06172 Abstracts Collection
Directed Model Checking
— Dagstuhl Seminar —

Stefan Edelkamp[1], Stefan Leue[2], and W. Visser[3]

[1] Univ. of Dortmund, DE
stefan.edelkamp@cs.uni-dortmund.de
[2] Univ. of Konstanz, DE
Stefan.leue@uni-konstanz.de
[3] NASA, USA
wvisser@ptolemy.arc.nasa.gov

Abstract. From 26.04.06 to 29.04.06, the Dagstuhl Seminar 06172 "Directed Model Checking" was held in the International Conference and Research Center (IBFI), Schloss Dagstuhl. During the seminar, several participants presented their current research, and ongoing work and open problems were discussed. Abstracts of the presentations given during the seminar as well as abstracts of seminar results and ideas are put together in this paper. The first section describes the seminar topics and goals in general. Links to extended abstracts or full papers are provided, if available.

Keywords. Model Checking, Artificial Intelligence, AI Plannning, Guided Traversal, State Explosion Problem

06172 Executive Summary – Directed Model Checking

This is a summary of the Dagstuhl Seminar 06172 *Directed Model Checking* that was held 26 - 29 April 2006 at Schloss Dagstuhl, Germany. Directed Model Checking is a software and hardware verification technique that performs a systematic, heuristics guided search of the state space of the model to be analyzed. It hence reconciles classical model checking technology with intelligent, heuristics driven search that has a long tradition in artificial intelligence, in particular in the area of action planning. The benefits are short or even optimally short error trails, in some instances a more efficient exploration of the state space, and the applicability of state space search in some application areas in which unintelligent search would not yield useful results.

The seminar brought together researchers from the system verification and the artificial intelligence domain in order to discuss the current state of the art, and to elicit and discuss research challenges and future directions.

Keywords: Model checking, heuristics, state space search, software and hardware verification

Dagstuhl Seminar Proceedings 06172
Directed Model Checking
http://drops.dagstuhl.de/opus/volltexte/2007/946

Joint work of: Edelkamp, Stefan; Leue, Stefan; Visser, Willem

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/944

Directed Search Algorithms for Probabilistic Timed Reachability

Husain Aljazzar (Universität Konstanz, D)

The inability to provide counterexamples for the violation of probabilistic timed reachability properties constrains the practical use of stochastic Model Checking, e.g. CSL Model Checking for continuous time Markov chains (CTMCs). Counterexamples are essential tools in determining the causes of property violations and are required during debugging. For a given probabilisitc timed property, we propose the use of explicit state model checking to determine a failure trace, i.e. a run leading into some property offending state.

Since we are interested in finding counterexamples that carry large amounts of probability mass we employ directed explicit state model checking technology to find such failure traces using a variety of heuristics guided search algorithms, such as Best First search and Z*.

The estimates used in computing the heuristics rely on a uniformisation of the CTMC. We apply this approach to a probabilistic model of the SCSI-2 protocol.

In the context of stochastic Model Checking, one failure trace is, in general, not enough to show the violation of a given property. Also for debugging, it makes more sense to look at a reasonable set of failure traces.

Thus, we extend the search algorithms so that a subgraph of the state transition graph is selected which can be used to show the violation of the given property.

Hence, This subgraph can be considered as a real counterexample.

We applied the extended approach to a probabilistic model for workstation cluster for parallel computing.

Keywords: Directed Model Checking, Stochastic Model Checking, Counterexample, Debugging, Heuristic Search Algorithms

Full Paper:
http://www.inf.uni-konstanz.de/soft/research/publications/pdf/formats05

See also: Husain Aljazzar, Holger Hermanns, and Stefan Leue: Counterexamples for Timed Probabilistic Reachability, Proceedings of the 3rd International Conference on Formal Modelling and Analysis of Timed Systems FORMATS'05, LNCS 3829, Lecture Notes in Computer Science, Springer Verlag, 2005.

