Stefan Edelkamp
Alessio Lomuscio (Eds.)

LNAI 4428

Model Checking and Artificial Intelligence

4th Workshop, MoChArt IV
Riva del Garda, Italy, August 2006
Revised Selected and Invited Papers

Springer

Stefan Edelkamp Alessio Lomuscio (Eds.)

Model Checking and Artificial Intelligence

4th Workshop, MoChArt IV
Riva del Garda, Italy, August 29, 2006
Revised Selected and Invited Papers

Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editors

Stefan Edelkamp
University of Dortmund
Computer Science Department
Otto-Hahn-Straße 14, 44227 Dortmund, Germany
E-mail: stefan.edelkamp@cs.uni-dortmund.de

Alessio Lomuscio
Imperial College London
Department of Computing
180 Queen's Gate, London SW7 2AZ, UK
E-mail: a.lomuscio@cs.ucl.ac.uk

Library of Congress Control Number: 2007932185

CR Subject Classification (1998): I.2.3, I.2, F.4.1, F.3, D.2.4, D.1.6

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN 0302-9743
ISBN-10 3-540-74127-5 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-74127-5 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12103942 06/3180 5 4 3 2 1 0

Lecture Notes in Artificial Intelligence 4428

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

Preface

Exploration of very large search spaces lies at the heart of many disciplines in computer science and engineering, especially systems verification and artificial intelligence. In particular, the technique of *model checking* is used to automatically verify the properties of a system. In the model checking approach, verifying that a system S satisfies a property P is investigated by automatically checking the satisfiability of the expression $M_S \models \phi_P$, where M_S is a suitable model representing all evolutions of S, and ϕ_P is a logical formula capturing the property P to be checked.

Model checking and artificial intelligence have enjoyed a healthy interchange of ideas over the past few years. On the one hand, model checking techniques have benefited from efficient search algorithms developed in artificial intelligence thereby increasing their efficiency, on the other, model checking techniques have been extended to deal with typical artificial intelligence formalisms, such as epistemic logics, thereby permitting the verification of systems based on artificial intelligence concepts. In addition to this, there remains a keen interest among researchers to use model checking to solve planning problems.

The forth MoChart workshop aimed at bringing together researchers interested in the interplay of these areas. The workshop was held as a satellite workshop of ECAI 2006, the 17th biennial European conference on Artificial Intelligence. Previous workshops were held in San Francisco in 2005 (as a satellite workshop of CONCUR 2005), Acapulco in 2003 (as a satellite workshop of IJCAI 2003), and Lyon in 2002 (as a satellite workshop of ECAI 2002).

This volume includes extended versions of eight of the nine papers selected for presentation at the workshop after a selective round of reviews, as well as three further papers selected from submissions to the post-proceedings. An article based on an invited presentation to the workshop is also included. The papers are included in the order in which they were presented at the workshop.

The volume begins with the invited contribution by Bertoli et al. on a broad overview of the use of model checking techniques for safety analysis, diagnosability and synthesis in reactive systems. This is followed by an article by Alechina et al. investigating the reasoning capabilities of resource bounded agents. A contribution by Edelkamp on a variety of genetic algorithms operating on pattern databases represented via OBDDs follows.

Hoffman et al. then discuss efficient optmization methods for model checking real-time systems by introducing predicate abstraction to generate efficient heuristic search functions. Analysis for real-time systems also features in the following paper by Edelkamp and Jabber investigating the performance of secondary storage solutions for three search algorithms. Concluding in this line, Lomuscio et al. present and evaluate algorithms for model checking networks of timed-automata with clock differences against epistemic real-time specifications.

Genetic algorithms feature again in the following article where Araragi and Cho suggest a reinforcement learning technique to check liveness in reactive systems. This is followed by a paper by Pecheur and Raimondi on model checking variants of CTL supporting explicit actions via OBDDs. In the next paper, Viganò proposes a methodology based on SPIN to verify the multi-agent frameworks of e-institutions described by an ad-hoc modelling language.

In a change of topic Kurkowski et al. present a SAT-based methodology for the verification of security protocols by modelling principals via networks of communicating automata. Wijs and Lisser conclude the volume by analyzing variations of distributed beam search algorithms to find solutions to scheduling problems via model checking.

All the papers represent solid contributions to the state of the art in the interplay between artificial intelligence and model checking and provide an interesting overview of the current trends of research worldwide.

We very much enjoyed the workshop and wish to thank the authors for their excellent contributions and the program committee for their outstanding service in selecting the papers.

We conclude by thanking Springer for enthusiastically supporting the idea of publishing the post-proceedings of the event. The British Royal Association of Engineering and the Deutsche Forschungsgemeinschaft are also acknowledged for their generous support.

