Automated Verification: Graphs, Logic, and Automata

Moshe Y. Vardi*
Rice University
Houston, TX, USA
vardi@cs.rice.edu
http://www.cs.rice/edu/~vardi

Abstract

Automated verification is one of the most successful applications of automated reason in computer science. In automated verification one uses algorithmic techniques to establish the correctness of the design with respect to a given property. Automated verification is based on a small number of key algorithmic ideas, tying together graph theory, automata theory, and logic. In this self-contained talk I will describe how this "holy trinity" gave rise to automated-verification tools, and mention some applications to planning.

1 Introduction

The recent growth in computer power and connectivity has changed the face of science and engineering, and is changing the way business is being conducted. This revolution is driven by the unrelenting advances in semiconductor manufacturing technology. Nevertheless, the U.S. semiconductor community faces a serious challenge: chip designers are finding it increasingly difficult to keep up with the advances in semiconductor manufacturing. As a result, they are unable to exploit the enormous capacity that this technology provides. The International Technology Roadmap for Semiconductors suggests that the semiconductor industry will require productivity gains greater than the historical 20% per-year to keep up with the increasing complexity of semiconductor designs. This is referred to as the "design productivity crisis". As designs grow more complex, it becomes easier to introduce flaws into the design. Thus, designers use various validation techniques to verify the correctness of the design. Unfortunately, these techniques themselves grow more expensive and difficult with design complexity. As the validation process has begun to consume more than half the project design resources, the semiconductor industry has begun to refer to this problem as the "validation crisis".

Formal verification is a process in which mathematical techniques are used to guarantee the correctness of a design with respect to some specified behavior. Automated formal-verification tools, based on *model-checking technology* [Clarke *et al.*, 1986; Lichtenstein and Pnueli, 1985; Queille and Sifakis, 1981; Vardi and Wolper, 1986] have enjoyed a substantial and growing use over the last few years, showing an ability to discover subtle flaws that result from extremely improbable events [Clarke *et al.*, 1999]. While until recently these tools were viewed as of academic interest only, they are now routinely used in industrial applications, resulting in decreased time to market and increased product integrity [Kurshan, 1997]. It is fair to say that automated verification is one of the most successful applications of automated reason in computer science [Clarke and Wing, 1996].

2 Basic Theory

The first step in formal verification is to come up with a *formal specification* of the design, consisting of a description of the desired behavior. One of the more widely used specification languages for designs is *temporal logic* [Pnueli, 1977]. In *linear* temporal logics, time is treated as if each moment in time has a unique possible future. Thus, linear temporal formulas are interpreted over linear sequences, and we regard them as describing the behavior of a single computation of a system. (For a discussion of linear time vs. branching time, see [Vardi, 2001].)

In the linear temporal logic LTL, formulas are constructed from a set $Prop$ of atomic propositions using the usual Boolean connectives as well as the unary temporal connective X ("next"), F ("eventually"), G ("always"), and the binary temporal connective U ("until"). For example, the LTL formula $G(\mathit{request} \rightarrow F\ \mathit{grant})$, which refers to the atomic propositions *request* and *grant*, is true in a computation precisely when every state in the computation in which *request* holds is followed by some state in the future in which *grant* holds. The LTL formula $G(\mathit{request} \rightarrow (\mathit{request}\ U\ \mathit{grant}))$ is true in a computation precisely if, whenever *request* holds in a state of the computation, it holds until a state in which *grant* holds is reached.

LTL is interpreted over *computations*, which can be viewed as infinite sequences of truth assignments to the atomic propositions; i.e., a computation is a function $\pi : I\!N \rightarrow 2^{Prop}$ that assigns truth values to the elements of $Prop$ at each time instant (natural number). For a computation π and a point $i \in I\!N$, the notation $\pi, i \models \varphi$ indicates that a formula φ holds at the point i of the computation π. For example,

*Supported in part by NSF grants CCR-9988322, CCR-0124077, IIS-9908435, IIS-9978135, and EIA-0086264, by BSF grant 9800096, and by a grant from the Intel Corporation.

