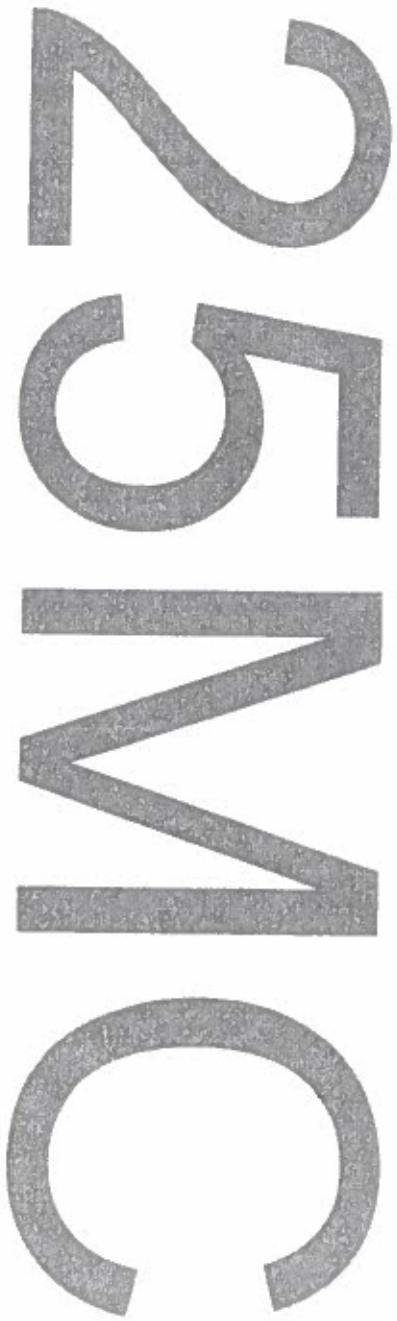
Moshe Vardi

Symposium
25 Years of Model Checking

1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019

Federated Logic Conference
Seattle 2006

Affiliated with CAV 2006



Organizers
Orna Grumberg, Technion
Helmut Veith, TU Munich

Symposium
25 Years of Model Checking

Model checking technology arguably ranges among the foremost applications of logic to computer science and computer engineering. In the twenty-five years since its invention, model checking has achieved multiple breakthroughs, bridging the gap between theoretical computer science, hardware and software engineering. Today, model checking is extensively used in the hardware industry, and has become feasible for verifying many types of software as well. Model checking has been introduced into computer science curricula at universities worldwide, and virtually has become a universal tool for the analysis of systems.

As the number of research groups and conferences in model checking is steadily increasing, the 25MC symposium focuses on the state of the art and the future challenges in model checking, seen through the eyes of the researchers which have shaped the field during the last decades.

The symposium consists of invited lectures delivered by leading researchers in the field of model checking. The invited speakers are encouraged to reflect the state of the art in their respective field of expertise, and to focus on the most important and exciting future research directions.

We are pleased to announce that the ACM is sponsoring a lunch and panel for all 25MC participants. The panel is on *Verification in the Next 25 Years* and will be comprised of the winners of the 1998 and 2006 ACM Kanellakis awards, Randy Bryant, Ed Clarke, Allen Emerson, Ken McMillan, as well as Gerard Holzmann, Robert Kurshan, Moshe Vardi and Pierre Wolper. The panel is sponsored by ACM Distinguished Lectureship Program - a program that encourages technical education and dissemination of technical information.

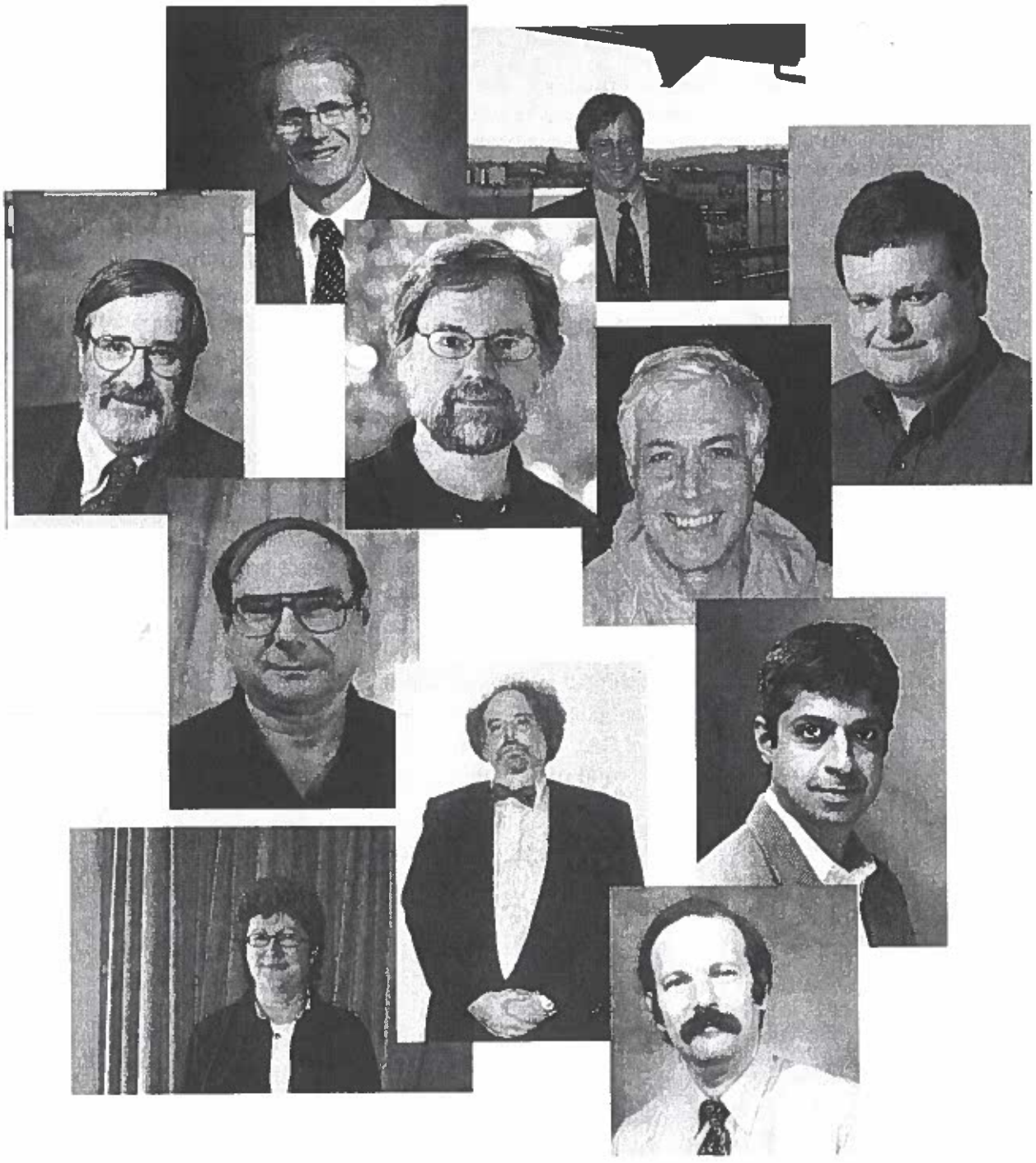
We are grateful to many people who helped making the symposium a success, in particular to Tom Ball for the support of CAV and FLOC, Mohammad Khaleghi for web design, ACM DLP for sponsoring the lunch and panel, and Limor Fix for organizing the panel.