February 2007

Stefan Edelkamp
Alessio Lomuscio

Organization

Program Committee

Massimo Benerecetti, Università di Napoli (Italy)
Armin Biere, University of Linz (Austria)
Rafael H. Bordini, University of Durham (UK)
Edmund Clarke, CMU (USA)
Alessandro Cimatti, ITC-IRST (Italy)
Stefan Edelkamp, University of Dortmund (Germany)
Enrico Giunchiglia, University of Genoa (Italy)
Jörg Hoffmann, Digital Enterprise Research Institute (Austria)
Froduald Kabanza, Université de Sherbrooke (Canada)
Richard Korf, UCLA, California (USA)
Stefan Leue, University of Konstanz (Germany)
Alessio Lomuscio, Imperial College London (UK)
Ron van der Meyden, University of New South Wales / NICTA (Australia)
Charles Pecheur, Université catholique de Louvain (Belgium)
Wojciech Penczek, University of Podlasie (Poland)
Mark Ryan, University of Birmingham (UK)
Brian Williams, MIT (USA)
Michael Wooldridge, University of Liverpool (UK)

Additional Referees

Shahid Jabbar, University of Dortmund (Germany)
Ansgar Fehnker, NICTA (Australia)

Lecture Notes in Artificial Intelligence (LNAI)

Vol. 4660: S. Džeroski, J. Todorovski (Eds.), Computational Discovery of Scientific Knowledge. X, 327 pages. 2007.

Vol. 4651: F. Azevedo, P. Barahona, F. Fages, F. Rossi (Eds.), Recent Advances in Constraints. VIII, 185 pages. 2007.

Vol. 4632: R. Alhajj, H. Gao, X. Li, J. Li, O.R. Zaïane (Eds.), Advanced Data Mining and Applications. XV, 634 pages. 2007.

Vol. 4617: V. Torra, Y. Narukawa, Y. Yoshida (Eds.), Modeling Decisions for Artificial Intelligence. XII, 502 pages. 2007.

Vol. 4612: I. Miguel, W. Ruml (Eds.), Abstraction, Reformulation, and Approximation. XI, 418 pages. 2007.

Vol. 4604: U. Priss, S. Polovina, R. Hill (Eds.), Conceptual Structures: Knowledge Architectures for Smart Applications. XII, 514 pages. 2007.

Vol. 4603: F. Pfenning (Ed.), Automated Deduction – CADE-21. XII, 522 pages. 2007.

Vol. 4597: P. Perner (Ed.), Advances in Data Mining. XI, 353 pages. 2007.

Vol. 4594: R. Bellazzi, A. Abu-Hanna, J. Hunter (Eds.), Artificial Intelligence in Medicine. XVI, 509 pages. 2007.

Vol. 4585: M. Kryszkiewicz, J.F. Peters, H. Rybinski, A. Skowron (Eds.), Rough Sets and Intelligent Systems Paradigms. XIX, 836 pages. 2007.

Vol. 4578: F. Masulli, S. Mitra, G. Pasi (Eds.), Applications of Fuzzy Sets Theory. XVIII, 693 pages. 2007.

Vol. 4573: M. Kauers, M. Kerber, R. Miner, W. Windsteiger (Eds.), Towards Mechanized Mathematical Assistants. XIII, 407 pages. 2007.

Vol. 4571: P. Perner (Ed.), Machine Learning and Data Mining in Pattern Recognition. XIV, 913 pages. 2007.

Vol. 4570: H.G. Okuno, M. Ali (Eds.), New Trends in Applied Artificial Intelligenee. XXI, 1194 pages. 2007.

Vol. 4565: D.D. Schmorrow, L.M. Reeves (Eds.), Foundations of Augmented Cognition. XIX, 450 pages. 2007.

Vol. 4562: D. Harris (Ed.), Engineering Psychology and Cognitive Ergonomics. XXIII, 879 pages. 2007.

Vol. 4548: N. Olivetti (Ed.), Automated Reasoning with Analytic Tableaux and Related Methods. X, 245 pages. 2007.

Vol. 4539: N.H. Bshouty, C. Gentile (Eds.), Learning Theory. XII, 634 pages. 2007.

Vol. 4529: P. Melin, O. Castillo, L.T. Aguilar, J. Kacprzyk, W. Pedrycz (Eds.), Foundations of Fuzzy Logic and Soft Computing. XIX, 830 pages. 2007.

Vol. 4511: C. Conati, K. McCoy, G. Paliouras (Eds.), User Modeling 2007. XVI, 487 pages. 2007.

Vol. 4509: Z. Kobti, D. Wu (Eds.), Advances in Artificial Intelligence. XII, 552 pages. 2007.

Vol. 4496: N.T. Nguyen, A. Grzech, R.J. Howlett, L.C. Jain (Eds.), Agent and Multi-Agent Systems: Technologies and Applications. XXI, 1046 pages. 2007.

Vol. 4483: C. Baral, G. Brewka, J. Schlipf (Eds.), Logic Programming and Nonmonotonic Reasoning. IX, 327 pages. 2007.

Vol. 4482: A. An, J. Stefanowski, S. Ramanna, C.J. Butz, W. Pedrycz, G. Wang (Eds.), Rough Sets, Fuzzy Sets, Data Mining and Granular Computing. XIV, 585 pages. 2007.