$\pi, i \models X\varphi$ iff $\pi, i+1 \models \varphi$, and and $\pi, i \models \varphi U \psi$ iff for some $j \geq i$, we have $\pi, j \models \psi$ and for all k, $i \leq k < j$, we have $\pi, k \models \varphi$. We say that π *satisfies* a formula φ, denoted $\pi \models \varphi$, iff $\pi, 0 \models \varphi$. The connectives F and G can be defined in terms of the connective U: $F\varphi$ is defined as $\mathbf{true}\ U\varphi$, and $G\varphi$ is defined as $\neg F \neg \varphi$.

Designs can be described in a variety of formal description formalisms. Regardless of the formalism used, a *finite-state design* can be abstractly viewed as a *labeled transition system*, i.e., as a structure of the form $M = (W, W_0, R, V)$, where W is the finite set of states that the system can be in, $W_0 \subseteq W$ is the set of initial states of the system, $R \subseteq W^2$ is a transition relation that indicates the allowable state transitions of the system, and $V : W \to 2^{Prop}$ assigns truth values to the atomic propositions in each state of the system. (A labeled transition system is essentially a Kripke structure.) A *path* in M that *starts at* u is a possible infinite behavior of the system starting at u, i.e., it is an infinite sequence $u_0, u_1 \ldots$ of states in W such that $u_0 = u$, and $u_i \; R \; u_{i+1}$ for all $i \geq 0$.The sequence $V(u_0), V(u_1) \ldots$ is a *computation* of M that *starts at* u. It is the sequence of truth assignments visited by the path, The *language* of M, denoted $L(M)$ consists of all computations of M that start at a state in W_0. Note that $L(M)$ can be viewed as a language of infinite words over the alphabet 2^{Prop}. $L(M)$ can be viewed as an abstract description of a system, describing all possible "traces". We say that M *satisfies* an LTL formula φ if all computations in $L(M)$ satisfy φ, that is, if $L(M) \subseteq \text{models}(\varphi)$. When M satisfies φ we also say that M is a model of φ, which explain why the technique is known as *model checking* [Clarke *et al.*, 1999].

One of the major approaches to automated verification is the *automata-theoretic approach*, which underlies many model checkers, e.g., SPIN [Holzmann, 1997]. The key idea underlying the automata-theoretic approach is that, given an LTL formula φ, it is possible to construct a finite-state automaton A_φ on infinite words that accepts precisely all computations that satisfy φ [Vardi and Wolper, 1994]. The type of finite automata on infinite words we consider is the one defined by Büchi 1962. A *Büchi automaton* is a tuple $A = (\Sigma, S, S_0, \rho, F)$, where Σ is a finite alphabet, S is a finite set of states, $S_0 \subseteq S$ is a set of initial states, $\rho : S \times \Sigma \to 2^S$ is a nondeterministic transition function, and $F \subseteq S$ is a set of accepting states. A *run* of A over an infinite word $w = a_1 a_2 \cdots$, is a sequence $s_0 s_1 \cdots$, where $s_0 \in S_0$ and $s_i \in \rho(s_{i-1}, a_i)$ for all $i \geq 1$. A run $s_0, s_1, \ldots$ is *accepting* if there is some accepting state that repeats infinitely often, i.e., for some $s \in F$ there are infinitely many i's such that $s_i = s$. The infinite word w is *accepted* by A if there is an accepting run of A over w. The *language* of infinite words accepted by A is denoted $L(A)$. The following fact establishes the correspondence between LTL and Büchi automata [Vardi and Wolper, 1994] (for a tutorial introduction for this correspondence, see [Vardi, 1996]):

Theorem 1 *Given an LTL formula φ, one can build a Büchi automaton $A_\varphi = (\Sigma, S, S_0, \rho, , F)$, where $\Sigma = 2^{Prop}$ and $|S| \leq 2^{O(|\varphi|)}$, such that $L(A_\varphi)$ is exactly the set of computations satisfying the formula φ.*