Orna Grumberg
Helmut Veith

Association for Computing Machinery
Advancing Computing as a Science & Profession

Symposium
25 Years of Model Checking

1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019



Symposium
25 Years of Model Checking

1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019

Schedule

09.00-9.30 **Edmund Clarke**
The Birth of Model Checking
09.30-10.00 **E. Allen Emerson**
The Beginnings of Model Checking: A Personal Perspective
10.00-10.30 **Bob Kurshan**
25 Years of Computer-Aided Verification

11.00-11.30 **Gerard Holzmann**
New Challenges in Model Checking
11.30-12.00 **David Dill**
A Retrospective on Murphi
12.00-12.30 **Rajeev Alur**
Model Checking: From Tools to Theory

12.30-14.00 **25MC Lunch & Panel**
Verification in the Next 25 Years

14.00-14.30 **Thomas Henzinger**
From Graph Models to Game Models
14.30-15.00 **Limor Fix**
Accelerating the Industrial Deployment of Model-Checking for SW Verification: Lessons from 10 years of Model-Checking Deployment for HW Verification in Intel
15.00-15.30 **Randy Bryant**
A View from the Engine Room: Computational Support for Symbolic Model Checking

16.00-16.30 **Ken McMillan**
The Evolution of Symbolic Model Checking
16.30-17.00 **Moshe Y. Vardi**
From Church and Prior to PSL
17.00-17.30 **Amir Pnueli**
The Merits of Temporal Testers: Transducers Compose while Acceptors Do Not

Model Checking: From Tools to Theory

Rajeev Alur
University of Pennsylvania

Model checking is often cited as a success story for transitioning and engineering ideas rooted in logics and automata to practice. In this talk, we will discuss how the efforts aimed at improving the scope and effectiveness of model checking tools have revived the study of logics and automata leading to unexpected theoretical advances whose impact is not limited to model checking.

In particular, we will explain how our efforts to add context-free specifications to software model checking led us to the model of nested words as a representation of data with both a linear ordering and a hierarchically nested matching of items. Such dual structure occurs in diverse corners of computer science ranging from executions of structured programs where there is a natural well-nested matching of entries to and exits from functions and procedures, to XML documents with the well-nested structure given by start-tags matched with end-tags. Finite-state acceptors for nested words define the class of regular languages of nested words that has all the appealing theoretical properties that the class of classical regular word languages enjoys. We will review the emerging theory of nested words and its potential applications.

Biographical Information

Rajeev Alur is Zisman Family Professor and Graduate Group Chair in the Department of Computer and Information Science at University of Pennsylvania. He obtained his bachelor's degree in computer science from Indian Institute of Technology at Kanpur in 1987, and PhD in computer science from Stanford University in 1991. Before joining Penn in 1997, he was with Computing Science Research Center in Bell Laboratories. His areas of research include formal modeling and analysis of reactive systems, hybrid systems, model

checking, software verification, logics and automata, and design automation for embedded software. His awards include President of India's Gold Medal for academic excellence (1987), US National Science Foundation's CAREER (1997) and ITR (2001) awards, Alfred P. Sloan Faculty Fellowship (1999), and designation as a highly cited scientist by the Institute for Scientific Information (2005). Prof. Alur has (co)chaired scientific meetings such as CAV (Intl Conf on Computer-Aided Verification), EMSOFT (ACM Symp on Embedded Software), HSCC (Intl Conf on Hybrid Systems: Computation and Control), and LICS (IEEE Symp on Logic in Computer Science). He currently serves as the chair of ACM SIGBED (Special Interest Group on Embedded Systems).

A View from the Engine Room: Computational Support for Symbolic Model Checking

Randy Bryant
Carnegie Mellon University

Symbolic model checking arose from the marriage of a Boolean-level representation of the model-checking task with a computational engine that supported the required set of operations. Ordered Binary Decision Diagrams (OBDDs) were the first approach with the necessary combination of efficiency and expressive power. More recently, Boolean satisfiability (SAT) checkers based on the Davis-Putnam-Logeman-Loveland (DPLL) algorithm have greatly extended the reach of bounded model checkers.

OBDDs and DPLL-based checkers have very complementary strengths. OBDDs readily handle quantified Boolean formulas (QBF), whereas attempts to extend DPLL to QBF have had limited success. Some SAT problems that are completely intractable for DPLL are quite trivial with OBDDs. On the other hand, OBDDs cannot cope with problems that involve large numbers of variables, whereas SAT checkers are relatively insensitive to the number of variables. Despite many attempts, it has proved difficult to combine the bottom-up approach of OBDDs with the top-down approach of DPLL. A successful hybrid of these two would have profound impact on symbolic model checking.

Biographical Information

Randal E. Bryant is Dean of the Carnegie Mellon University School of Computer Science. He has been on the faculty at Carnegie Mellon for 21 years, starting as an Assistant Professor and progressing to his current rank of University Professor.

Dr. Bryant's research focuses on methods for formally verifying digital hardware, and more recently some forms of software. His 1986 paper on symbolic Boolean manipulation using Ordered Binary Decision Diagrams (BDDs) has the highest citation count of any publication in the Citeseer database of computer science literature. In addition, he

has developed several techniques to verify circuits by symbolic simulation, with levels of abstraction ranging from transistors to very high-level representations.