Vol. 4481: J. Yao, P. Lingras, W.-Z. Wu, M. Szczuka, N.J. Cercone, D. Ślęzak (Eds.), Rough Sets and Knowledge Technology. XIV, 576 pages. 2007.

Vol. 4476: V. Gorodetsky, C. Zhang, V.A. Skormin, L. Cao (Eds.), Autonomous Intelligent Systems: Multi-Agents and Data Mining. XIII, 323 pages. 2007.

Vol. 4455: S. Muggleton, R. Otero, A. Tamaddoni-Nezhad (Eds.), Inductive Logic Programming. XII, 456 pages. 2007.

Vol. 4452: M. Fasli, O. Shehory (Eds.), Agent-Mediated Electronic Commerce. VIII, 249 pages. 2007.

Vol. 4451: T.S. Huang, A. Nijholt, M. Pantic, A. Pentland (Eds.), Artifical Intelligence for Human Computing. XVI, 359 pages. 2007.

Vol. 4438: L. Maicher, A. Sigel, L.M. Garshol (Eds.), Leveraging the Semantics of Topic Maps. X, 257 pages. 2007.

Vol. 4434: G. Lakemeyer, E. Sklar, D.G. Sorrenti, T. Takahashi (Eds.), RoboCup 2006: Robot Soccer World Cup X. XIII, 566 pages. 2007.

Vol. 4429: R. Lu, J.H. Siekmann, C. Ullrich (Eds.), Cognitive Systems. X, 161 pages. 2007.

Vol. 4428: S. Edelkamp, A. Lomuscio (Eds.), Model Checking and Artificial Intelligence. IX, 185 pages. 2007.

Vol. 4426: Z.-H. Zhou, H. Li, Q. Yang (Eds.), Advances in Knowledge Discovery and Data Mining. XXV, 1161 pages. 2007.

Vol. 4411: R.H. Bordini, M. Dastani, J. Dix, A.E.F. Seghrouchni (Eds.), Programming Multi-Agent Systems. XIV, 249 pages. 2007.

Vol. 4410: A. Branco (Ed.), Anaphora: Analysis, Algorithms and Applications. X, 191 pages. 2007.

Vol. 4399: T. Kovacs, X. Llorà, K. Takadama, P.L. Lanzi, W. Stolzmann, S.W. Wilson (Eds.), Learning Classifier Systems. XII, 345 pages. 2007.

Vol. 4390: S.O. Kuznetsov, S. Schmidt (Eds.), Formal Concept Analysis. X, 329 pages. 2007.

Vol. 4389: D. Weyns, H.V.D. Parunak, F. Michel (Eds.), Environments for Multi-Agent Systems III. X, 273 pages. 2007.

Vol. 4384: T. Washio, K. Satoh, H. Takeda, A. Inokuchi (Eds.), New Frontiers in Artificial Intelligence. IX, 401 pages. 2007.

Vol. 4371: K. Inoue, K. Satoh, F. Toni (Eds.), Computational Logic in Multi-Agent Systems. X, 315 pages. 2007.

Vol. 4369: M. Umeda, A. Wolf, O. Bartenstein, U. Geske, D. Seipel, O. Takata (Eds.), Declarative Programming for Knowledge Management. X, 229 pages. 2006.

Vol. 4342: H. de Swart, E. Orłowska, G. Schmidt, M. Roubens (Eds.), Theory and Applications of Relational Structures as Knowledge Instruments II. X, 373 pages. 2006.

Vol. 4335: S.A. Brueckner, S. Hassas, M. Jelasity, D. Yamins (Eds.), Engineering Self-Organising Systems. XII, 212 pages. 2007.

Vol. 4334: B. Beckert, R. Hähnle, P.H. Schmitt (Eds.), Verification of Object-Oriented Software. XXIX, 658 pages. 2007.

Vol. 4333: U. Reimer, D. Karagiannis (Eds.), Practical Aspects of Knowledge Management. XII, 338 pages. 2006.

Vol. 4327: M. Baldoni, U. Endriss (Eds.), Declarative Agent Languages and Technologies IV. VIII, 257 pages. 2006.

Vol. 4314: C. Freksa, M. Kohlhase, K. Schill (Eds.), KI 2006: Advances in Artificial Intelligence. XII, 458 pages. 2007.

Vol. 4304: A. Sattar, B.-h. Kang (Eds.), AI 2006: Advances in Artificial Intelligence. XXVII, 1303 pages. 2006.

Vol. 4303: A. Hoffmann, B.-h. Kang, D. Richards, S. Tsumoto (Eds.), Advances in Knowledge Acquisition and Management. XI, 259 pages. 2006.

Vol. 4293: A. Gelbukh, C.A. Reyes-Garcia (Eds.), MICAI 2006: Advances in Artificial Intelligence. XXVIII, 1232 pages. 2006.

