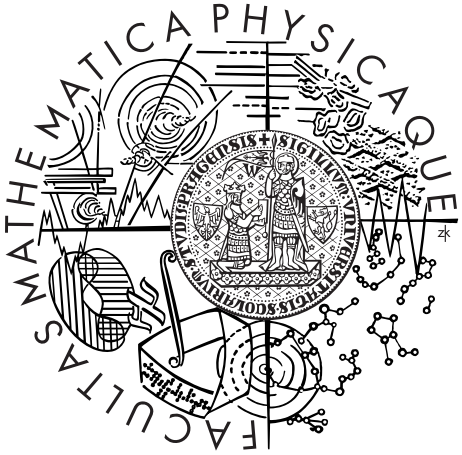
Charles University in Prague
Faculty of Mathematics and Physics

DOCTORAL THESIS ABSTRACT



Martin Suda

Resolution-based methods for linear temporal reasoning

Department of Theoretical Computer Science and
Mathematical Logic

Supervisor:	Prof. RNDr. Roman Barták, Ph.D.
Co-supervised by:	Prof. Dr. Christoph Weidenbach Max Planck Institute für Informatik, Saarbrücken, Germany
Study programme:	Computer Science
Specialization:	Theoretical computer science (4I-1)

Prague 2015

Disertační práce byla vypracována v rámci doktorského studia pod dvojím vedením (*cotutelle*) na Matematicko-fyzikální fakultě Univerzity Karlovy v Praze a na Fakultě přírodních věd a technoligie Saarské Univerzity v Saarbrückenu v Německu v letech 2007-2015.

Uchazeč	Martin Suda suda@ktiml.mff.cuni.cz
Školitelé	Prof. RNDr. Roman Barták, Ph.D. Katedra teoretické informatiky a matematické logiky Matematicko-fyzikální fakulta Univerzita Karlova v Praze Malostranské náměstí 25, 118 00 Praha 1 bartak@ktiml.mff.cuni.cz
	Prof. Dr. Christoph Weidenbach Max-Planck-Institut für Informatik Campus E1 4, 66123 Saarbrücken, Německo weidenbach@mpi-inf.mpg.de
Oponenti	Prof. Dr. Armin Biere Johannes Kepler Universität Linz Altenbergerstraße 69, 4040 Linz, Rakousko biere@jku.at
	Prof. Dr. Jörg Hoffmann Universität des Saarlandes Campus E1 1, 66123 Saarbrücken, Německo hoffmann@cs.uni-saarland.de
Předseda RDSO 4I-1	Prof. RNDr. Václav Koubek, DrSc. Katedra teoretické informatiky a matematické logiky Matematicko-fyzikální fakulta Univerzita Karlova v Praze Malostranské náměstí 25, 118 00 Praha 1 koubek@ktiml.mff.cuni.cz

V souladu se smlouvou o dvojím vedení doktorské práce proběhla obhajoba dne 16.10.2015 v 10:00 hodin na Saarské Univerzitě v Saarbrückenu v Německu.

Contents

1. Introduction

The presented thesis explores the potential of resolution-based reasoning methods for automatically establishing properties of systems which evolve in time. Following the *automation of logic* tradition, its main aim is to develop and analyze new algorithms for automated reasoning and to establish experimentally whether they can be successfully applied in practice. Relevant applications for the developed algorithms are found successively in three independent but closely related fields: theorem proving in linear temporal logic, verification of hardware circuits, and automated planning. The corresponding reasoning tasks share a common notion of time, which is modeled as a discrete linear sequence of time moments. They also share a formalism for representing the individual moments, namely propositional logic. It is the second property which enables to approach the tasks using resolution, a well understood rule of logical inference suitable for automation. It is the temporal aspect of the mentioned tasks, on the other hand, which poses the main challenge.

In this abstract, we first provide an overview of the three mentioned fields: theorem proving in linear temporal logic (1.1), verification of hardware circuits (1.2) and automated planning (1.3) and of the corresponding reasoning tasks. We then review the resolution technology used in the thesis to approach these tasks (1.4) and elaborate on the inherent challenges connected with reasoning about time and on the adopted strategy to overcoming them (1.5). Then we provide a summary of the main contributions of the thesis (2).