A Hybrid of Counterexample-based and Proof-based Abstraction

Nina Amla (Cadence - Sunnyvale, USA)

Counterexample- and proof-based refinement are complementary approaches to iterative abstraction. In the former case, a single abstract counterexample is eliminated by each refinement step, while in the latter case, all counterexamples of a given length are eliminated. In counterexample-based abstraction, the concretization and refinement problems are relatively easy, but the number of iterations tends to be large. Proof-based abstraction, on the other hand, puts a greater burden on the refinement step, which can then become the performance bottleneck. In this talk, we show that counterexample- and proof-based refinement are extremes of a continuum, and propose a hybrid approach that balances the cost and quality of refinement. In a study of a large number of industrial verification problems, we find that there is a strong relation between the effort applied in the refinement phase and the number of refinement iterations. For this reason, proof-based abstraction is substantially more efficient than counterexample-based abstraction.

However, a judicious application of the hybrid approach can lessen the refinement effort without unduly increasing the number of iterations, yielding a method that is somewhat more robust overall.

Joint work of: Amla, Nina; McMillan, Ken

Model Checking Networked Applications

Cyrille Artho (National Institute of Informatics - Tokyo, J)

Software model checkers can be applied directly to single-process programs, which typically are multi-threaded. On the other hand, multiple processes can usually not be model checked directly. Several approaches to deal with this problem are possible: Network operations may be replaced with (application-specific) stubs, a special model-checking-aware cache layer may be inserted between the model checker and the system, or processes may be converted into threads.

This talk covers these possibilities, emphasizing the last solution called "process centralization". Previous work has not covered all issues regarding centralizing, most importantly, TCP/IP communication.

Keywords: Software model checking, network, inter-process communication

Do (not) Know Much About the History, Biology (of Directed Model Checking)

Dragan Bosnacki (TU Eindhoven, NL)

In the first part of this presentation I am going to talk about some early forms of Directed Model Checking which were used in Approver. Approver is probably the first tool for automated verification of communication protocols. It was written by Jan Hajek in the end of the 70's at the Eindhoven University of Technology. Besides being interesting from a historical perspective, I hope that the ideas for fast bug finding that were used in Approver can be further developed in the context of Directed Model Checking.

The second part of the talk is devoted to some applications of model checking for the analysis of biological networks and in particular the possible role of directed model checking in such applications.

Finally, I will breafly go over some topics of combining directed model checking with other techniques that I would like to discuss off line during the seminar.

Keywords: Directed model checking, history of model checking, bioinformatics

Efficient Software Model Checking of Data Structure Properties

Chandrasekhar Boyapati (Univ. of Michigan - Ann Arbor, USA)

This talk presents novel language and analysis techniques that significantly speed up software model checking of data structure properties. Consider checking a red-black tree implementation. Traditional software model checkers systematically generate all red-black tree state(within some given bounds) and check every red-black tree operation (such as insert, delete, or lookup) on every red-black tree state. Our key idea is as follows. As our checker checks a red-black tree operation o on a red black tree state s, it uses program analysis techniques to identify other red-black tree states s'_1, s'_2, ..., s'_k on which the operation o behaves similarly. Our analyses guarantee that if o executes correctly on s, then o will execute correctly on every s'_i. Our checker therefore does not need to check o on any s'_i once it checks o on s. It thus safely prunes those state transitions from its search space, while still achieving complete test coverage within the bounded domain. Our preliminary results show *orders of magnitude improvement* over previous approaches. We believe our techniques can make software model checking significantly faster, and thus enable checking of much larger programs and complex program properties than currently possible.

Keywords: Software Model Checking, Program Analysis, Linked Data Structures

Joint work of: Boyapati, Chandrasekhar; Darga, Paul

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/945

Directed Model Checking with Distance-Preserving Abstractions

Klaus Dräger (Universität des Saarlandes, D)

Recent work on pattern databases has shown that abstraction-based heuristics are useful for directed model checking. The central research question is how to compute a useful abstraction without hitting the state space explosion problem.

We present a method that automatically computes an abstraction for a given system such that the distance between each concrete state and the closest error state in the concrete system is closely approximated by the corresponding distance in the abstraction. The cost of computing the abstraction (and the resulting informedness of the heuristic) can be adjusted through a control parameter.

Our experiments using these abstractions show a dramatic reduction both in the number of states explored by the model checker and in the total runtime.

Full Paper:
http://springerlink.com/link.asp?id=1m63183j35474876

See also: K.Dräger and B.Finkbeiner and A. Podelski, Directed Model Checking with Distance-Preserving Abstractions. In: Proceedings of the 13th International SPIN Workshop on Model Checking of Software, Lecture Notes in Computer Science 3925, pages 19-34. Springer, 2006

Controlling Factors in Evaluating Directed Model Checking Techniques

Matthew Dwyer (University of Nebraska, USA)

Recent advances in directed model checking have made it possible to detect errors in applications that have been thoroughly tested and are in wide-spread use.