Dr. Bryant is a fellow of the IEEE and the ACM, as well as a member of the National Academy of Engineering. His awards include the 1997 ACM Kanellakis Theory and Practice Award (shared with Edmund M. Clarke, Ken McMillan, and Allen Emerson) for contributing to the development of symbolic model checking, as well as the 1989 IEEE W.R.G. Baker Prize for the best paper appearing in any IEEE publication during the preceding year.

Dr. Bryant received his B.S. in Applied Mathematics from the University of Michigan in 1973, and his PhD from MIT in 1981. He was on the faculty at Caltech from 1981 to 1984.

1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019

The Birth of Model Checking

Edmund M. Clarke
Carnegie Mellon University

1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019

Model Checking did not originate in a historical vacuum. There was an important problem – concurrent system correctness – that needed to be solved. There was also considerable research on modeling concurrent systems using various types of state machines and on computing the reachable state spaces of the state machines. Computers also became much more powerful. Perhaps most importantly, researchers including Pnueli, Lamport, and Owicki realized that temporal logic, which dates back to the 1950s if not earlier, was the ideal specification language for concurrent systems.

In my talk, I will trace the development of the ideas that led up to Model Checking starting in the mid 1970s. I will also discuss the crucial early years, extending into the early 1990s, when so many of the ideas we associate with model checking and take for granted (counterexamples, for instance) were introduced. Below is a detailed outline of my talk.

The Original Problem: Verifying Concurrent Systems

- My heresy: Concurrent Systems are often easier to verify than sequential code!

Early work on Automatic Verification of Concurrent Systems

- Research by the Petri net community
- Protocol verification by Bochmann, Holtzmann, and others

Fixpoint Theorems and The Mu-Calculus

Dataflow Analysis

- Killdall
- Ullman and his students
- Cousot

My PhD Thesis and My Early Research on Concurrent Systems

- Program invariants as fixed points
- The characterization problem for Hoare Logics
- Abstract interpretation for concurrent programs

Temporal Logic and How It Originated

Temporal Logic and Program Verification
- Burstall 74, Kroeger 77, and Pnueli 77

Pnueli's Seminal 1977 Paper
- Did he invent model checking?

Branching Time Logics
- Emerson and Clarke 1980
- Ben-Ari, Manna, and Pnueli 1981

Model Checking - The way Allen Emerson and I defined it in 1981

Model Checking Is Seriously Proposed For Verification of Concurrent Systems
- Clarke and Emerson 81
- Quielle and Sifakis 82
- Similarities and differences

Clarke, Emerson, and Sistla (83 / 86) and The EMC Model Checker
- Linear algorithm
- Fairness constraints

Hardware Verification using Model Checking
- Mishras thesis
- Dills thesis

The MCB Model Checker and Counterexamples

The Complexity of LTL Model Checking
- Sistla and Clarke 82 (PSPACE completeness)
- Lichtenstein and Pnueli 1985 (linear in size of model)
- Emerson and Lei 1985 (CTL* Model Checking)

The MCB Model Checker and Counterexamples

Automata Theoretic Model Checking

Links to Process Algebra
- Bisimulation and stuttering bisimulation
- Characterizing Kripke Structures in temporal logic

Symbolic Model Checking and the partial order reduction

Methods for Dealing with Very Complex Systems

Landmark Events In Model Checking Since The Early 1990s

Conclusion and Challenges for the Future

Biographical Information

Edmund M. Clarke received a B.A. degree in mathematics from the University of Virginia, Charlottesville, VA, in 1967, an M.A. degree in mathematics from Duke University, Durham NC, in 1968, and a Ph.D. degree in Computer Science from Cornell University, Ithaca NY, in 1976. After receiving his Ph.D., he taught in the Department of Computer Science, Duke University,for two years. In 1978 he moved to Harvard University, Cambridge, MA where he was an Assistant Professor of Computer Science in the Division of Applied Sciences. He left Harvard in 1982 to join the faculty in the Computer Science Department at Carnegie-Mellon University, Pittsburgh, PA. He was appointed Full Professor in 1989. In 1995 he became the first recipient of the FORE Systems Professorship, an endowed chair in the School of Computer Science.

Dr. Clarke's interests include software and hardware verification and automatic theorem proving. In his Ph.D. thesis he proved that certain programming language control structures did not have good Hoare style proof systems. In 1981 he and his Ph.D. student Allen Emerson first proposed the use of Model Checking as a verification technique for finite state concurrent systems. His research group pioneered the use of Model Checking for hardware verification. His group also developed symbolic Model Checking using BDDs. This important technique was the subject of Kenneth McMillan's Ph.D. thesis, which received an ACM Doctoral Dissertation Award. In addition, his research group devèloped the first parallel resolution theorem prover (Parthenon) and the first theorem prover to be based on asymbolic computation system (Analytica).

Dr.Clarke has served on the editorial boards of Distributed Computing, Logic and Computation, and IEEE Transactions in Software Engineering. He is the former editor-in-chief of Formal Methods in Systems Design. He is on the organizing committee of Logic in Computer Science (LICS) and on the steering committee ofComputer-Aided Verification (CAV). He received a Technical Excellence Award from the Semiconductor Research Corporation in 1995 and an Allen Newell Award for Excellence in Research from the Carnegie Mellon Computer Science Department in 1999. He was a co-winner along with Randy Bryant, Allen Emerson, and Kenneth McMillan of the ACM Kanellakis Award in 1999 for the development of Symbolic Model Checking. In 2004 he received the IEEE Harry H. Goode Memorial Award for significant and pioneering contributions to formal verification of hardware and software systems, and for the profound impact these contributions have had on the electronics industry. He was elected to the National Academy of Engineering in 2005 for contributions to the formal verification of hardware and software correctness. Dr. Clarke is a Fellow of the Association for Computing Machinery and the IEEE Computer Society, and a member of Sigma Xi and Phi Beta Kappa.

1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019

A Retrospective on Murphi

David Dill
Stanford University

Murphi is an explicit-state model checker for safety properties. It has been widely used in academia and industry, especially for multiprocessor cache coherence protocols. The success of Murphi in its "ecological niche" is due to several research innovations as well as some good engineering decisions. This talk will review the history of the development of the tool, some of the interesting ideas in it, problems to which it has been applied, and the ad hoc and somewhat random path that lead to a working tool.