1.1 Theorem proving in linear temporal logic

Linear-time temporal logic (LTL) is a modal logic designed for specifying temporal relations between events which occur over time. Originally conceived by Kamp [18] to formally capture temporal relationships expressible in natural language, LTL was introduced into computer science by Pnueli [29] as a specification language for reactive systems, i.e. systems with non-terminating computations. Nowadays, LTL is well established and widely used in practice.

LTL extends classical propositional logic by introducing temporal operators which specify the way in which a formula φ should be interpreted with respect to the flow of time. For example, the formula $\Diamond\varphi$ stands for "φ holds *eventually* in the future", $\Box\varphi$ means "φ holds *always* in the future", and $\bigcirc\varphi$ expresses that "φ holds at the *next* moment of time". Time is considered to be a linear discrete sequence of time moments represented by propositional valuations, informally also called *worlds*. Such a potentially infinite sequence forms an LTL interpretation. To prove an LTL formula φ means to establish that it is valid, i.e., that all LTL interpretations actually make φ true. For example, the LTL formula $\Box\psi \to \bigcirc\psi$ is valid (a theorem), whereas the LTL formula $\bigcirc\psi \to \Box\psi$ is not, but is satisfiable, i.e., there is an LTL interpretation in which it is true. Similarly to classical logic, one can prove an LTL formula φ by showing that it does not have a counterexample, i.e. that its negation $\neg\varphi$ is not satisfiable. By this duality, LTL proving and satisfiability checking are essentially two sides of the same coin.

The traditional use of LTL lies in formal verification of reactive systems where

the logic serves as a specification language for expressing the system's desired behavior. Such a specification is subsequently checked against a model of the system during the process of model checking [8]. More recently, the importance of LTL satisfiability checking and thus also theorem proving is becoming recognized [31, 33]. This is, for instance, essential for assuring quality of formal specifications [27]. In more detail, because specifications are ultimately written by humans they may contain bugs. A priori excluding specifications which are either valid or unsatisfiable represents a useful sanity check, because in the former case the specification would be trivially satisfied by any system and in the latter by none. Another motivation comes from the fact that satisfiability is a precondition for realizability and thus a satisfiability checking procedure can be useful in debugging specifications for system synthesis [17].

The presented thesis approaches the problem of LTL theorem proving by building on the work of Fisher [11] who showed how to transform any LTL formula into a certain clausal normal form. The normal form reduces the formula complexity in terms of temporal operator nestings and so removes one of the obstacles on the way to applying resolution-based methods for automated reasoning in LTL.

1.2 Verification of hardware circuits

With the growing reliance on hardware technology in our lives, ranging from the use of personal computers and mobile phones to areas such as traffic control and medicine, where human life is directly affected, the importance of the correct behavior of the used devices is becoming more than obvious. Formal verification provides methods to rigorously establish that a designed circuit meets its specification, thus greatly helping to increase our confidence in the correctness of the final device.

In this work, we focus on a particular phase in the design process in which the designed circuit obtains a representation referred to as gate-level netlist, an abstraction not unrelated to that of the Boolean circuit from theoretical computer science. Moreover, we will be interested in verifying *sequential* circuits, which in addition to logic gates for computing Boolean functions also employ state-holding elements called *latches*. Sequential circuits compute in cycles and in each cycle externally supplied inputs together with the old values stored in the latches participate in producing the circuit's output and the new values to be stored.[1] Simply put, sequential circuits have memory.

Let us have a look at Figure 1.1 for a small example. The figure depicts a simple circuit storing a single bit of memory (latch l). If during a particular cycle the value of the input bit i is zero, the stored value is preserved ($l' = l$). If the value of the input is one, the stored value is read (output $o = l$) and at the same time toggled $l' = \neg l$. This follows from the inner working of the two employed gates, the AND gate computing logical conjunction ($\wedge$) and the XOR gate computing the exclusive or operation ($\oplus$). The semantics of a latch is to

[1] More precisely, this describes the behavior of a sequential circuit of a *synchronous* type. There are also asynchronous circuits in which the state can change at any time in response to changing inputs.

