open problems [43,44]. Yet the scope in principle of first-order logic is vast: any deductive inference from the accepted set-theoretic axioms of mathematics can in principle be found by first-order proof search. In contrast with this, the practical scope of these methods is tiny.

As soon as we reach higher-order logic, where quantification over functions or predicates is allowed, the problem is no longer even semidecidable [25,60]; in fact this holds even if we merely ask whether *first-order* formulas about arithmetic are *true*, as distinct from being deductive consequences of some decidable axioms. A particularly sharp form of this result is the undecidability of Hilbert's 10th problem [42]: there is no algorithm to decide whether a multivariate polynomial equation has a solution over the integers.

2 Interactive Theorem Proving

Traditionally, in the 'theorem proving' approach to verification, one works in a relatively expressive formalism such as higher-order logic, and accepts the fact that automation of the validity checking process is going to be impossible, or at least practically infeasible. Instead, one approaches the task in a more deductive way just like a traditional mathematical proof, verifying the result by machine, but applying a sequence of logical reasoning principles wholly or partly under human guidance. In fact, interactive theorem provers in this sense only appeared some time after the first experiments with fully automated provers, perhaps in disillusionment over their relatively limited scope. The first interactive provers in the modern sense were probably the SAM (semi-automated mathematics) series, whose manifesto stated [29]:

> Semi-automated mathematics is an approach to theorem-proving which seeks to combine automatic logic routines with ordinary proof procedures in such a manner that the resulting procedure is both efficient and subject to human intervention in the form of control and guidance. Because it makes the mathematician an essential factor in the quest to establish theorems, this approach is a departure from the usual theorem-proving attempts in which the computer *unaided* seeks to establish proofs.

Nowadays there is active research activity in both 'interactive' and 'automated' theorem proving. Not long after the SAM project, the AUTOMATH [19,20], Mizar [64,65], NQTHM [3] and LCF [28] proof checkers appeared, and each of them in its way has been profoundly influential. Many of the most successful interactive theorem provers around today are directly descended from one (or more) of these. As well as the automated methods we have already mentioned, such as SMT and first-order proving, there

have been some successes for more specialized automated methods, e.g. in real algebra [14,61], polynomial ideals [5,39] and geometry [11,69].

It is desirable to make even 'interactive' provers as efficient as possible by incorporating powerful automated subsystems, so that the human can focus on the really difficult and creative parts of the proof. One approach is simply to combine the interactive provers with automated external systems [36,53,56]. However, most interactive theorem provers aim to prove theorems with a high degree of logical rigor, so relying on external tools is not uncontroversial. One way of combining efficiency and rigor is to use the external tool only to provide a certificate that the theorem prover can relatively easily check in a rigorous fashion [32]. Indeed, one can even have the external tool provide a complete logical *proof*, in the case of an automated theorem prover [34].

There are numerous interactive theorem provers in the world. The book [70] gives an instructive survey of some of the main interactive theorem provers: HOL, Mizar, PVS, Coq, Otter/IVY, Isabelle/Isar, Alfa/Agda, ACL2, PhoX, IMPS, Metamath, Theorema, Lego, Nuprl, Omega, B and Minlog. In each case, a proof of the irrationality of $\sqrt{2}$ is given, and some of the key features surveyed.

3 Why Theorem Proving?

Experience suggests, unsurprisingly, that highly automated techniques such as symbolic simulation [9], symbolic trajectory evaluation [57] and model checking [7] are much easier to learn, and often more productive to use, than methods like theorem proving that rely more on human interaction. They therefore tend to win acceptance much more easily, particularly in industrial practice but even in terms of academic interest. For example, the idea of using temporal logic for program verification goes at least back to [8,50]. Yet this attracted relatively little interest until the subsequent development of effective decision algorithms, particularly symbolic model checking. So with this in mind, why would one want to use theorem proving instead of model checking or other highly automated techniques? We can think of several reasons, which we rank in approximately decreasing order of perceived importance.