Biographical Information

David L. Dill is a Professor of Computer Science and, by courtesy, Electrical Engineering at Stanford University. He has been on the faculty at Stanford since 1987. He has an S.B. in Electrical Engineering and Computer Science from Massachusetts Institute of Technology (1979), and an M.S and Ph.D. from Carnegie-Mellon University (1982 and 1987).

Prof. Dill has research interests in a variety of areas, including computational systems biology and the theory and application of formal verification techniques to system designs, including hardware, protocols, and software. He has also done research in asynchronous circuit verification and synthesis, and in verification methods for hard real-time systems. He was the Chair of the Computer-Aided Verification Conference held at Stanford University in 1994. From July 1995 to September 1996, he was Chief Scientist at 0-In Design Automation. He is an IEEE Fellow and an ACM Fellow.

The Beginnings of Model Checking: A Personal Perspective

E. Allen Emerson
University of Texas

Model checking realizes in small part the Dream of Leibniz to permit calculation of the truth value of formalized assertions. Model checking provides an automated method for verifying concurrent systems that hinges on efficient graph-theoretic reachability analysis. At the time of its introduction in the early 1980's, the prevailing paradigm for verification was a manual one of proof-theoretic reasoning using formal axioms and inference rules oriented towards sequential programs. The need to encompass concurrent programs, and the desire to avoid the difficulties with manual, deductive proofs motivated the development of model checking.

Key mathematical ingredients underlying model checking include:

- Temporal logic, a flexible and expressive formalism to describe ongoing behavior of concurrent programs; temporal logic is arguably the most important factor in the success of model checking.
- structures to represent the behavior of finite state synchronization skeletons for concurrent systems.
- the Mu-calculus, a very general and highly expressive temporal logic that defines correctness in terms of fixpoints, permitting the systematic description of efficient graph search algorithms.
- the Tarski-Knaster Theorem for evaluating fixpoint expressions.
- Automata on infinite objects for specification of properties, modeling of concurrent systems, and verification of correctness.
- Abstractions to simplify and compress the representations of programs.

Despite being hampered by state explosion, over the past 25 years model checking has had a substantive impact on program verification efforts. Formal Verification

has progressed from discussions of how to manually prove programs correct to the routine, automated, model-theoretic verification of many programs.

We will discuss the theoretical and pragmatic themes underlying this transformation, and then go on to make some forecasts as to what might be needed in the future.

1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019

Biographical Information

E. Allen Emerson earned his BS in mathematics at the University of Texas and his PhD in applied mathematics at Harvard, where he worked with Ed Clarke. In 1981 Clarke and Emerson introduced model checking. Emerson has continued to make contributions to areas in formal methods including verification, temporal logic, and automata. He serves on the editorial boards of leading formal methods journals. He is co-winner of the ACM Kanellakis Theory and Practice Award. He is now an Endowed Professor of Computer Sciences at The University of Texas.

Accelerating the Industrial Deployment of Model-Checking for SW Verification: Lessons from 10 Years of Model-Checking Deployment of HW Verification in Intel

Limor Fix
Intel Research Pittsburgh

1 Introduction

Today hardware formal verification tools are offered as products by many EDA companies. However, 15 years ago no company has offered these tools and only few large hardware companies have developed hardware verification CAD tools in house. In Intel, the development of the formal verification system has started in 1990 as an experimental research project and in 1995 deployment have started in two leading design projects in Santa Clara and Portland. Today, most Intel design projects use formal property verification tools, some more intensively than others. The challenge of deploying the new validation technology and methodology was very big. The technology was not mature enough and offered only limited capacity and moreover there is a lot of risk in changing the way the design is conducted and therefore changing it was very hard. In this extended abstract, we describe the evolution of Intel formal property verification tools and the learnings that have been gathered over the years. Some technologies and some methodologies were more successful than others. In particular, linear temporal logic, modular verification, automated abstraction, embedded assertions, and bounded model checking have been key enablers to the success of hardware formal verification. The paper ends with discussion about formal verification of software and how we may accelerate its industrial deployment.

2 Hardware Verification in Intel Since 1995

Intel engineers have been using formal verification tools to verify properties of hardware designs. The first generation of Intel formal property verification system,

called Prover, included an enhanced version of SMV, and a specification language, called FSL, that was an hardware linear temporal language inspired by LTL. The compiler for FSL translated the linear logic into automata using tableau-like algorithms. Two lead CPU design teams used Prover for a period of four years. Both teams reported successful usage of the new verification technology and in particular high quality bugs have been found. Very important leanings were generated by the two design teams about how, where, when and by whom hardware formal verification tools should be used and moreover the remaining technology challenges were identified. Automated abstraction and Modular verification using assume-guarantee paradigm were used to overcome the limited capacity of the tools. Properties were developed to capture the intended behavior of the inputs and the outputs of each module. Only selected areas of the CPU were formally verified. These were areas of high risk in which new complex functionality was added. The decision on when to use the formal tools was not easy. On the one hand, using the tools very early in the design cycle, even before the entire design was coded and thus before the RTL simulation had started, was very successful. Bugs were revealed early and did not even reach the early simulation models. On the other hand, since the RTL was unstable at that time the RTL changes required recoding of the properties and assumptions again and again. The team ended up using the tools relatively late and have indicated that support for early verification would be very beneficial. We have encouraged both the validation and the RTL-design groups to use Prover. In terms of number of users, we had better success with the validation groups. However, the few designers that have used the tools indicated bigger success in finding more bugs with less efforts. It became clear that developing good specification was very difficult. In particular, it was hard for designers to develop high level properties that do not mimic the details of the implementation, it was hard to train the designers to use a linear temporal language, it was impossible to know if enough properties were developed to express the entire desired behavior of the design, and it was hard to maintain the properties since the design was changing and as a results the properties had to be changed accordingly. Other challenges were also identified, one obvious challenge was the limited capacity of the model checker. As a result, an assume-guarantee approach was developed, the design was divided into smaller components and assumptions were placed on the inputs of the verified components. However, the introduction of assumptions created a set of additional challenges: it was hard to know which assumptions were needed, it was hard to identify circular reasoning, and it was hard to make sure all assumptions were verified. The most important learning from the first usage of formal verification tools was the need to either identify which traditional validation activities can be replaced by the formal verification effort. Or, alternatively, the need to integrate smoothly the formal verification activity into the rest of the design and validation

efforts and to minimize any additional manual effort by either the designers or the validators. In the following years, Intel formal verification system has developed very fast, moving from a single BDD-based model checking engine to multiple engines. A SAT-based model checker, a symbolic simulation engine and a parallel solver were added. Formal specification coverage tools were added to be able to measure the completeness of the set of properties that have been developed. A database of properties and assumptions was developed to help detect circular reasoning and track the status of all properties. With high effort, a new generation of the specification language, called ForSpec, was developed. This language has two versions, a standalone version in which the properties are developed in a separate file detached from the RTL design and an embedded version in which properties are developed as embedded assertions inside the RTL model, that is, as part of the Verilog code. In 2003, Intel has donated ForSpec to Accelera, part of the effort to make ForSpec an IEEE standard for formal verification language. The resulting IEEE standard, has adopted major parts of ForSpec standalone version. The IEEE standard for SystemVerilog has adopted major parts of ForSpec embedded version for the SystemVerilog assertions. In the last two years we have extended the use of formal verification technology to other parts of the design. A very successful system has been developed for the verification of microcode.