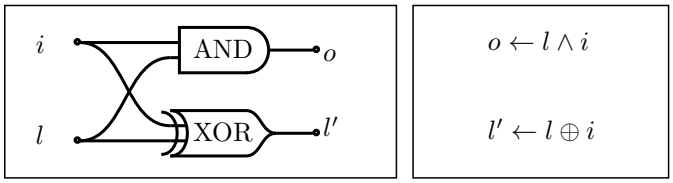


Figure 1.1: A simple circuit (left) and its imperative interpretation (right).

take the value l' computed at the end of one cycle and pass it back as l for the computation of the following cycle. This defines the discrete flow of time in the context of the circuit's computation.

One verifies a sequential circuit with respect to a property expressing its correct behavior. In the most common setting called the verification of invariance properties, which we will consider in this work, the property is a propositional formula φ over variables corresponding to the latches. Formula φ specifies an *invariant* of the circuit which must be satisfied in all the states reachable from a fixed initial state. Dually, the invariant is violated if there is a computation of the circuit starting from the initial state, processing in cycles a particular sequence of inputs and ending in a state where φ does not hold. Such a state is usually called a *bad* state.

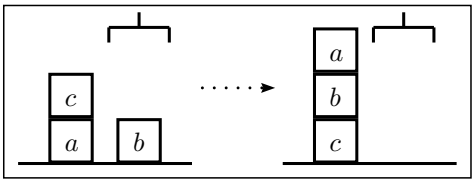
Sequential verification is a computationally hard problem. An intuition about this can be drawn by realizing that the size of the circuit's state space is exponential in the number of its latches. Moreover, exponentially many states may need to be traversed on the potential path from an initial state to a bad state. One way to combat this well known *state explosion problem* is to avoid explicit enumeration of the states and instead use and manipulate symbolic representation of whole sets of states. This approach was pioneered by McMillan [24] who used Binary Decision Diagrams as such a representation.

By adapting the resolution-based methods to the problem of verification of invariance properties, the theses naturally arrives to a symbolic algorithm where the representation of sets of states is based on clausal propositional logic.

1.3 Automated planning

Automated planning is a classical discipline of artificial intelligence [32, 12]. Operating with a given formal, high-level description of a world, its fundamental task is to look for a sequence of actions that lead to achieving a specified goal. Depending on the modeled scenario, the goal-achieving sequence, or simply the plan, may describe anything from a route for a space rover collecting samples on a remote planet, instructions for the preparation of a complex chemical from individual simpler compounds, to a set of trajectories for conveyors loading crates in a warehouse. This generality is enabled by the expressiveness of the language used for describing the input task, which is traditionally based on first-order logic.

Figure 1.2 depicts an example situation from the emblematic *blocks world* planning domain. The task in the domain is to plan the instructions of a robotic arm to rearrange stacks of blocks. The domain description consists of a set of predicates for defining the state of the world (here, $on(X, Y)$, $clear(X)$, etc.) and a set of operators defining the possible transformations (here, $unstack(X, Y)$,



Operator $unstack(X, Y)$
pre : $clear(X)$, $on(X, Y)$, $arm\text{-}empty$
add : $holding(X)$, $clear(Y)$
del : $clear(X)$, $on(X, Y)$, $arm\text{-}empty$

Figure 1.2: The initial and goal configurations of simple planning scenario (left) and an example operator (right).

$stack(X, Y)$, ...). An instance of a planning task in the domain is then specified by supplying a set of concrete objects (in our example, we have blocks a, b and c) and their initial and goal configurations (for instance, the facts $on(c, a)$, $clear(c)$, and *arm-empty* are part of the definition of the initial configuration, whereas $on(a, b)$ and $on(b, c)$ would be part of the goal).

Operators are parametrized schemas for describing concrete actions. That means that an action is obtained by substituting concrete objects for the operator's parameters. In our example, we for instance obtain the action $unstack(c, a)$ by substituting c for X and a for Y in operator $unstack(X, Y)$. This action is *applicable* in the initial configuration, because all its three preconditions (pre): $clear(c)$, $on(c, a)$ and *arm-empty* are satisfied there. When this action is applied, we arrive to a new configuration by adding (add) facts that are supposed to additionally hold (here $holding(c)$ and $clear(a)$) and by deleting (del) facts that now cease to hold (here the same three that were required as preconditions). Such a transition from one configuration to the next via action application is what defines a single time step within the planning semantics.