The ability to find errors that have eluded traditional validation methods is due to the development and combination of sophisticated algorithmic techniques that are embedded in the implementations of analysis tools.

Evaluating new analysis techniques is typically performed by running an analysis tool on a collection of subject programs, perhaps enabling and disabling a given technique in different runs.

While seemingly sensible, this approach runs the risk of attributing improvements in the cost-effectiveness of the analysis to the technique under consideration, when those improvements may actually be due to details of analysis tool implementations that are uncontrolled during evaluation.

In this talk, we report on the results of empirical studies that we have performed that illustrate the existence of several factors that can significantly influence the cost of directed model checking. You may not find the nature of these factors too surprising, but we believe the data on the degree of influence of those factors on analysis cost is significant enough that it will cause you to want to “think twice” when designing evaluations of future techniques.

We conclude with several recommendations as to how the influence of these factors can be mitigated when evaluating techniques.

Joint work of: Dwyer, Matthew; Person, Suzette; Elbaum, Sebastian

Directed Model Checking: Search Algorithms and Heuristics

Stefan Edelkamp (Universität Dortmund, D)

In this talk we introduce to state-of-the-art heuristic search algorithms and heuristics for model checking as a way to mitigate the state explosion problem in implicit exploration

In the first part we address frontier search, breadth-first heuristic search, and variants of the SetA* algorithm. As acceleration techniques we study partial search, incremental hashing, collapse compression and state reconstruction. Furthermore, study symbolic exploration as well as external and parallel execution.

In the second part on estimates we distinguish between explanatory and trail-directed heuristics. In particular, we introduce to pattern databases and FSM heuristic as well to property-specific (e.g. deadlock), formula-based and tool-inherent (e.g. planning) heuristics,

Keywords: Heuristic Search, Model Checking Planning

See also: Stefan Edelkamp, Stefan Schroedl, Sven Koenig; Heurstic Search: Theory and Practice, Morgan Kaufmann, To Appear

Abstraction Refinement of Hybrid Systems

Ansgar Fehnker (National ICT Australia Ltd. - Sydney, AU)

In this talk I will present the basic ideas of counterexample guided abstraction refinement for hybrid systems, using set of fragments. It wil focus in particular on how to select a suitable set of fragments that will be checked, explored, and if necessary refined in the current iteration for the current abstraction.

Keywords: Abstraction refinement, hybrid systems, cut sets

Potential Applications of Directed Model Checking in Testing

Wolfgang Grieskamp (Microsoft Research, USA)

The talk gives an overview over the vitual execution environment (or software model-checker) XRT developed at Microsoft Research. It then digs into applications of XRT in testing, and how directed search is used today and may be used in the future for improvements.

Keywords: Software-model checking, model-based testing, unit testing

External Memory Directed Model Checking

Shahid Jabbar (Universität Dortmund, D)

Practical model checking is still hurdled by the amount of internal memory available. Virtual memory, might seem useful at a first glance, can in fact slows down the performance of the algorthm due to accesssive page faults.

We will discuss algorithms that utilize hard-disk during model checking.

Starting from a simple external breadth-first search, we will develop External Memory algorithms for "Directed model checking". Both safety and liveness checking for LTL properties will be discussed. The performance of external memory algorithms is measured in the number of Input/Output operations performed on the hard-disk. We will evaluate our algorithms on this new model and prove their optimality and correctness.

External memory algorithms have a large potential to evolve into distributed algorithms. One such extension will be presented.

Related Literature:

=>Stefan Edelkamp, Shahid Jabbar, and Stefan Schroedl, External A*.

In KI 2004: Advances in Artificial Intelligence (German Conference on AI) by Biundo et. al (eds.). Lecture Notes in Artificial Intelligence (LNAI), volume 3238, pages 226 - 240, Springer-Verlag, Ulm, Germany, 2004.

=>Shahid Jabbar, Stefan Edelkamp, I/O Efficient Directed Model Checking.