3.1 Beyond the Scope of Automated Methods

Some problems are simply not in the scope of any of the mainstream automated methods. Anything of interest about a finite-state transition system is decidable in principle, and thanks to model checkers, quite effectively in practice. But this trivial decidability breaks down as soon as we start to think about infinite-state systems, or even systems of an arbitrary finite size. For example, in a multiprocessor cache system with N nodes, we can usually model the essentials of cache coherence as a reachability problem in a finite state transition system for any *particular* value of N. But we might actually want a guarantee that such a parametrized system is correct for *any* (finite) value of the parameter N, and this is not a priori within the scope of automated methods. Although there is an extensive body of research on techniques for fully automated verification of parametrized systems [21,23,51], methods requiring at least some human guidance seem to be necessary in most practical applications.

For another example, consider verifying the correctness of floating-point arithmetic circuits. The desired specification in the IEEE Standard governing binary floating-point arithmetic [35] is in terms of real numbers, not bitstrings. Roughly speaking, one needs to prove that, for example, the result of a floating-point square root operation $\sqrt{x}$ is the closest floating-point number to the exact mathematical answer, which is in general an irrational number. It is not at all clear how to express this in limited formalisms where reasoning about arbitrary real numbers is impossible. One can come up with reasonably natural specifications for simple integer adders and multipliers in Boolean terms, but this becomes progressively more difficult when one considers division and square root, and seems quite impractical for transcendental functions. Thus, it is not surprising that one of the most popular and successful application areas of theorem proving to industrial verification is in the domain of floating-point arithmetic [30,37,38,45,49,55,58].

3.2 Verification of Underlying Theory

Even if some key properties of the system can be proved automatically, a global verification often demands deeper analysis of the underlying environment and background assumptions. Many program verification techniques simply rely on extracting verification conditions syntactically from an annotated program. Yet the connection between those verification conditions and the correct running of the program is often not formalized or verified. However, by starting from a formal semantics of the programming language in a general mathematical theorem prover, such properties can be rigorously proved in a unified way [27]. Also, some programs depend essentially on non-trivial mathematics. If one merely verifies the programming aspects, taking for granted the underlying theory, one risks either making a mistake in the theory itself or mis-applying it. For example, when verifying a program or circuit to perform elliptic curve cryptography, one could prove a specification of satisfying finality by formalizing also the underlying mathematics [63].

3.3 More Efficient

Even when something is within the scope of automated methods in principle, and despite the remarkable efficiency of many modern automated tools, larger systems can be difficult to verify in a practical amount of time. For example, going back to the example of a parametrized system, one may find that some such system is automatically verifiable for each specific N, but that the state space, and hence the runtime, increases so dramatically as N increases that one can't get past $N = 2$ or $N = 3$. In general, techniques like model checking that rely on state exploration tend to degrade in performance as the size of the system and/or specification increases. By contrast, the more deductive style of proof that theorem provers encourage often proceeds in a more structured way, e.g. using induction, and is largely independent of the size. Indeed, to describe techniques like model-checking as 'automatic' is in some ways a bit misleading. Very often, one needs to make significant modifications to or decompositions of the problem, or tweak parameters of the checker such as BDD variable ordering, in order to bring it within practical reach. Sometimes this gets so tedious and unproductive that one would be better off just settling on a deductive proof in the first place.

3.4 More Intellectually Stimulating

Theorem proving encourages a style of verification where the human uses a conceptual understanding of the system to construct a mathematical proof, rather than a 'push-button' approach of waiting for a yes/no answer from a black box. Although this may be, in a crude sense, much less productive, it can be valuable because it forces the human to articulate dimly perceived intuitions about the system, and so perhaps gain a significantly deeper conceptual understanding that may even help to improve the system. An example is reported in [31], which describes the formal verification of division algorithms. On formalizing one of the standard theorems [41], the author noticed that the full strength of one of the assumptions was never used, and a sharper theorem actually allowed the implementation of faster algorithms with the same behavior. Of course, it is not inconceivable that a sufficiently close reading would have led to the same revelation, but it is less likely without the level of logical rigor that a theorem prover imposes.