3 Industrial deployment of model checking for SW verification

A large body of very successful research already exists in the area of software formal verification. The goal of wide spread deployment of these techniques has not been achieved. As our experience in hardware verification taught us the following questions need to be answered: how, where, when and by whom. Below I present my beliefs that are based only on my hardware experience and of cause the future may prove me wrong. As for How, embedded assertions have been proven very successful in hardware verification. For assertions to be successful, they need to be dense enough. A very successful method to increase the number of assertions is to generate some of them automatically and to develop a library of parameterized assertions that express common requirements for software correctness. Once assertions are embedded in the program they need to be utilized for several proposes, for example, for compiler optimization and for debugging using gdb-like debuggers. Database of assertions need to be generated automatically from the program code for managing the status of the assertions. As for Where, today parallel programming becomes a necessity due to the power wall in silicon technology. Most

state-of-the-art computing devices have multiple processing units and the move to chip multiprocessors is happening very fast. New programming paradigms are being developed to combat the difficulties of parallel programming. The new developed languages should include embedded assertions as integral part of the language. Additional area in which formal verification of software should be applied to is security. The problem of viruses, spyware, and worms is growing fast and has very high costs. Extra efforts to reduce vulnerability of software are likely to be invested to prevent these costs. As for When, as with hardware, assertions development should become an integral part of writing software and assertions should be embedded in the code while the code is generated and they should be as dense as possible. As for By whom, in large software companies, just as in hardware companies, large validation groups are focused on raising the quality of the code by intensive testing. I believe most assertions need to be developed by the programmers themselves and should be always turned on. The validation teams may add more assertions later and most importantly they need to work with the assertion database to complete the verification of all assertions.

Biographical Information

Limor is the Associate Director of Intel Research Pittsburgh. Limor is an Intel Principal Engineer and she has a PhD in Computer Science from the Technion, Israel. After graduation, Limor conducted post-doc research in Cornell University and then she joined Intel in Israel. For 10 years Limor has led a major change in Intel's validation methodology. She developed innovative formal verification tools and methodology that have been widely adopted by Intel's design teams. Limor has published more than 30 papers, she was invited to more than 15 technical program committees of leading international conferences. In the last three years Limor was on the Executive Committee of DAC - the Design Automation Conference.

From Graph Models to Game Models

Thomas A. Henzinger
EPFL Lausanne & UC Berkeley

A model that preserves the individuality of multiple actors in a transition system is a game model. The resulting game is played on the state space of the system. The players represent different processes of the system, or the environment. In a given state, the choices made by the players determine the successor state. Some of the players may compete, others collaborate, with the objective of satisfying a temporal specification. A graph model (or Kripke structure) is the special case of a game model where there is only a single player (who resolves nondeterminism). Game solving, therefore, is the generalization of model checking to two or more players.

Game models are necessary to ask and answer certain important questions about systems. In particular, the problem of synthesizing a reactive system that satisfies a temporal specification is a problem of constructing a winning strategy. Other applications include compatibility checking for components, receptiveness and refinement checking for models, and test-case generation. Moreover, game models offer the flexibility to limit the knowledge of a player when making a choice, to allow a player probabilistic choices, and to combine the choices of different players in synchronous or asynchronous fashion. The precise formalization of a game will determine the difficulty of solving it. We survey classical and recent results on games with omega-regular winning conditions, as well as their applications in system design and verification.

Biographical Information

Thomas A. Henzinger is professor of Computer and Communication Sciences at EPFL in Lausanne, Switzerland, and adjunct professor of Electrical Engineering and Computer Sciences at the University of California, Berkeley. He holds a Dipl.-Ing. degree in Computer Science from Kepler University in Linz, Austria, an M.S. degree in Computer and

Information Sciences from the University of Delaware, and a Ph.D. degree in Computer Science from Stanford University (1991). He was assistant professor of Computer Science at Cornell University, professor of Electrical Engineering and Computer Sciences at the University of California, Berkeley, and director at the Max-Planck Institute for Computer Science in Saarbrücken, Germany. His research focuses on modern systems theory, especially models, algorithms, and tools for the design and verification of software, hardware, and embedded systems. His HyTech tool was the first model checker for mixed discrete-continuous systems. He is a member of Academia Europaea, a member of the German Academy of Sciences (Leopoldina), and a fellow of the IEEE.

New Challenges in Model Checking

Gerard J. Holzmann, Rajeev Joshi, Alex Groce
Jet Propulsion Laboratory, California Institute of Technology

Abstract

In the last twenty-five years, the notion of performing software verification with logic model checking techniques has evolved from intellectual curiosity to accepted technology with significant potential for broad practical application. In this symposium contribution we look back at the main steps in this evolution and illustrate how the challenges have changed over the years, as we sharpened our theories and tools. In the full paper we will then discuss a typical challenge in software verification that we face today - and that perhaps we can look back on in another 25 years as having inspired the next logical step towards a broader integration of model checking into the software development process. In this summary, we set the scene.

Keywords. Logic model checking, software verification, software reliability, software structure, grand challenge project, flash file system challenge.

Looking back on the last twenty-five years of development in logic model checking, we can recognize some patterns in what we collectively considered to be the main obstacles to a broader applications of this technique to *software* (instead of hardware) verification problems. As some of the obstacles were overcome, new challenges were identified and targeted. The following brief list illustrates these developments based primarily on the evolution of what ultimately became the SPIN model checker.