In Verification, Model Checking and Abstract Interpretation (VMCAI'05) by Cousot(Ed.). Lecture Notes in Computer Science (LNCS), volume 3385, Springer-Verlag, pages 313-329, Paris, France, 2005.

=>Shahid Jabbar and Stefan Edelkamp, Parallel External Directed Model Checking With Linear I/O. In Verification, Model Checking and Abstract Interpretation (VMCAI'06) by E. A. Emerson and K. S. Namjoshi (Eds.).

Lecture Notes in Computer Science (LNCS), volume 3855, Springer-Verlag, pages 237-251, Charleston, South Carolina, USA, Jan. 2006.

=>Stefan Edelkamp and Shahid Jabbar, Large Scale External Directed Liveness Checking. In 13th International SPIN Workshop on Model Checking of Software (SPIN 2006) by Valmari (Ed.). Lecture Notes in Computer Science (LNCS), volume 3925, pages 1-18, Springer, Vienna, Austria, Mar. 2006.

Challenges and Applications of Assembly-Level Software Model Checking

Tilman Mehler (Universität Dortmund, D)

Recent approaches in software model checking rely on investigating the actual source code of the a program rather than a formal model. There are several challenges that need to be overcome to build such a model checker. First, the tool must be capable to handle the full semantics of the underlying programming language. This implies a considerable amount of additional work unless the interpretation of the program is done by some existing infrastructure. The second challenge lies in the increased memory requirements needed to memorize entire program configurations. This additionally aggravates the problem of large state spaces that every model checker faces anyway.

In this talk, we present the experimental C++ model checker StEAM, developed in the course of the authors PhD thesis. Unlike tools such as the the second generation of the Java model checker JPF, StEAM does not implement its own virtual machine, but rather ties the model checking algorithm to an existing virtual machine. To address the problem of large program states, we call attention to the fact that most transitions in a program only change small fractions of the entire program state. Based on this observation, we devise an incremental storing of states which considerably lowers the memory requirements of program exploration. To further alleviate the per-state memory requirement, we apply state reconstruction, where states are no longer memorized explicitly but through their generating path. Another problem that results from the large state description of a program lies in the computational effort of hashing, which is exceptionally high for the used approach.

Based on the same observation as used for the incremental storing of states, we propose an incremental hash function which only needs to process the changed parts of the program's state.

Keywords: Software model checking, memory reduction, heuristics, hashing

Using the runtime stack and static calling context to improve distance heuristics in directed model checking

Eric Mercer (Brigham Young Univ. - Provo, USA)

State exploration in directed software model checking is guided using a heuristic function to move states near errors to the front of the search queue. Distance heuristic functions rank states based on the number of transitions needed to move the current program state into an error location. Inlining functions at call-sites in the control flow graph to capture calling context in the distance heuristic function leads to exponential growth in the computation. We present a new algorithm that implicitly inlines functions at call sites to compute distance data with unbounded

calling context that is polynomial in the number of nodes in the control flow graph. The new algorithm propagates distance data through call sites during a depth-first traversal of the program. We show in a series of benchmark examples that the new heuristic function with unbounded distance data is more efficient than the same heuristic function using data from an bounded control full graph that inlines functions at their call sites.

Keywords: Directed model checking distance heuristic CFG FSM runtime stack

Joint work of: Rungta, Neha; Mercer, Eric

Guided Simulation of Autonomous Controllers

Charles Pecheur (Univ. cath. de Louvain, B)

AI software is often used as a means for providing greater autonomy to automated systems, capable of coping with harsh and unpredictable environments. Due in part to the enormous space of possible situations that they aim to address, autonomous systems pose a serious challenge to traditional test-based verification approaches. We describe an approach to the verification of autonomous control software based on guided execution of the actual program. This approach is the basis of the Livingstone PathFinder (LPF) tool. LPF applies state space exploration algorithms to an instrumented testbed, consisting of the controller embedded in a simulated operating environment. Although LPF has focused on NASA's Livingstone model-based diagnosis system applications, the architecture is modular and adaptable to other systems. We present different facets of LPF and experimental results from applying the software to a Livingstone model of the main propulsion feed subsystem for a prototype space vehicle.