Vol. 4289: M. Ackermann, B. Berendt, M. Grobelnik, A. Hotho, D. Mladenič, G. Semeraro, M. Spiliopoulou, G. Stumme, V. Svátek, M. van Someren (Eds.), Semantics, Web and Mining. X, 197 pages. 2006.

Vol. 4285: Y. Matsumoto, R.W. Sproat, K.-F. Wong, M. Zhang (Eds.), Computer Processing of Oriental Languages. XVII, 544 pages. 2006.

Vol. 4274: Q. Huo, B. Ma, E.-S. Chng, H. Li (Eds.), Chinese Spoken Language Processing. XXIV, 805 pages. 2006.

Vol. 4265: L. Todorovski, N. Lavrač, K.P. Jantke (Eds.), Discovery Science. XIV, 384 pages. 2006.

Vol. 4264: J.L. Balcázar, P.M. Long, F. Stephan (Eds.), Algorithmic Learning Theory. XIII, 393 pages. 2006.

Vol. 4259: S. Greco, Y. Hata, S. Hirano, M. Inuiguchi, S. Miyamoto, H.S. Nguyen, R. Słowiński (Eds.), Rough Sets and Current Trends in Computing. XXII, 951 pages. 2006.

Vol. 4253: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part III. XXXII, 1301 pages. 2006.

Vol. 4252: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part II. XXXIII, 1335 pages. 2006.

Vol. 4251: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part I. LXVI, 1297 pages. 2006.

Vol. 4248: S. Staab, V. Svátek (Eds.), Managing Knowledge in a World of Networks. XIV, 400 pages. 2006.

Vol. 4246: M. Hermann, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning. XIII, 588 pages. 2006.

Vol. 4223: L. Wang, L. Jiao, G. Shi, X. Li, J. Liu (Eds.), Fuzzy Systems and Knowledge Discovery. XXVIII, 1335 pages. 2006.

Vol. 4213: J. Fürnkranz, T. Scheffer, M. Spiliopoulou (Eds.), Knowledge Discovery in Databases: PKDD 2006. XXII, 660 pages. 2006.

Vol. 4212: J. Fürnkranz, T. Scheffer, M. Spiliopoulou (Eds.), Machine Learning: ECML 2006. XXIII, 851 pages. 2006.

Vol. 4211: P. Vogt, Y. Sugita, E. Tuci, C.L. Nehaniv (Eds.), Symbol Grounding and Beyond. VIII, 237 pages. 2006.

Vol. 4203: F. Esposito, Z.W. Raś, D. Malerba, G. Semeraro (Eds.), Foundations of Intelligent Systems. XVIII, 767 pages. 2006.

Vol. 4201: Y. Sakakibara, S. Kobayashi, K. Sato, T. Nishino, E. Tomita (Eds.), Grammatical Inference: Algorithms and Applications. XII, 359 pages. 2006.

Vol. 4200: I.F.C. Smith (Ed.), Intelligent Computing in Engineering and Architecture. XIII, 692 pages. 2006.

Vol. 4198: O. Nasraoui, O. Zaïane, M. Spiliopoulou, B. Mobasher, B. Masand, P.S. Yu (Eds.), Advances in Web Mining and Web Usage Analysis. IX, 177 pages. 2006.

Vol. 4196: K. Fischer, I.J. Timm, E. André, N. Zhong (Eds.), Multiagent System Technologies. X, 185 pages. 2006.

Vol. 4188: P. Sojka, I. Kopeček, K. Pala (Eds.), Text, Speech and Dialogue. XV, 721 pages. 2006.

Vol. 4183: J. Euzenat, J. Domingue (Eds.), Artificial Intelligence: Methodology, Systems, and Applications. XIII, 291 pages. 2006.

Vol. 4180: M. Kohlhase, OMDoc – An Open Markup Format for Mathematical Documents [version 1.2]. XIX, 428 pages. 2006.

Vol. 4177: R. Marín, E. Onaindía, A. Bugarín, J. Santos (Eds.), Current Topics in Artificial Intelligence. XV, 482 pages. 2006.

Vol. 4160: M. Fisher, W. van der Hoek, B. Konev, A. Lisitsa (Eds.), Logics in Artificial Intelligence. XII, 516 pages. 2006.

Table of Contents

A Symbolic Model Checking Framework for Safety Analysis, Diagnosis, and Synthesis*

Piergiorgio Bertoli, Marco Bozzano, and Alessandro Cimatti

ITC-irst - Via Sommarive 18 - 38050 Povo - Trento - Italy
{bertoli,bozzano,cimatti}@irst.itc.it

Abstract. Modern reactive control system are typically very complex entities, and their design poses substantial challenges. In addition to ensuring functional correctness, other steps may be required: with safety analysis, the behavior is analyzed, and proved compliant to some requirements considering possible faulty behaviors; diagnosis and diagnosability are forms of reasoning on the run-time explanation of faulty behaviors; planning and synthesis allow the automated construction of controllers that implement desired behaviors. Symbolic Model Checking (SMC) is a formal technique for ensuring functional correctness that has achieved a substantial industrial penetration in the last decade. In this paper, we show how SMC can be used as a convenient framework to express safety analysis, diagnosis and diagnosability, and synthesis. We also discuss how model checking tools can be used and extended to solve the resulting computational challenges.