The blocks world domain is a very simple one and a specialized algorithm for efficiently solving tasks in this domain can easily be devised. The key asset of automated planning, however, lies in that it is *domain independent*. It aims to provide methods for uniformly solving tasks in any domain that can be described by its language. That is what makes automated planning versatile, but also challenging. Another interesting perspective on domain independence is to view the planning formalism as a high-level declarative programming language decoupling the problem from its solution [14].

Thanks to the connection of the standard planning formalism to first-order logic, some of the historically first approaches to automated planning formulated the task as first-order theorem proving [13, 23]. The later discovery of Kautz and Selman [19], who proposed to restate the same problem in terms of *propositional satisfiability*, founded a new powerful approach to planning, which is still actively studied today. This planning as satisfiability paradigm shows how to encode a planning task into "propositional logic plus linear time" and provides the necessary bridge to planning for the resolution-based methods developed in the presented thesis.

1.4 Resolution-based reasoning

The presented thesis makes use of resolution in two main forms. Explicitly, as an inference rule in the context of saturation-based theorem proving, and implicitly, as a proof system underlying the computation of most of the modern propositional

satisfiability (SAT) solvers. These two forms are, in fact, closely related.

Resolution and saturation

The history of resolution as a logical rule is at least as old as the history of the automated theorem proving research field. One of the first appearances of resolution can be found in the work of Davis and Putnam [10], who used the principle within their theorem proving procedure for eliminating ground atomic formulas. Subsequently, Robinson [30] showed how to use unification to lift resolution from ground formulas to formulas with variables and established saturation-based theorem proving with resolution as the main inference rule as one of the most successful approaches to automatically proving theorems in first-order logic.

Resolution operates on a formula in clause normal form which consists of a conjunctive set of clauses, each clause being a disjunction of literals. From two clauses $C \vee a$ and $D \vee \neg a$ containing complementary literals a and $\neg a$, respectively, the resolution inference rule (or, more precisely, a propositional version thereof) allows one to derive a resolvent $C \vee D$. Such an inference is typically summarized as

$$\mathcal{I}\frac{C \vee a \quad D \vee \neg a}{C \vee D}.$$

Resolution forms the basis of a refutationally complete calculus, which means that from any unsatisfiable set of clauses one can derive an obvious contradiction in the form of the empty clause by a finite number of resolution inferences.[2]

Saturation is a process of performing all available inferences in a systematic way until the empty clause is derived, or a set closed under the inferences and not containing the empty clause is obtained which signifies that the original formula was satisfiable. Because saturation typically produces a large number of clauses, techniques for controlling and restricting the process are of great practical importance.

One of the most important achievements in this area was the development of the *superposition* calculus [1, 2] and an associated set of saturation strategies which 1) allow a restriction of considered inferences to only those satisfying certain ordering constraints on the participating literals, 2) introduce a powerful notion of abstract redundancy to justify active removal of clauses that provably cannot contribute to the derivation of the empty clause, 3) still guarantee overall completeness. While the ideas of superposition were originally conceived in the context of first-order theorem proving with equality, they can be restated and successfully applied already on the level of propositional logic [3].

Resolution and SAT solving

By revising the already mentioned procedure of Davis and Putnam [10] and replacing the explicit application of resolution by a splitting rule for case analysis, Davis at al. [9] introduced an algorithm nowadays known as the DPLL procedure and started an extremely successful field of practical propositional satisfiability (SAT) checking.

[2]Combined with the explicit or implicit use of the factoring rule, which takes care of contracting multiple occurrences of the same literal in a clause to just a single occurrence.

DPLL is best described as a backtrack search in the space of partial truth assignments, supported by a *unit propagation* rule for early detection of conflicts. The power of modern SAT solvers still originates from DPLL, but is extended further by the use of efficient data structures and clever branching heuristics [26], non-chronological backtracking with *conflict driven clause learning* [22] and many other techniques [6].

Although procedurally DPLL and its extensions are very different from saturation, the corresponding procedures can still be seen to implicitly generate a resolution proof of the input formula. Results of this form are not always easy to derive, but are important, for instance, for establishing the proof theoretic strength of the individual extensions [4, 28].