1. ***Specification Formalisms:*** Our initial challenge, in a period that we can indicate very approximately as 1975-1985, was to find usable formalisms for constructing ***models*** with provable properties. The focus in this period was on the identification of specification formalisms that could facilitate or enable formal analysis. Many different types of formalisms were tried, but ultimately automata-based models were found to provide the most solid foundation for tool-based analysis. The earliest predecessor of SPIN, the *pan* tool from 1980, for instance, was based on a process algebra. This tool was succeeded in 1983 by a tool called *trace*, and in that transition the tool's foundation changed to automata, leading the way in 1989 for SPIN to conform to

the automata-theoretic foundation for logic model checking from [5]. Some early work on automata based checking techniques includes the work on the Approver tool of Jan Hajek in 1978 [0]. Alas the algorithms used in this tool were never published and it therefore had little influence.

2. ***Efficient Algorithms:*** The next challenge, between roughly 1985 and 1995, was in developing good *data structures* and especially efficient ***algorithms*** to improve the range and efficiency of model checking systems. This development produced BDD-based and symbolic verification methods, as well as the partial order reduction methods that are at the core of logic model checking systems today. Partial order reduction was integrated into SPIN in the early nineties, based on work by Doron Peled.

3. ***Model Extraction*** **from Code:** The third challenge, between 1995 and 2005, was to find ways to apply model checking techniques not to ***models*** of code, but more directly to ***implementation level code***, using software abstraction and model extraction techniques. This work led to an extension of the SPIN model checker with support for embedded software in abstract models [2]. This change enabled the application of SPIN to the verification of implementation level software for call processing in a commercial voice and data switch, which may well have been the first application of formal software verification at a full industrial scale [1]. Similarly, this challenge led to the successes at Microsoft in the formal verification of device driver code [4], and the work at Stanford on the CMC model checker [3].

4. ***Today's Challenge:*** This brings us to the fourth, and current, challenge for work that may well turn out to define the primary emphasis for our work in logic model checking for the period 2005 to 2015. This fourth challenge is to find effective ways to ***structure software*** such that formal verification techniques, and especially logic model checking techniques, become simpler to use and more effective in identifying potential violations of correctness properties in executable code.

Our full symposium contribution will be focused on a description of this new challenge with an example of an application to mission critical software development for spacecraft.

Acknowledgements. The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

[0] J. Hajek, Progress Report on the Automatic and Proven Protocol Verifier (Approver), Computer Communication Review, ACM SigComm 8/1, January 1978, 15-16.

[1] G.J. Holzmann, M.H. Smith, Automating software feature verification, Bell Labs Technical Journal, Vol. 5, No. 2, pp. 72-87, April-June 2000.

[2] G.J. Holzmann, R. Joshi, Model-driven software verification, Proc. 11th SPIN Workshop, Barcelona, Spain, April 2004, Springer-Verlag, LNCS 2989, pp. 77-92.

[3] M. Musuvathi, D.Y.W. Park, A. Chou, D.R. Engler, D.L. Dill, CMC: A pragmatic approach to model checking real code, Proc. Fifth Symposium on Operating Systems Design and Implementation, Dec. 2002.

[4] T. Ball, S.K. Rajamani, Automatically Validating Temporal Safety Properties of Interfaces, Proc. 8th SPIN Workshop on Model Checking Software, Toronto, May 2001, Springer-Verlag, LNCS 2057, pp. 103-122.

[5] M. Vardi, P. Wolper, An authomata-theoretic approacj to automatic program verification, Proc. 1st Annual Symposium on Logic in Computer Science, 1986, pp. 332-344.

Biographical Information

Gerard J. Holzmann earned his Ph.D. at Delft University of Technology. He joined Bell Labs in 1980 as a researcher in the Computing Science Research Center in Murray Hill, NJ. In 2003 he joined NASA/JPL in Pasadena, CA, to initiate and lead its new Laboratory for Reliable Software. Dr. Holzmann is the primary author of the SPIN software verification system, which was awarded the ACM Software System Award in 2001. In 2002 Holzmann received the ACM SIGSOFT Outstanding Research Award, and in 2005 he was elected to the U.S. National Academy of Engineering. Holzmann was also one of the recipients of the 2006 ACM Paris Kanellakis Theory and Practice Award. Holzmann has taught courses in software verification and logic model checking techniques at Princeton University, Columbia University, and currently teaches a regular course on model checking techniques at the California Institute of Technology. He has published four books, and holds seven U.S. patents, many of which have nothing to do with formal verification ...

25 Years of Computer-Aided Verification

Bob Kurshan
Cadence Berkeley

I will trace the evolution of the title subject from the laboratory to the marketplace, following a variety of technologies over the daunting hurdles of "technology transfer".

Biographical Information

- Ph.d. mathematics 1968
- research in algebra and approximation theory 1968-1982
- started working on automata-theoretic verification in 1982
- book: Computer-Aided Verification of Coordinating Processes Princeton Univ. Press (1994)
- designed and built the COSPAN verification system (1983-) together with Zvi Har'El, Ronald H. Hardin, and a number of others, based upon the theory described in the book.

COSPAN has been in use (and continuous development) since 1986, having been applied directly to a number of commercial projects inside AT&T, Lucent, NCR and Intel, as well as having been licensed to numerous universities for educational use. In 1998 COSPAN was released commercially by Lucent under the trademark FormalCheck and licensed for resale to Cadence Design Systems, Inc. In that same year, FormalCheck won the Innovation of the Year Award, from EDN. More recently, SAT-based algorithms from Cadence Berkeley Labs have been integrated into COSPAN, which is now marketed by Cadence as IFV.

The Evolution of Symbolic Model Checking

Ken McMillan
Cadence Berkeley

This talk will trace the evolution of symbolic model checking over nearly two decades. We will note several general trends. In the early years the emphasis was on increasingly sophisticated representations and image computation methods. This includes various generalizations, such as regular model checking, and a proliferation of methods for Boolean quantifier elimination. More recently, the trend has been toward approximate fixed point computations, which in effect allow us to prove weaker properties of larger systems. This has resulted in a synthesis of symbolic model checking and abstract interpretation. Still, given all this progress it is surprising to find BDD-based symbolic model checking at the heart of many modern systems.