1 Introduction

In recent years, complex applications increasingly rely on implementations based on software and digital systems. Typical examples are transportation domains (e.g. railways, avionics, space), telecommunications, hardware, industrial control. The design of such complex systems is a very hard task. On the one hand, more and more functionalities must be implemented, in order to provide for flexible, user-configurable products. On the other hand, there is a need to achieve higher degrees of assurance, given the criticality of the functions.

For the above reasons, the engineering of complex systems has witnessed the introduction of model-based design techniques and tools. The idea is to write system models, expressed at different levels of abstraction, and to provide support tools to automatically analyze them. Different languages can be used to express such models; in general, they can be encoded and treated in terms of transition systems. Model checking is a verification technique to check whether a system (modeled as a transition system) satisfies certain requirements (modeled as temporal logic).

Goal of this paper is to draw a unifying view between different aspects of engineering, within the framework of model checking. We show how many different stages in model-based design can be cast in the framework of model checking, and can benefit from the advanced symbolic model checking techniques and tools.

* This work has been partly supported by the E.U.-sponsored project ISAAC, contract no. AST3-CT-2003-501848.

S. Edelkamp and A. Lomuscio (Eds.): MoChart IV, LNAI 4428, pp. 1–18, 2007.

We proceed in order of increasing complexity. We start by defining the problem of model checking, and providing an overview of the available techniques. Then, we consider the field of *safety analysis*: while in model checking the behavior of the system is analyzed under nominal conditions, in safety analysis the problem is to check the behavior of the design in presence of failures. This phase is carried out at design time. The only increase required in the framework is the specification of a selected set of failure mode variables. The next problem is *diagnosis*, that can be seen as the problem of safety analysis carried out at run-time. On one side, only one trace at the time is considered. On the other side, diagnosis is usually performed on systems which provide limited run-time sensing, making the problem much harder. Another interesting and related problem, known as *diagnosability*, is the analysis, *at design time*, of diagnosis capabilities. We conclude with the problem of planning, which in the general setting used in this paper amounts to the problem of *synthesis*, i.e. automatic generation of controllers from specifications. The problem has been addressed in many variations, and has interesting overlappings with diagnosis. In particular, in the case of planning under partial observability, actions must be planned in order to achieve a given amount of information.

This paper is structured as follows. In Section 2 we describe model checking, and overview the main symbolic implementation techniques. In Section 3 we present the ideas underlying safety analysis. In Section 4 we discuss the role of model checking in diagnosis and diagnosability. In Section 5 we discuss planning based on symbolic model checking, and in Section 6 we draw some conclusions, and outline directions for future work.

2 Symbolic Model Checking

Model checking [21,22,45] is a formal verification technique that is widely used to complement classical techniques such as testing and simulation. In particular, while testing and simulation may only verify a limited portion of the possible behaviors of complex, asynchronous systems, model checking provides a formal guarantee that some given specification is obeyed. In model checking, the verified system is modeled as a state transition system (typically of finite size). The specifications are expressed as temporal logic formulæ, that express constraints over the dynamics of systems. Model checking then consists in exhaustively exploring every possible system behavior, to check automatically that the specifications are satisfied. In the case of finite models, termination is guaranteed. Very relevant for debugging purposes, when a specification is not satisfied, a counterexample is produced, witnessing the offending behavior of the system. Formally, model checking relies on the following definition of a *system*:

Definition 1 (System). *A system is a tuple* $\mathcal{M} = \langle \mathcal{S}, \mathcal{S}_i, \mathcal{I}, \mathcal{R} \rangle$ *where:*

- $\mathcal{S}$ *is a finite set of states,*
- $\mathcal{S}_i \subseteq \mathcal{S}$ *is the set of initial states,*
- $\mathcal{I}$ *is a finite set of inputs,*
- $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{I} \times \mathcal{S}$ *is the transition relation*

The transition relation specifies the possible transitions from state to state, triggered by the applications of inputs to the system. For technical reasons, it is required to be total, i.e. for each state there exists at least a successor state. From such a tuple, abstracting away from inputs, one can immediately extract a *state transition graph*, a Kripke structure that only describes transitions from states to states.