The presented thesis exploits a particularly fine-grained connection [40] between CDCL, a version of DPLL enhanced with the conflict driven clause learning technique, and the superposition framework [3]. The connection, in particular, relates unit propagation and clause learning to the concept of a *productive clause* from the completeness proof of superposition. It enables a transformation of a proof method first developed in the saturation setting into an efficient SAT-based LTL prover.

1.5 The temporal challenge

Although resolution is a well established method for automated reasoning in general, the temporal dimension of the problems studied in the presented thesis represents an extra challenge. On an abstract level, this challenge manifests itself in at least two forms depending on whether we primarily focus on satisfiability or unsatisfiability detection. From the perspective of satisfiability detection, we deal with the challenge of large models. As mentioned before, a path from an initial state to a bad state of a circuit can be exponential in the size of the task description and a similar observation holds for planning tasks. Models in LTL are formally even infinite sequences and although they can be represented finitely, one still faces a potential exponential blowup.

From the perspective of unsatisfiability detection, we face a challenge of the discovery of an inductive argument. Indeed, it seems that a form of induction is always needed to make the step from partial results of the form "a short path does not exists" to the ultimate claim "no path (of any length) exists". The option to exhaust each of the exponentially many path lengths separately does not seem to lead to a feasible solution.

The general idea of the presented approach to overcoming the respective challenges rests: 1) on assigning time indexes to signature symbols to deal with the challenge of large models and 2) on a proof repetition detection and replaying technique to deal with the challenge of the discovery of an inductive argument.

Concerning the former, the thesis takes inspiration in the way time is encoded in the planning as satisfiability paradigm [20] or, equivalently, in the related bounded model checking approach [5] known from verification. This solution needs to be complemented by an additional idea in the case of LTL, where we formally need infinitely many indexes to describe the whole model. We devise a way to use labels in the spirit of Lev-Ami et al. [21] to finitely represent clauses over the obtained infinite signature to overcome this obstacle.

Concerning the latter, the thesis proposes a proof repetition detection and replaying technique in a similar way to how the melting rule for loop detection proposed by Horbach and Weidenbach [16] is used for inductive reasoning in the context of superposition for fixed domains [15]. This is, however, only a high-level analogy since the details of the respective studied problems are considerably different.

2. Main contributions

The presented thesis contains five collections of contributions structured into five main chapters. The first three chapters deal with LTL theorem proving, in the fourth the attention moves to the verification of hardware circuits and the final chapter considers automated planning. Several new algorithms are developed and analyzed, their performance is experimentally evaluated, and they are put into the context of related work within the respective research fields.

Although the observation that the problems studied in the thesis are closely related is not new, it is brought to a much more explicit level by devising a single representation for the corresponding reasoning tasks. This enables to approach the problems in a uniform way using resolution and thus to demonstrate how close the problems can be put together and the corresponding tasks aligned on the right level of abstraction. It also paves the way for further exchange of ideas between the research fields. As such, it can be considered one of the main contribution of this thesis. On a more fine-grained level, the five main chapters of the thesis contain the following contributions.

2.1 Labelled superposition for LTL

The first main chapter of the thesis presents LPSup, a new calculus for proving theorems in LTL and uses it as a basis for a new decision procedure for the logic. The main idea is to treat temporal formulas as infinite sets of purely propositional clauses over an extended signature in which symbols are indexed by time moments and to represent these infinite sets by finite sets of labeled propositional clauses. This new representation naturally leads to the replacement of a complex temporal resolution rule, suggested by an existing resolution calculus for LTL, by a fine grained repetition check of finitely saturated labeled clause sets followed by a simple inference.

The developed completeness argument is based on the standard model building idea from superposition. It inherently justifies ordering restrictions, redundancy elimination and effective partial model building. The last property can be directly used to effectively generate counter-examples to non-valid LTL conjectures out of saturated labeled clause sets in a straightforward way.

The computations of LPSup are studied further by interpreting the logic-based, symbolic operations of the calculus on the semantic level of explicit valuations. This perspective later enables to reveal interesting connections to related approaches and to identify the strengths and potential weaknesses of the presented method.