References

1. Aagaard, M., Harrison, J. (eds.): TPHOLs 2000. LNCS, vol. 1869. Springer, Heidelberg (2000)
2. Barrett, C., Sebastiani, R., Seshia, S., Tinelli, C.: Satisfiability modulo theories. In: Biere, A., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, vol. 4. IOS Press, Amsterdam (2008)
3. Boyer, R.S., Moore, J.S.: A Computational Logic. ACM Monograph Series. Academic Press, London (1979)
4. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers C-35, 677–691 (1986)
5. Buchberger, B.: Ein algorithmisches Kriterium fur die Lösbarkeit eines algebraischen Gleichungssystems. Aequationes Mathematicae 4, 374–383 (1970); English translation, An Algorithmical Criterion for the Solvability of Algebraic Systems of Equations. In: [6], pp. 535–545
6. Buchberger, B., Winkler, F. (eds.): Gröbner Bases and Applications. London Mathematical Society Lecture Note Series, vol. 251. Cambridge University Press, Cambridge (1998)
7. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. Information and Computation 98, 142–170 (1992)
8. Burstall, R.M.: Program proving as hand simulation with a little induction. In: Information Processing 1974: Proceedings of IFIP Congress 1974, Stockholm, pp. 308–312. North-Holland, Amsterdam (1974)
9. Carter, W.C., Joyner, W.H., Brand, D.: Symbolic simulation for correct machine design. In: Proceedings of the 16th ACM/IEEE Design Automation Conference, pp. 280–286. IEEE Computer Society Press, Los Alamitos (1979)
10. Caviness, B.F., Johnson, J.R. (eds.): Quantifier Elimination and Cylindrical Algebraic Decomposition. Texts and monographs in symbolic computation. Springer, Heidelberg (1998)
11. Chou, S.-C.: An introduction to Wu's method for mechanical theorem proving in geometry. Journal of Automated Reasoning 4, 237–267 (1988)
12. Church, A.: An unsolvable problem of elementary number-theory. American Journal of Mathematics 58, 345–363 (1936)
13. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Kozen, D. (ed.) Logic of Programs 1981. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)

14. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Brakhage, H. (ed.) GI-Fachtagung 1975. LNCS, vol. 33, pp. 134–183. Springer, Heidelberg (1975)
15. Cook, S.A.: The complexity of theorem-proving procedures. In: Proceedings of the 3rd ACM Symposium on the Theory of Computing, pp. 151–158 (1971)
16. Davis, M. (ed.): The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions. Raven Press, NY (1965)
17. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem proving. Communications of the ACM 5, 394–397 (1962)
18. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM 7, 201–215 (1960)
19. de Bruijn, N.G.: The mathematical language AUTOMATH, its usage and some of its extensions. In: Laudet, M., Lacombe, D., Nolin, L., Schützenberger, M. (eds.) Symposium on Automatic Demonstration. Lecture Notes in Mathematics, vol. 125, pp. 29–61. Springer, Heidelberg (1970)
20. de Bruijn, N.G.: A survey of the project AUTOMATH. In: Seldin, J.P., Hindley, J.R. (eds.) To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism, pp. 589–606. Academic Press, London (1980)
21. Delzanno, G.: Automatic verification of parameterized cache coherence protocols. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 53–68. Springer, Heidelberg (2000)
22. Een, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
23. Fontaine, P.: Techniques for verification of concurrent systems with invariants. PhD thesis, Institut Montefiore, Université de Liège (2004)
24. Gilmore, P.C.: A proof method for quantification theory: Its justification and realization. IBM Journal of research and development 4, 28–35 (1960)
25. Gödel, K.: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I. Monatshefte für Mathematik und Physik 38, 173–198 (1931); English translation, On Formally Undecidable Propositions of Principia Mathematica and Related Systems, I. In: [67], pp. 592–618 or [16], pp. 4–38
26. Goldberg, E., Novikov, Y.: BerkMin: a fast and robust Sat-solver. In: Kloos, C.D., Franca, J.D. (eds.) Design, Automation and Test in Europe Conference and Exhibition (DATE 2002), Paris, France, pp. 142–149. IEEE Computer Society Press, Los Alamitos (2002)
27. Gordon, M.J.C.: Mechanizing programming logics in higher order logic. In: Birtwistle, G., Subrahmanyam, P.A. (eds.) Current Trends in Hardware Verification and Automated Theorem Proving, pp. 387–439. Springer, Heidelberg (1989)
28. Gordon, M.J.C., Milner, R., Wadsworth, C.P.: Edinburgh LCF: A Mechanised Logic of Computation. LNCS, vol. 78. Springer, Heidelberg (1979)
29. Guard, J.R., Oglesby, F.C., Bennett, J.H., Settle, L.G.: Semi-automated mathematics. Journal of the ACM 16, 49–62 (1969)
30. Harrison, J.: Formal verification of floating point trigonometric functions. In: Johnson, S.D., Hunt Jr., W.A. (eds.) FMCAD 2000. LNCS, vol. 1954, pp. 217–233. Springer, Heidelberg (2000)
31. Harrison, J.: Formal verification of IA-64 division algorithms. In: Aagaard and Harrison [1], pp. 234–251
32. Harrison, J., Théry, L.: A sceptic's approach to combining HOL and Maple. Journal of Automated Reasoning 21, 279–294 (1998)
33. Hooker, J.N.: A quantitative approach to logical inference. Decision Support Systems 4, 45–69 (1988)