Keywords: Simulation, verification, autonomy, diagnosis, guided search

Joint work of: Pecheur, Charles; Lindsey, Tony

Full Paper:
http://www.info.ucl.ac.be/∼pecheur/publi/lpf-tacas04.pdf

See also: Tony Lindsey and Charles Pecheur. Simulation-Based Verification of Autonomous Controllers with Livingstone PathFinder. In: TACAS'04, Lecture Notes in Computer Science, vol. 2988, Springer Verlag, 2004.

Branch and Bound with SPIN

Theo Ruys (University of Twente, NL)

The use of model checkers to solve discrete optimization problems is appealing. A model checker can first be used to verify that the model of the problem is correct.

Subsequently, the same model can be used to find an optimal solution for the problem.

In this talk we show how to use the SPIN model checker to search for the optimal solution using SPIN's default depth first search (DFS) order.

We use Promela's embedded C features – which were introduced in SPIN version 4 – to elegantly include search variables into the Promela model in order to improve the search. We show how Branch-and-Bound techniques can be added to the LTL property that is used to find the solution. This LTL property is then dynamically changed during the verification. We also show how the syntactical reordering of statements and/or processes in the Promela model can improve the efficiency of the search even further.

The talk is partly based on the SPIN 2003 paper by the author.

Keywords: SPIN, Promela, optimization problem, branch and bound

Liveness Checking as Safety Checking to Find Shortest Counterexamples

Viktor Schuppan (München, D)

While verification of ω-regular properties requires detection of fair repeated reachability, simple reachability is sufficient to check safety properties. We present an efficient translation from checking fair repeated reachability to reachability for finite state systems. As reachability is amenable to BFS, we obtain a practical method to find shortest fair lasso-shaped paths. Whether a shortest fair lasso in the product of a model and an automaton representing the specification indeed represents a shortest counterexample in the model depends on the translation of the specification into a Büchi automaton. We discuss requirements on Büchi automata that ensure that this is the case. Experimental results indicate that the difference in counterexample length that is due to using different algorithms to find a fair cycle is larger than the difference that stems from using different Büchi automata to encode the specification.

Keywords: Model checking, liveness, safety, Büchi automata, shortest counterexamples

Joint work of: Schuppan, Viktor; Biere, Armin

Heuristics for Model Checking CCS processes

Maria Luisa Villani (Univ. of Sannio - Benevento, I)

Model Checking suffers from the state explosion problem due to the exponential increase in the size of a finite state model as the number of system components grows.

Directed model checking can be a solution to this problem if suitable heuristics are used to avoid the construction of the complete system's model in order to deduce the satisfaction or not of the property at hand.

In this line, we propose structure-based heuristic functions operating on processes described in the Calculus of Communicating Systems (CCS) that can be used with informed search strategies like A*, IDA* and Greedy. In particular, we defined admissible and non-admissible heuristics specifically for deadlock detection and one admissible heuristic for the verification of any other formula expressed in the Selective Hennessy-Milner logic.

The results of some experiments we have conducted to evaluate the method will be shown.

Keywords: Heuristics, CCS

Joint work of: Gradara, Sara; Santone, Antonella; Villani, Maria Luisa

Solving Scheduling Problems by Untimed Model Checking

Anton Wijs (CWI - Amsterdam, NL)

We show how scheduling problems can be modelled in untimed process algebra, by using special tick actions. A minimal-time trace leading to a particular action, is one that minimizes the number of tick steps. As a result, we can use any (timed or untimed) model checking tool to find shortest schedules. Instantiating this scheme to μCRL, we profit from a richer specification language than timed model checkers usually offer. Also, we can profit from efficient distributed state space generators. We propose a variant of breadth-first search that visits all states between consecutive tick steps, before moving to the next time slice. We experimented with a sequential and a distributed implementation of this algorithm. We also experimented with beam search, which visits only parts of the search space, to find near-optimal solutions. Our approach is applied to find optimal schedules for test batches of a realistic clinical chemical analyser, which performs several kinds of tests on patient samples.

Keywords: Process algebra, scheduling, search algorithms, untimed model checking

Joint work of: Wijs, Anton; Pol, Jaco van de; Bortnik, Elena

See also: A.J. Wijs, J.C. van de Pol, and E. Bortnik. Solving Scheduling Problems by Untimed Model Checking, The Clinical Chemical Analyser Case Study. In Proc. 10th International Workshop on Formal Methods for Industrial Critical Systems (FMICS'05), pages 54-61. ACM Press, 2005