The material of this chapter is a revised and extended version of previously published work [37, 38].

2.2 LTL proving with partial model guidance

Building on the understanding acquired in the previous chapter, chapter two approaches the problem of LTL theorem proving from a different angle. Abandoning

the saturation paradigm and a new decision procedure is designed, based on the observation that LPSup can build partial models on the fly. It is shown how to use these models to drastically restrict the selection of inferences and thus to effectively guide the proof search.

Relying on the previously mentioned connection between conflict driven clause learning and superposition, the model guidance idea is implemented within the SAT solving framework. In more detail, a new decision procedure called LS4 is designed, by using an efficient SAT solver as a subroutine. A non-trivial bookkeeping is needed, however, to maintain the correspondence with the labeled clauses of LPSup, a prerequisite for the discovery of full LTL models as well as for the detection of overall unsatisfiability.

The chapter proves that LS4 is correct and terminating. On an extensive set of LTL benchmarks, it is experimentally demonstrated that the implementation of LS4 is one of the strongest available LTL provers. In a detailed comparison with related work an attempt is then made to discover the key aspects behind LS4's success.

This chapter is based on an earlier publication [39], but has been thoroughly revised and notably extended.

2.3 Variable and clause elimination for LTL

Chapter three studies preprocessing techniques for clause normal forms of LTL formulas. For that purpose, the mechanism of labeled clauses introduced in chapter one is further extended, which here allows to faithfully lift simplification ideas from SAT to LTL. The chapter demonstrates this by adapting variable and clause elimination, an effective preprocessing technique used by modern SAT solvers. Presented experiments confirm that even in the temporal setting substantial reductions in formula size and subsequent decrease of solver runtime can be achieved.

The results of this chapter have been published in [34, 36].

2.4 Reachability, model checking, and triggered clause pushing for PDR

Chapter four returns to the model guidance idea. It first shows how to specialize LS4, the algorithm previously developed for proving theorems in LTL, to decide (non-)reachability in symbolically represented transition systems. The obtained algorithm Reach can be immediately used to verify invariance (and safety) properties of hardware circuits.

It then proceeds to place the new algorithm within the context of related work from the verification area. The fact that Reach builds on the SAT-solver technology and, more specifically, the form of the logical formula it indirectly evaluates make the algorithm related to the bounded model checking method of Biere et al. [5]. On the other hand, the ability of Reach to efficiently detect unsatisfiable instances can be attributed to the distinctive way, in which the algorithm generates and utilizes interpolants [25].

Ultimately, Reach is found closely related to Property Directed Reachability (PDR), also known as IC3, an algorithm recently introduced by Bradley [7]. We show how to transform Reach into PDR by a small change in the core of the algorithm and by the addition of three independent enhancements. This enables viewing PDR from the new perspective of the model guidance idea.

Chapter four continues by proposing triggered clause pushing, an additional improvement of PDR, namely of the clause propagation phase of the algorithm. The idea is to collect models computed by the SAT solver during clause propagation and use them as witnesses for why the respective clauses could not be pushed forward. Witnesses are then used as a pre-filter on the subsequent push operations making them cheaper on average.

The chapter is closed by a detailed experimental evaluation of Reach, of the several extensions of the algorithm leading to PDR, and of PDR improved by triggered clause pushing. The relative utility of the individual improvements is examined along with its dependance on other conditions, such as the chosen search direction and the satisfiability status of the input problem.

2.5 Property Directed Reachability for Automated Planning

Given the exceptional success of PDR in hardware model checking, a natural question arises whether the algorithm could be adapted and applied in the related field of automated planning. Chapter five gives a positive answer to this question.

First, it is noticed that the commonly used encodings from the planning as satisfiability paradigm [19] can be easily modified to yield an input for PDR. However, the chapter also discovers a non-obvious alternative to such a direct combination. It is shown that the SAT solver inside PDR can be replaced by a planning-specific procedure implementing the same interface. This SAT-solver-free variant of the algorithm is not only more efficient, but also offers additional insights and opportunities for further improvements.