34. Hurd, J.: Integrating Gandalf and HOL. In: Bertot, Y., Dowek, G., Hirschowitz, A., Paulin, C., Théry, L. (eds.) TPHOLs 1999. LNCS, vol. 1690, pp. 311–321. Springer, Heidelberg (1999)
35. IEEE. Standard for binary floating point arithmetic. ANSI/IEEE Standard 754-1985, The Institute of Electrical and Electronic Engineers, Inc., 345 East 47th Street, New York, NY 10017, USA (1985)
36. Joyce, J.J., Seger, C.: The HOL-Voss system: Model-checking inside a general-purpose theorem-prover. In: Joyce, J.J., Seger, C. (eds.) HUG 1993. LNCS, vol. 780, pp. 185–198. Springer, Heidelberg (1994)
37. Kaivola, R., Aagaard, M.D.: Divider circuit verification with model checking and theorem proving. In: Aagaard and Harrison [1], pp. 338–355
38. Kaivola, R., Kohatsu, K.: Proof engineering in the large: Formal verification of the Pentium (R) 4 floating-point divider. In: Margaria, T., Melham, T.F. (eds.) CHARME 2001. LNCS, vol. 2144, pp. 196–211. Springer, Heidelberg (2001)
39. Kandri-Rody, A., Kapur, D.: Algorithms for computing Gröbner bases of polynomial ideals over various Euclidean rings. In: Fitch, J. (ed.) EUROSAM 1984 and ISSAC 1984. LNCS, vol. 174, pp. 195–206. Springer, Heidelberg (1984)
40. Krstic, S., Goel, A.: Architecting solvers for SAT modulo theories: Nelson-Oppen with DPLL. In: Konev, B., Wolter, F. (eds.) FroCos 2007. LNCS (LNAI), vol. 4720, pp. 1–27. Springer, Heidelberg (2007)
41. Markstein, P.W.: Computation of elementary functions on the IBM RISC System/6000 processor. IBM Journal of Research and Development 34, 111–119 (1990)
42. Matiyasevich, Y.V.: Enumerable sets are Diophantine. Soviet Mathematics Doklady 11, 354–358 (1970)
43. McCune, W.: Solution of the Robbins problem. Journal of Automated Reasoning 19, 263–276 (1997)
44. McCune, W., Padmanabhan, R.: Automated Deduction in Equational Logic and Cubic Curves. LNCS, vol. 1095. Springer, Heidelberg (1996)
45. Moore, J.S., Lynch, T., Kaufmann, M.: A mechanically checked proof of the correctness of the kernel of the $AMD5_K86$ floating-point division program. IEEE Transactions on Computers 47, 913–926 (1998)
46. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th Design Automation Conference (DAC 2001), pp. 530–535. ACM Press, New York (2001)
47. Nelson, G., Oppen, D.C.: Simplification by cooperating decision procedures. ACM Transactions on Programming Languages and Systems 1, 245–257 (1979)
48. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: from an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). Journal of the ACM 53, 937–977 (2006)
49. O'Leary, J., Zhao, X., Gerth, R., Seger, C.-J.H.: Formally verifying IEEE compliance of floating-point hardware. Intel Technology Journal 1999-Q1, 1–14 (1999), `http://developer.intel.com/technology/itj/q11999/articles/art_5.htm`
50. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, pp. 46–67 (1977)
51. Pnueli, A., Ruah, S., Zuck, L.: Automatic Deductive Verification with Invisible Invariants. In: Margaria, T., Yi, W. (eds.) ETAPS 2001 and TACAS 2001. LNCS, vol. 2031. Springer, Heidelberg (2001)
52. Queille, J.P., Sifakis, J.: Specification and verification of concurrent programs in CESAR. In: Dezani-Ciancaglini, M., Montanari, U. (eds.) Programming 1982. LNCS, vol. 137, pp. 195–220. Springer, Heidelberg (1982)