This correspondence reduces the verification problem to an automata-theoretic problem as follows [Vardi and Wolper, 1986]. Suppose that we are given a system M and an LTL formula φ. We check whether $L(M) \subseteq \text{models}(\varphi)$ as follows: (1) construct the automaton $A_{\neg\varphi}$ that corresponds to the *negation* of the formula φ, (2) take the *cross product* of the system M and the automaton $A_{\neg\varphi}$ to obtain an automaton $A_{M,\varphi}$, such that $L(A_{M,\varphi}) = L(M) \cap L(A_{\neg\varphi})$, and (3) check whether the language $L(A_{M,\varphi})$ is nonempty, i.e., whether $A_{M,\varphi}$ accepts *some* input.

Theorem 2 *Let M be a labeled transition system and φ be an LTL formula. Then M satisfies φ iff $L(A_{M,\varphi}) = \emptyset$.*

If $L(A_{M,\varphi})$ is empty, then the design is correct. Otherwise, the design is incorrect and the word accepted by $L(A_{M,\varphi})$ is an incorrect computation. Model-checking tools use algorithms for checking emptiness of Büchi automata [Emerson and Lei, 1985; Courcoubetis *et al.*, 1992] to check emptiness of $L(A_{M,\varphi})$. In case of nonemptiness, the incorrect computation is presented to the user as a finite trace, possibly followed by a cycle. Thus, once the automaton $A_{\neg\varphi}$ is constructed, the verification task is reduced to automata-theoretic problems, namely, intersecting automata and testing emptiness of automata, which have highly efficient solutions [Vardi, 1996]. Furthermore, using data structures that enable compact representation of very large state space makes it possible to verify designs of significant complexity [Burch *et al.*, 1992].

The linear-time framework is not limited to using LTL as a specification language. ForSpec is a recent extension of LTL, designed to address the need of the semiconductor industry [Armoni *et al.*, 2002]. There are those who prefer to use automata on infinite words as a specification formalism [Vardi and Wolper, 1994]; in fact, this is the approach of COSPAN [Kurshan, 1994]. In this approach, we are given a design represented as a finite transition system M and a property represented by a Büchi (or a related variant) automaton P. The design is correct if all computations in $L(M)$ are accepted by P, i.e., $L(M) \subseteq L(P)$. This approach is called the *language-containment* approach. To verify M with respect to P, we: (1) construct the automaton P^c that *complements* P, (2) take the product of the system M and the automaton P^c to obtain an automaton $A_{M,P}$, and (3) check that the automaton $A_{M,P}$ is nonempty. As before, the design is correct iff $A_{M,P}$ is empty. Thus, the verification task is again reduced to automata-theoretic problems, namely intersecting and complementing automata and testing emptiness of automata.

3 Concluding Remarks

Over the last few years, automated formal verification tools, such as model checkers, have shown their ability to provide a thorough analysis of reasonably complex designs [Goering, 1997]. Companies such as AT&T, Cadence, Fujitsu, HP, IBM, Intel, Motorola, NEC, SGI, Siemens, and Sun are using model checkers increasingly on their own designs to reduce time to market and ensure product quality, cf. [Beer *et al.*, 1994].

There has recently also been an fruitful interaction between model checking and planning in AI. On one hand, the usage of satisfiability technology to solve planning problems [Kautz and Selman, 1992] has been adopted into model checking

[Biere *et al.*, 1999; Copty *et al.*, 2001]. At the same time, symbolic techniques from model checking have been adopted into AI planning [Cimatti and Roveri, 2000]. Finally, temporal logic and the automata-theoretic perspective has shown to be useful for planning with temporally extended goals [Bacchus and Kabanza, 1998; 2000; Giacomo and Vardi, 1999; Calvanese *et al.*, 2002].

References

[Armoni *et al.*, 2002] R. Armoni, L. Fix, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, A. Tiemeyer, E. Singerman, M.Y. Vardi, and Y. Zbar. The ForSpec temporal language: A new temporal property-specification language. In *Proc. 8th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, Lecture Notes in Computer Science 2280, pages 296–311. Springer-Verlag, 2002.