This claim is confirmed empirically in the experimental part of chapter five. Then the chapter compares a new implementation of the proposed version of PDR to the state-of-the-art planners to find it highly competitive, solving the most problems on several benchmark domains. Moreover, in a separate set of experiments, PDR is also evaluated with respect to the quality of produced plans, the detection of unsatisfiable planning problems, and in the context of optimal planning.

The material of chapter five has been published in [35].

2.6 Conclusion

The thesis ends with a concluding chapter which provides a unifying perspective on the presented results, summarizes the lessons learned and suggests directions for future research.

Bibliography

[1] Leo Bachmair and Harald Ganzinger. On restrictions of ordered paramodulation with simplification. In Mark E. Stickel, editor, *CADE*, volume 449 of *Lecture Notes in Computer Science*, pages 427–441. Springer, 1990.

[2] Leo Bachmair and Harald Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *J. Log. Comput.*, 4(3):217–247, 1994.

[3] Leo Bachmair and Harald Ganzinger. Resolution theorem proving. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 19–99. Elsevier and MIT Press, 2001.

[4] Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *J. Artif. Intell. Res. (JAIR)*, 22:319–351, 2004.

[5] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In Rance Cleaveland, editor, *TACAS*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 1999.

[6] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[7] Aaron R. Bradley. SAT-based model checking without unrolling. In Ranjit Jhala and David A. Schmidt, editors, *VMCAI*, volume 6538 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2011.

[8] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT Press, 2001.

[9] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[10] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.

[11] Michael Fisher. A resolution method for temporal logic. In John Mylopoulos and Raymond Reiter, editors, *IJCAI*, pages 99–104. Morgan Kaufmann, 1991.

[12] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning – theory and practice*. Elsevier, 2004.

[13] C. Cordell Green. Application of theorem proving to problem solving. In Donald E. Walker and Lewis M. Norton, editors, *IJCAI*, pages 219–240. William Kaufmann, 1969.

[14] Jörg Hoffmann. Everything you always wanted to know about planning - (but were afraid to ask). In Joscha Bach and Stefan Edelkamp, editors, *KI 2011: Advances in Artificial Intelligence, 34th Annual German Conference on AI, Berlin, Germany, October 4-7,2011. Proceedings*, volume 7006 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2011.

[15] Matthias Horbach and Christoph Weidenbach. Superposition for fixed domains. In Michael Kaminski and Simone Martini, editors, *Computer Science Logic, 22nd International Workshop, CSL 2008, 17th Annual Conference of the EACSL, Bertinoro, Italy, September 16-19, 2008. Proceedings*, volume 5213 of *Lecture Notes in Computer Science*, pages 293–307. Springer, 2008.

[16] Matthias Horbach and Christoph Weidenbach. Deciding the inductive validity of $\forall\exists^*$ queries. In Erich Grädel and Reinhard Kahle, editors, *Computer Science Logic, 23rd international Workshop, CSL 2009, 18th Annual Conference of the EACSL, Coimbra, Portugal, September 7-11, 2009. Proceedings*, volume 5771 of *Lecture Notes in Computer Science*, pages 332–347. Springer, 2009.

[17] Barbara Jobstmann and Roderick Bloem. Optimizations for LTL synthesis. In *Formal Methods in Computer-Aided Design, 6th International Conference, FMCAD 2006, San Jose, California, USA, November 12-16, 2006, Proceedings*, pages 117–124. IEEE Computer Society, 2006.

[18] Johan Anthony Willem Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, Calif., 1968.

[19] Henry A. Kautz and Bart Selman. Planning as satisfiability. In *ECAI*, pages 359–363, 1992.

[20] Henry A. Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In William J. Clancey and Daniel S. Weld, editors, *AAAI/IAAI, Vol. 2*, pages 1194–1201. AAAI Press / The MIT Press, 1996.

[21] Tal Lev-Ami, Christoph Weidenbach, Thomas W. Reps, and Mooly Sagiv. Labelled clauses. In Frank Pfenning, editor, *CADE*, volume 4603 of *Lecture Notes in Computer Science*, pages 311–327. Springer, 2007.

[22] João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999.

[23] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969. reprinted in McC90.

[24] Kenneth L. McMillan. *Symbolic model checking*. Kluwer, 1993.