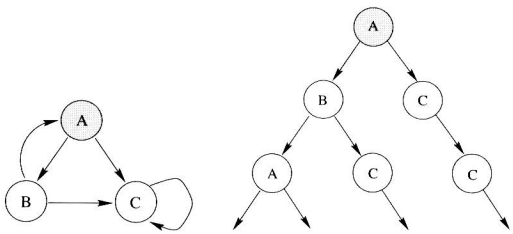

Fig. 1. A State Transition Graph and its unwinding

A path in such a Kripke structure is obtained starting from a state $s \in \mathcal{S}_i$, and then repeatedly appending states reachable through $\mathcal{R}$; since inputs have been abstracted away, a path corresponds to the evolution of the system *for some* sequence of inputs. Given the totality of $\mathcal{R}$, behaviors are infinite. Since a state can have more than one successor, the structure can be thought of as unwinding into an infinite tree, representing all the possible executions of the system starting from the initial states. Figure 1 shows a state transition graph and its unwinding from the initial (colored) state. A Kripke structure is typically associated with a set of propositions $\mathcal{P}$, and with a labeling function that maps each state onto a truth assignment to such propositions. In the following, we assume that one truth assignment to the variables in $\mathcal{P}$ is associated to at most one state, and we write $s \models p$ to indicate that a proposition p holds in a state s.

Traditionally, two temporal logics are most commonly used for model checking, CTL and LTL [31]. Computation Tree Logic (CTL) is interpreted over the computation tree of the Kripke structure, while Linear Temporal Logic (LTL) is interpreted over the set of its paths. These two logics have incomparable expressive power, and differ in how they handle branching in the underlying computation tree: CTL temporal operators quantify over the paths departing from a given state, while LTL operators describe properties of all possible computation paths.

Model checking is the problem of deciding whether a certain temporal formula φ holds in a given Kripke structure $\mathcal{M}$ (see [24] for a detailed overview). In the following we use the notation $\mathcal{M} \models \varphi$. The first model checking algorithms used an explicit representation of the Kripke structure as a labeled, directed graph [21,22,45]. Explicit state model checking is based on the exploration of the Kripke structure based on the expansion and storage of individual states. Over the years, explicit state model checking has reached impressive performance (see for instance the SPIN model checker [38]). The key problem, however, is that explicit state techniques are subject to the so-called state explosion problem, i.e. they need to explore and store the states of the state transition graph. In industrial sized systems, this amounts to an extremely large number of

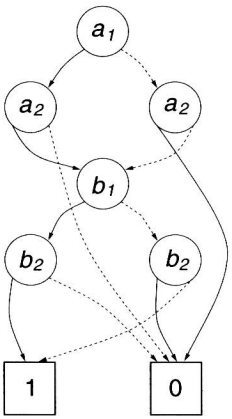

Fig. 2. A BDD for the formula $(a_1 \leftrightarrow a_2) \wedge (b_1 \leftrightarrow b_2)$

states. In fact, the Kripke structure is typically the result of the combination of a number of components (e.g. the communicating processes in a protocol), and the size of the resulting structure may be exponential in the number of components.

A major breakthrough was enabled by the introduction of *symbolic model checking* [40]. In symbolic model checking, rather than individual states and transitions, the idea is to manipulate sets of states and transitions, using a logical formalism to represent the characteristic functions of such sets [26,43,52,15]. Since a small logical formula may admit a large number of models, this results in most cases in a very compact representation which can be effectively manipulated. Each state is presented by an assignment to the propositions (variables) in $\mathcal{P}$ (equivalently, by the corresponding conjunction of literals). A set of states is represented by the disjunction of the formulae representing each of its states. The basic set theoretic operations (intersection, union, projection) are given by logical operations (such as conjunction, disjunction, and quantification). In the following we use x to denote the vector of variables representing the states of a given system; we write $\mathcal{S}_i(x)$ for the formula representing the initial states. A similar construction can be applied to represent inputs (for which we use a vector of variables i). A set of "next" variables x' is used for the state resulting after the transition: a transition from s to s' is then represented as a truth assignment to the current and next variables. We use $\mathcal{R}(x, i, x')$ for the formula representing the transition relation expressed in terms of those variables.

Obviously, the key issue is the use of an efficient logical representation. The first one used for symbolic model checking was provided by Ordered Binary Decision Diagrams [12,13] (BDDs for short). BDDs are a representation for boolean formulas, which is canonical once an order on the variables has been established. This allows equivalence checking in constant time. Figure 2 depicts the BDD for the boolean formula $(a_1 \leftrightarrow a_2) \wedge (b_1 \leftrightarrow b_2)$, using the variable ordering a_1, a_2, b_1, b_2. Solid lines represent "then" arcs (the corresponding variable has to be considered positive), dashed lines represent "else" arcs (the corresponding variable has to be considered negative). Paths from the root to the node labeled with "1" represent the satisfying assignments of the represented boolean formula (e.g., $a_1 \leftarrow 1, a_2 \leftarrow 1, b_1 \leftarrow 0, b_2 \leftarrow 0$).

Such a powerful logical computation machinery provides the ideal basis for the implementation of algorithms manipulating sets of states. In fact, the use of BDDs makes it possible to verify very large systems (larger than 10^{20} states [15,40,14]). Symbolic model checking has been successful in various fields, allowing the discovery of design bugs that were very difficult to highlight with traditional techniques. For instance, in [23] the authors discovered previously undetected and potential errors in the design of the cache coherence protocol described in the IEEE Futurebus+ Standard 896.1.1991, and in [29] the cache coherence protocol of the Scalable Coherent Interface, IEEE Standard 1596-1992 was verified, finding several errors.