[Bacchus and Kabanza, 1998] Fahiem Bacchus and Froduald Kabanza. Planning for temporally extended goals. *Ann. of Mathematics and Artificial Intelligence*, 22:5–27, 1998.

[Bacchus and Kabanza, 2000] Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *AI J.*, 116(1–2):123–191, 2000.

[Beer *et al.*, 1994] I. Beer, S. Ben-David, D. Geist, R. Gewirtzman, and M. Yoeli. Methodology and system for practical formal verification of reactive hardware. In *Proc. 6th Conference on Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 182–193, Stanford, June 1994.

[Biere *et al.*, 1999] A. Biere, A. Cimatti, E.M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Proc. 36th Design Automation Conference*, pages 317–320. IEEE Computer Society, 1999.

[Büchi, 1962] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. Logic, Method. and Philos. Sci. 1960*, pages 1–12, Stanford, 1962. Stanford University Press.

[Burch *et al.*, 1992] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[Calvanese *et al.*, 2002] D. Calvanese, G. De Giacomo, and M.Y. Vardi. Reasoning about actions and planning in ltl action theories. In *Proc. 8th Int'l Conf. on the Principles of Knowledge Representation and Reasoning*, pages 593–602. Morgan Kaufmann, 2002.

[Cimatti and Roveri, 2000] A. Cimatti and M. Roveri. Conformant planning via symbolic model checking. *J. of AI Research*, 13:305–338, 2000.

[Clarke and Wing, 1996] E.M. Clarke and J.M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28:626–643, 1996.

[Clarke *et al.*, 1986] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.

[Clarke *et al.*, 1999] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[Copty *et al.*, 2001] F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M.Y. Vardi. Benefits of bounded model checking at an industrial setting. In *Computer Aided Verification, Proc. 13th International Conference*, volume 2102 of *Lecture Notes in Computer Science*, pages 436–453. Springer-Verlag, 2001.

[Courcoubetis *et al.*, 1992] C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1:275–288, 1992.

[Emerson and Lei, 1985] E.A. Emerson and C.-L. Lei. Temporal model checking under generalized fairness constraints. In *Proc. 18th Hawaii International Conference on System Sciences*, North Holywood, 1985. Western Periodicals Company.

[Giacomo and Vardi, 1999] G. De Giacomo and M.Y. Vardi. Automata-theoretic approach to planning for temporally extended goals. In *Proc. European Conf. on Planning*, volume 1809 of *Lecture Notes in AI*, pages 226–238. Springer-Verlag, 1999.

[Goering, 1997] R. Goering. Model checking expands verification's scope. *Electronic Engineering Today*, February 1997.

[Holzmann, 1997] G.J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, 23(5):279–295, May 1997. Special issue on Formal Methods in Software Practice.

[Kautz and Selman, 1992] H. Kautz and B. Selman. Planning as satisfiability. In *Proc. European Conference on Artificial Intelligence*, pages 359–379, 1992.

[Kurshan, 1994] R.P. Kurshan. *Computer Aided Verification of Coordinating Processes*. Princeton Univ. Press, 1994.

[Kurshan, 1997] R.P. Kurshan. Formal verification in a commercial setting. In *Proc. Conf. on Design Automation (DAC'97)*, volume 34, pages 258–262, 1997.

[Lichtenstein and Pnueli, 1985] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symp. on Principles of Programming Languages*, pages 97–107, New Orleans, January 1985.

[Pnueli, 1977] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. on Foundation of Computer Science*, pages 46–57, 1977.

[Queille and Sifakis, 1981] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th International Symp. on Programming*, volume

137 of *Lecture Notes in Computer Science*, pages 337–351. Springer-Verlag, 1981.

[Vardi and Wolper, 1986] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st Symp. on Logic in Computer Science*, pages 332–344, Cambridge, June 1986.

[Vardi and Wolper, 1994] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.

[Vardi, 1996] M.Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, Berlin, 1996.

[Vardi, 2001] M.Y. Vardi. Branching vs. linear time: Final showdown. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2031 of *Lecture Notes in Computer Science*, pages 1–22. Springer-Verlag, 2001.