Biographical Information

Ken McMillan has worked on various aspects of model checking. He began his Ph.D. work with Ed Clarke in 1987, at Carnegie Mellon, where he developed SMV, a symbolic model checker based on Binary Decision Diagrams. He used this system to find errors in the cache coherence protocols of the Encore Gigamax distributed multiprocessor. He also worked on partial order methods based on Petri net unfoldings, state-space search methods for analog circuits, symmetry reductions, controller synthesis and structural induction for process networks. His thesis on symbolic model checking was co-winner of the ACM doctoral dissertation award. He was later recognized for this work by the ACM Paris Kannelakis award, along with Randy Bryant, Ed Clarke, and Alan Emerson.

After taking a year off, he spent a year at Bell Labs and then moved to the just-opened Cadence Berkeley Labs. In the mid 90's, he worked on alternative Boolean representations, decision problems for partial order logics, and deciding correctness conditions for concurrency. He became interested in compositional methods, developing SMV into a compositional proof system, supporting a style of proof by "functional decomposition"

based on model checking. This system was made available on the Internet, and has since been downloaded more than 14000 times.

Later, in the early 2000's, he began work on SAT-based methods in model checking, introducing the proof-based abstraction method, and interpolant-based model checking. He extended the interpolant methods to first-order logic, allowing various applications to predicate abstraction, and is currently working on interpolation-based software model checking.

Ken holds a BS in electrical engineering from the University of Illinois at Urbana (1984), an MS in electrical engineering from Stanford (1986) and a Ph.D. in computer science from Carnegie Mellon (1992). Before his Ph.D. studies, he worked as a chip designer and a biomedical engineer. He lives in Berkeley, California.

1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019

The Merits of Temporal Testers: Transducers Compose while Acceptors Do Not

Amir Pnueli
New York University & Weizmann Institute of Science

A central component in model checking temporal properties of finite-state systems is the translation of an LTL formula into a finite-state ω automaton. According to the standard translation, automaton $\mathcal{A}_\varphi$ corresponds to LTL formula φ if the set of sequences accepted by $\mathcal{A}_\varphi$ is precisely the set of sequences that satisfy φ. This implies that the correspondence between $\mathcal{A}$ and φ is required only at the initial position of each sequence. That is, an infinite state sequence $\sigma : s_0, s_1, \ldots$ is accepted by automaton $\mathcal{A}_\varphi$ iff $(\sigma, 0) \models \varphi$.

In this talk we consider a stronger notion of translation in which, for a formula φ, we construct an automaton T_φ, called a *temporal tester for* φ, which has a Boolean output variable x such that $x = 1$ at position $j \geq 0$ of a sequence σ iff $(\sigma, j) \models \varphi$. A tester for an LTL formula φ can be viewed as a (possibly non-deterministic) *transducer* that keeps observing a state sequence σ and, at every position $j \geq 0$, outputs a boolean value which equals 1 iff $(\sigma, j) \models \varphi$. While acceptors, such as the Büchi automaton $\mathcal{A}_\varphi$, do not compose, transducers do. In Fig. 1, we show how transducers for the formulas φ, ψ, and $p\mathcal{U}q$ can be composed into a transducer for the formula $\varphi\mathcal{U}\psi$.

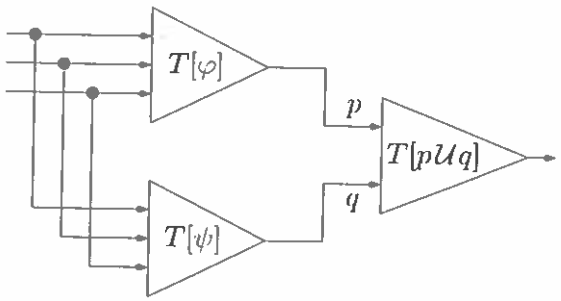


Figure 1: Composition of transducers to form $T[\varphi\mathcal{U}\psi]$.

There are several important advantages to the use of temporal testers as the basis for the construction of automata for temporal formulas:

- The construction is *modular* and *compositional*. Therefore, it is sufficient to specify testers for the basic temporal formulas: $X\,p$ (*next* p), $p\mathcal{U}q$, $Y\,p$ (*previously* p), and $p\mathcal{S}q$, where p and q are assertions (state formulas). Testers for more complex formulas can be derived by composition and variable connections as in Fig. 1.
- The testers for the basic formulas are naturally symbolic. Thus, a general tester, which is a synchronous parallel composition (automata product) of symbolic modules can also be easily represented symbolically.
- As shown in the presentation, the basic processes of model checking and run-time monitoring can be performed directly on the symbolic representation of the testers. There is no need for determinization or reduction to explicit state representation.

In spite of these advantages, the complexity of constructing a transducer (temporal tester) for an arbitrary LTL formula is not worse than that of the lower-functionality acceptor. In its symbolic representation, the size of a tester is linear in the size of the formula. This implies that the worst-case state complexity is exponential.

Due to the property of modularity, we can outline a model-checking algorithm for LTL formulas which is compositional in the structure of the formula. This type of compositionality has been long considered to be a unique feature of CTL, but impossible for LTL. Based on this approach to LTL model checking, we will present a compositional method for model-checking arbitrary CTL* formulas [KP05].

We will show how the notion of temporal testers is easily and naturally extended to deal with many of the new operators introduced in the new hardware property specification language PSL [ZP06]. It can also be adapted to deal with versions of LTL which are evaluated over finite sequences as well as infinite sequences, as is often required in applications of testing and run-time verification.

References

[KP05] Y. Kesten and A. Pnueli. A Compositional Approach to CTL* Verification. *Theor. Comp. Sci.*, 331:397–428, 2005.

[ZP06] A. Zaks and A. Pnueli. PSL Model Checking and Run-time Verification via Testers. In *Formal Methods (FM) 2006*, Lect. Notes in Comp. Sci., McMaster Univeristy Hamilton, Ontario, 2006. Springer-Verlag.

Biographical Information

Amir Pnueli was born in Nahalal, Israel, on Apr. 22, 1941. He finished his B.Sc. degree in Mathematics at the Technion, Haifa, and received his Ph.D. degree in Applied Mathematics at the Weizmann Institute of Science, Rehovot, Israel, submitting a thesis on "Calculation of Tides in the Ocean" in 1967. He switched into Computer Science while being a post-doctoral fellow at Stanford University, and at Watson Research Center, Yorktown. He then returned to Israel to a position of a Senior Researcher in the Department of Applied Mathematics of the Weizmann Institute. In 1973, Prof. Pnueli moved to Tel-Aviv University, where he founded the Department of Computer Science and was its first chairman. In 1981, Pnueli returned to the Weizmann Institute, as a Professor of Computer Science. In 1997, Pnueli was appointed as the head of the "Minerva Center for Verification of Reactive Systems". Prof. Pnueli is the 1996 recipient of the ACM Turing award "For his seminal work introducing temporal logic into computing science and for outstanding contributions to program and system verification." Since 1999, Pnueli is a Professor of Computer Science at the Courant Institute of Mathematical Sciences at New York University.