A more recent advance in the field originates from the introduction of *bounded model checking* (BMC) [7,6]. The idea is twofold. First, we look for a witness to a property violation that can be presented within a certain bound, say k transitions. Second, we generate a propositional formula that is satisfiable if and only if a witness to the property violation exists. The formula is obtained by unwinding the symbolic description of the transition relation over time. In particular, we use $k+1$ vectors of state variables $x_0, \ldots, x_k$, whose assignments represent the states at the different steps, and k vectors of input variables $i_1, \ldots, i_k$, that represent the inputs at the different transitions:

$$\mathcal{S}_i(x_0) \wedge \mathcal{R}(x_0, i_1, x_1) \wedge \ldots \wedge \mathcal{R}(x_{k-1}, i_k, x_k)$$

Additional constraints are used to limit such assignments to witness the violation of the property, and to impose a cyclic behaviour when required. The solution technique leverages the power of modern SAT solvers [30], which are able to check the satisfiability of formulae with hundreds of thousands of variables, and millions of clauses.

In comparison to BDD-based algorithms, the advantages of SAT-based techniques are twofold [25]. First, SAT-based algorithms have higher capacity, i.e. they can deal with a larger number of variables. Second, SAT solvers have a high degree of automation, and are less sensitive than BDDs to the specific parameters (e.g. variable ordering). As a result, SAT-based technologies have been introduced in industrial settings to complement and often to replace BDD-based techniques. In addition, SAT has become the core of many other algorithms and approaches, such as inductive reasoning (e.g. [50]), incremental bounded model checking (e.g. [36]), and abstraction (e.g. [35]). A survey of the recent developments can be found in [44].

3 Safety Analysis

The goal of safety analysis is to investigate the behavior of a system in degraded conditions, that is, when some parts of the system are not working properly, due to malfunctions. Safety analysis includes a set of activities, that have the goal of identifying all possible hazards of the system, and that are performed in order to ensure that the system meets the safety requirements that are required for its deployment and use. Safety analysis activities are particularly critical in the case of reactive systems, because hazards can be the result of complex interactions involving the dynamics of the system [51]. Traditionally, safety analysis activities have been performed manually. Recently, there has been a growing interest in model-based safety analysis using formal methods [11,9,1,10] and in particular symbolic model checking.

Model-based safety analysis is carried out on formally specified models which take into account system behavior in the presence of malfunctions, that is, possible *faults* of some components. Symbolically, the occurrence of such faults is modeled with a set of additional propositions, called *failure mode variables*. Intuitively, a failure mode variable is true when the corresponding fault has occurred in the system (different failure mode variables are associated to different faults). In the rest of this section, we assume to have a system $\mathcal{M} = \langle \mathcal{S}, \mathcal{S}_i, \mathcal{I}, \mathcal{R} \rangle$ with a set of failure mode variables $\mathcal{F} \subseteq \mathcal{P}$. Furthermore, for the sake of simplicity, we assume that failure modes are *permanent* (*once failed, always failed*), that is, we assume that the following condition holds:

$$\forall f \in \mathcal{F}, s_1, s_2 \in \mathcal{S}, i \in \mathcal{I} \quad (\langle s_1, i, s_2 \rangle \in \mathcal{R} \wedge s_1 \models f) \Rightarrow s_2 \models f \qquad (1)$$

The theory can be extended to the more general case of *sporadic* or *transient* failure modes, that is, when faults are allowed to occur sporadically (e.g., a sensor showing an invalid reading for a limited period of time), possibly repeatedly over time, or when repairing is possible.

In this section we briefly describe two of the most popular safety analysis activities, that is, *fault tree analysis* (FTA) and *failure mode and effect analysis* (FMEA), and we discuss their relationship with the symbolic model checking techniques illustrated in Section 2. Fault Tree Analysis [53] is an example of deductive analysis, which, given the specification of an undesired state, usually referred to as a *top level event*, systematically builds all possible chains of one of more basic faults that contribute to the occurrence of the event. The result of the analysis is a *fault tree*, that is, a representation of the logical interrelationships of the basic events that lead to the undesired state. In its simpler form (see Fig. 3) a fault tree can be represented with a two-layer logical structure, namely a top level disjunction of the combinations of basic faults causing the top level event. Each combination, which is called *cut set*, is in turn the conjunction of the corresponding basic faults. In general, logical structures with multiple layers can be used. A cut set is formally defined via CTL as follows.