53. Rajan, S., Shankar, N., Srivas, M.K.: An integration of model-checking with automated proof-checking. In: Wolper, P. (ed.) CAV 1995. LNCS, vol. 939, pp. 84–97. Springer, Heidelberg (1995)
54. Robinson, J.A.: A machine-oriented logic based on the resolution principle. Journal of the ACM 12, 23–41 (1965)
55. Rusinoff, D.: A mechanically checked proof of IEEE compliance of a register-transfer-level specification of the AMD-K7 floating-point multiplication, division, and square root instructions. LMS Journal of Computation and Mathematics 1, 148–200 (1998), http://www.onr.com/user/russ/david/k7-div-sqrt.html
56. Seger, C., Joyce, J.J. : A two-level formal verification methodology using HOL and COSMOS. Technical Report 91-10, Department of Computer Science, University of British Columbia, 2366 Main Mall, University of British Columbia, Vancouver, B.C, Canada V6T 1Z4 (1991)
57. Seger, C.-J.H., Bryant, R.E.: Formal verification by symbolic evaluation of partially-ordered trajectories. Formal Methods in System Design 6, 147–189 (1995)
58. Slobodová, A.: Challenges for Formal Verification in Industrial Setting. In: Brim, L., Haverkort, B.R., Leucker, M., van de Pol, J. (eds.) FMICS 2006 and PDMC 2006. LNCS, vol. 4346, pp. 1–22. Springer, Heidelberg (2007)
59. Stålmarck, G., Säflund, M.: Modeling and verifying systems and software in propositional logic. In: Daniels, B.K. (ed.) Safety of Computer Control Systems (SAFECOMP 1990), Gatwick, UK, pp. 31–36. Pergamon Press, Oxford (1990)
60. Tarski, A.: Der Wahrheitsbegriff in den formalisierten Sprachen. Studia Philosophica 1, 261–405 (1936); English translation, The Concept of Truth in Formalized Languages. In: [62], pp. 152–278
61. Tarski, A.: A Decision Method for Elementary Algebra and Geometry. University of California Press (1951); Previous version published as a technical report by the RAND Corporation (1948); prepared for publication by McKinsey, J.C.C. Reprinted In: [10], pp. 24–84
62. Tarski, A. (ed.): Logic, Semantics and Metamathematics. Clarendon Press (1956)
63. Théry, L., Hanrot, G.: Primality proving with elliptic curves. In: Schneider, K., Brandt, J. (eds.) TPHOLs 2007. LNCS, vol. 4732, pp. 319–333. Springer, Heidelberg (2007)
64. Trybulec, A.: The Mizar-QC/6000 logic information language. ALLC Bulletin (Association for Literary and Linguistic Computing) 6, 136–140 (1978)
65. Trybulec, A., Blair, H.A.: Computer aided reasoning. In: Parikh, R. (ed.) Logics of Programs, Brooklyn. LNCS, vol. 193, pp. 406–412. Springer, Heidelberg (1985)
66. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proceedings of the London Mathematical Society 42(2), 230–265 (1936)
67. van Heijenoort, J. (ed.): From Frege to Gödel: A Source Book in Mathematical Logic 1879–1931. Harvard University Press (1967)
68. Vardi, M.: An automata-theoretic approach to linear temporal logic. In: Moller, F., Birtwistle, G. (eds.) Logics for Concurrency. LNCS, vol. 1043, pp. 238–266. Springer, Heidelberg (1996)
69. Wen-tsün, W.: On the decision problem and the mechanization of theorem proving in elementary geometry. Scientia Sinica 21, 157–179 (1978)
70. Wiedijk, F.: The Seventeen Provers of the World. LNCS (LNAI), vol. 3600. Springer, Heidelberg (2006)