Additional Honors and Awards

In May 97, Pnueli received an honorary doctorate from the University of Uppsala, Sweden. In December 1998, Amir Pnueli received an honorary doctorate from Universite Joseph Fourier, Grenoble, France. In November 2000 he received an honorary doctorate from Carl von Ossietzky Universitaet Oldenburg in Germany. In 1999, Pnueli was accepted as a foreign member of the (American) National Academy of Engineering. Prof. Pnueli is the 2000 recipient of the Israel prize in the category of exact sciences for his "outstanding contribution in the field of Computer Science". The Israel prize is the most prestigious prize awarded by the state of Israel In 2001, Pnueli was elected to the Israeli Academy of Arts and Sciences. In 2004, he was elected to the European Academy of Sciences (EAS). In 2006 he has been elected to Academia Europeae.

Membership and Editorial Responsibilities

Amir Pnueli is a member of the steering committees of the conferences CAV (Computer Aided Verification), FTRTFT (Formal Techniques for Real-Time and Fault-Tolerant Systems), Workshop Series on Hybrid Systems, and ICTL (International Conference on Temporal Logic). He is an associate editor of "Science of Computer Programming", the "Journal of Logic and Computations", "Formal Methods in System Design", and IGPL (Journal of the Interest Group in Pure and Applied Logics). He is a member of the Beirat (board of governors) for the Leibniz Center for Research in Computer Science at the Hebrew University, since its inception in 1985. Pnueli is a member of the IFIP's WG2.2 working group on Formal Description of Programming. He has been on the program committees of POPL, FOCS, LICS, CAV, FTRTFT, Concur, PODC, and ICALP.

Research Activity

Prof. Pnueli is mainly known for the introduction of temporal logic into Computer Science; his work on the application of temporal logic and similar formalisms for the specification and verification of reactive systems; the identification of the class of "Reactive Systems"

as systems whose formal specification, analysis, and verification require a distinctive approach; and the development of a rich and detailed methodology, based on temporal logic, for the formal treatment of reactive system; extending this methodology into the realm of real-time systems; and more recently, introducing into formal analysis the models of hybrid systems with appropriate extension of the temporal-logic based methodology.

Beside his more theoretical work, concerning a complete axiom system and proof theory for program verification by temporal logic, he also contributed to algorithmic research in this area. He developed a deductive system for linear-time temporal logic and model-checking algorithms for the verification of temporal properties of finite-state systems. Together with David Harel, Pnueli worked on the semantics and implementation of Statecharts, a visual language for the specification, modeling, and prototyping of reactive systems. This language has been applied to avionics, transport, and electronic hardware systems. His current research interests involve synthesis of reactive modules, automatic verification of multi-process systems, and specification methods that combine transition systems with temporal logic.

Together with Zohar Manna, he is the author of a 3-volumes textbook on Temporal Logic and its application to Reactive Systems of which the first two volumes are:

1. Z. Manna & A. Pnueli: "The Temporal Logic of Reactive and Concurrent Systems: Specification", Springer-Verlag, 1991.
2. Z. Manna & A. Pnueli: "Temporal Verification of Reactive Systems: Safety", Springer-Verlag, 1995.

Industrial Activities

In 1971, Prof. Pnueli co-founded the software company Mini-Systems, which until 1982 was the sole software provider for Scitex, Israel, manufacturers of computer aided design systems in the color press and graphic printing areas. During this period, Pnueli designed and supervised several real time mini-computer systems, including message switching, Computer-Aided Teaching, on-line instrument testing, military real time systems, operating systems and compilers.

In 1984, Mini-Systems was acquired by Scitex, and Prof. Pnueli moved on to found, together with two of his previous partners and Prof. David Harel, also from the Weizmann Institute, the company AdCad. In the years 1984-1989, Prof. Pnueli supervised and designed (together with David Harel) the first version of the Statemate system. This implementation effort required continuous research to clarify the semantics of synchronous languages, among which, Statecharts occupy a prominent position.

The company AdCad later evolved into i-Logix, a firm constructing Computer Aided Software Engineering tools for the specification and design of real time reactive embedded systems, and which is the current producer of the various versions of the Statemate systems and its later derivatives.

He is married and has 3 children, and 3 grandchildren.

From Church and Prior to PSL

Moshe Y. Vardi
Rice University

One of the surprising developments in the area of program verification is how ideas introduced originally by Prior in the 1950s ended up yielding by 2004 an industrial-standard property-specification language called PSL. This development was enabled by the equally unlikely transformation of mathematical machinery, introduced by Büchi in the early 1960 for second-order number theory, into effective algorithms for model checking tools. This talk will attempt to depict the tangled skein of this development.

Biographical Information

Moshe Y. Vardi is the George Professor in Computational Engineering and Director of the Computer and Information Technology Institute at Rice University. He chaired the Computer Science Department at Rice University from January 1994 till June 2002. Prior to joining Rice in 1993, he was at the IBM Almaden Research Center, where he managed the Mathematics and Related Computer Science Department. His research interests include database systems, computational-complexity theory, multi-agent systems, and design specification and verification. Vardi received his Ph.D. from the Hebrew University of Jerusalem in 1981. He is the author and co-author of over 300 technical papers, as well as a book titled "Reasoning about Knowledge", and the editor of several collections. Vardi is the recipient of three IBM Outstanding Innovation Awards, a co-winner of the 2000 Gödel Prize, and a co-winner of the 2005 ACM Paris Kanellakis Award for Theory and Practice. He holds honorary doctorates from the University of Saarland, Germany, and the University of Orleans, France. Vardi is an editor of several international journals and the president of the International Federation of Computational Logicians. He is Guggenheim Fellow, as well as a Fellow of the Association of Computing Machinery, the American Association for the Advancement of Science, and the American Association for Artificial Intelligence. He was designated Highly Cited Researcher by the Institute for Scientific Information, and was elected as a member of the US National Academy of Engineering and the European Academy of Sciences. He recently co-chaired the ACM Task Force on Job Migration.