Definition 2 (Cut set). *Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{S}_i, \mathcal{I}, \mathcal{R} \rangle$ be a system with a set of failure mode variables $\mathcal{F} \subseteq \mathcal{P}$, let $FC \subseteq \mathcal{F}$ be a fault configuration, and $TLE \in \mathcal{P}$ a top level event. We say that FC is a cut set of TLE, written $cs(FC, TLE)$ if*

$$\mathcal{M} \models EF\Big(\bigwedge_{f \in FC} f \;\wedge\; \bigwedge_{f \in (\mathcal{F} \setminus FC)} \neg f \;\wedge\; TLE\Big).$$

Intuitively, a fault configuration corresponds to the set of active failure mode variables. Typically, among the possible fault configurations, one is interested in isolating those that are minimal in terms of failure mode variables, that is, those that represent simpler explanations, in terms of system faults, for the occurrence of the top level event. Under the hypothesis of independent faults, these configurations also represent the most probable explanations for the top level event, and therefore they have a higher importance in reliability analysis. Minimal configurations are called *minimal cut sets* and are formally defined as follows.

Definition 3 (Minimal Cut Sets). *Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{S}_i, \mathcal{I}, \mathcal{R} \rangle$ be a system with a set of failure mode variables $\mathcal{F} \subseteq \mathcal{P}$, let $F = 2^{\mathcal{F}}$ be the set of all fault configurations, and*

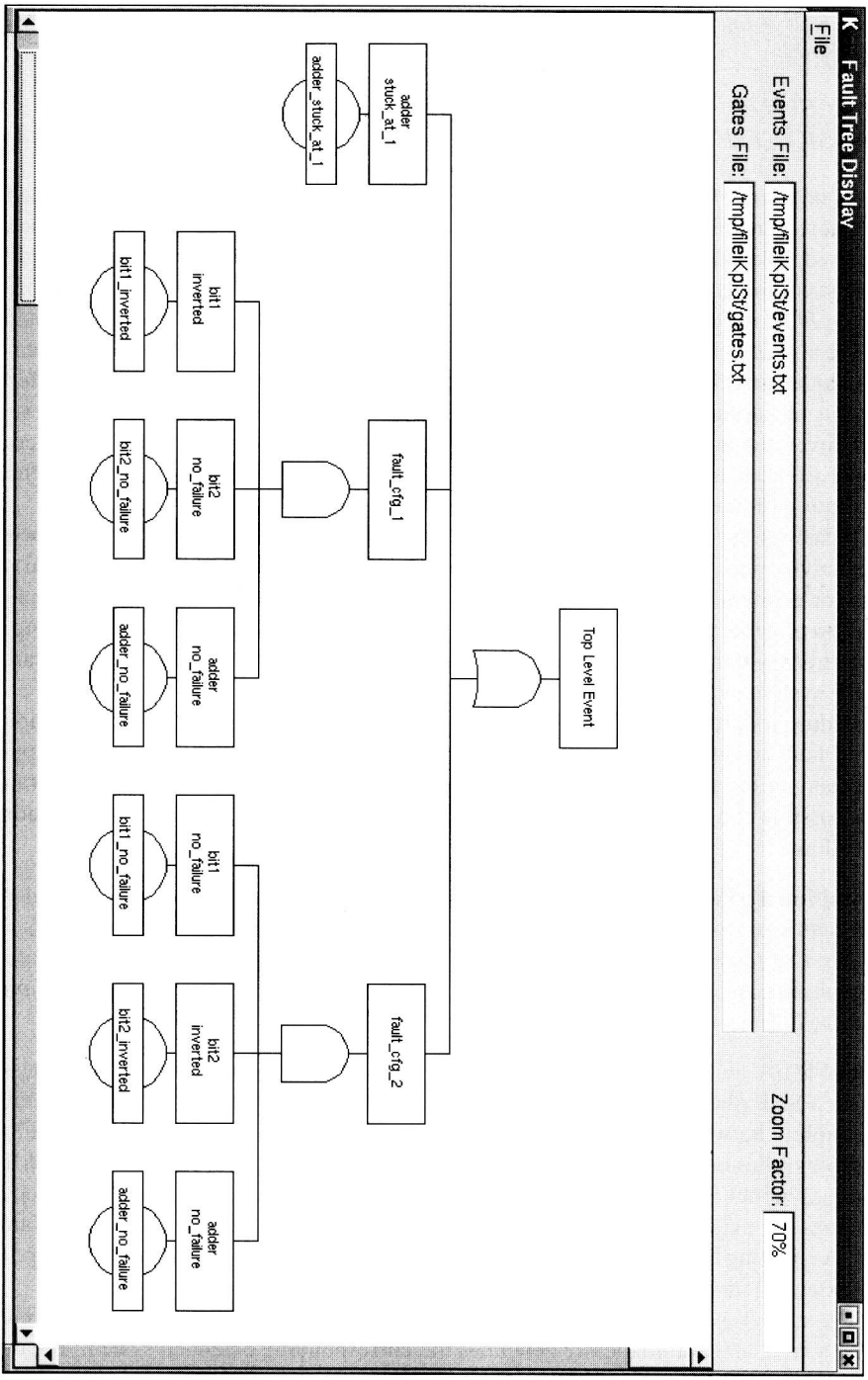

Fig. 3. An example of fault tree