[25] Kenneth L. McMillan. Interpolation and SAT-based model checking. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *CAV*, volume 2725 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2003.

[26] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535. ACM, 2001.

[27] Ingo Pill, Simone Semprini, Roberto Cavada, Marco Roveri, Roderick Bloem, and Alessandro Cimatti. Formal analysis of hardware requirements. In Ellen Sentovich, editor, *Proceedings of the 43rd Design Automation Conference, DAC 2006, San Francisco, CA, USA, July 24-28, 2006*, pages 821–826. ACM, 2006.

[28] Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers with restarts. In Ian P. Gent, editor, *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings*, volume 5732 of *Lecture Notes in Computer Science*, pages 654–668. Springer, 2009.

[29] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977.

[30] John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[31] Kristin Y. Rozier and Moshe Y. Vardi. LTL satisfiability checking. *STTT*, 12(2):123–137, 2010.

[32] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[33] Viktor Schuppan and Luthfi Darmawan. Evaluating LTL satisfiability solvers. In Tevfik Bultan and Pao-Ann Hsiung, editors, *ATVA*, volume 6996 of *Lecture Notes in Computer Science*, pages 397–413. Springer, 2011.

[34] Martin Suda. Variable and clause elimination for LTL satisfiability checking. In Marek Košta and Thomas Sturm, editors, *MACIS 2013 Nanning, China*, pages 60–74, 2013.

[35] Martin Suda. Property directed reachability for automated planning. *J. Artif. Intell. Res. (JAIR)*, 50:265–319, 2014.

[36] Martin Suda. Variable and clause elimination for LTL satisfiability checking. *Mathematics in Computer Science*, 9(3):327–344, 2015.

[37] Martin Suda and Christoph Weidenbach. Labelled superposition for PLTL. In Nikolaj Bjørner and Andrei Voronkov, editors, *LPAR*, volume 7180 of *Lecture Notes in Computer Science*, pages 391–405. Springer, 2012.

[38] Martin Suda and Christoph Weidenbach. Labelled superposition for PLTL. Research Report MPI-I-2012-RG1-001, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, January 2012.

[39] Martin Suda and Christoph Weidenbach. A PLTL-prover based on labelled superposition with partial model guidance. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *IJCAR*, volume 7364 of *Lecture Notes in Computer Science*, pages 537–543. Springer, 2012.

[40] Christoph Weidenbach. *Automated Reasoning – The Art of Generic Problem Solving*. Unpublished.

Author's publications

Reviewed publications relevant for the thesis

1. **M. Suda and C. Weidenbach.** *Labelled superposition for PLTL.* In Nikolaj Bjørner and Andrei Voronkov, editors, *LPAR*, volume 7180 of *Lecture Notes in Computer Science*, pages 391–405. Springer, 2012.
   Cited in:

   - D. Dhungana, C. H. Tang, C. Weidenbach, and P. Wischnewski. *Automated verification of interactive rule-based configuration systems.* In Ewen Denney, Tevfik Bultan, and Andreas Zeller, editors, *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*, pages 551–561. IEEE, 2013.
   - C. Dixon, B. Konev, M. Fisher, and S. Nietiadi. *Deductive temporal reasoning with constraints. J. Applied Logic*, 11(1):30–51, 2013.
   - M. Ferrari, C. Fiorentini, and G. Fiorino. *Towards a tableau-based procedure for PLTL based on a multi-conclusion rule and logical optimizations.* In Davide Ancona, Marco Maratea, and Viviana Mascardi, editors, *Proceedings of the 30th Italian Conference on Computational Logic, Genova, Italy, July 1-3, 2015.*, volume 1459 of *CEUR Workshop Proceedings*, pages 117–121. CEUR-WS.org, 2015.

2. **M. Suda and C. Weidenbach.** *A PLTL-prover based on labelled superposition with partial model guidance.* In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *IJCAR*, volume 7364 of *Lecture Notes in Computer Science*, pages 537–543. Springer, 2012.
   Cited in:

   - D. Dhungana, C. H. Tang, C. Weidenbach, and P. Wischnewski. *Automated verification of interactive rule-based configuration systems.* In Ewen Denney, Tevfik Bultan, and Andreas Zeller, editors, *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*, pages 551–561. IEEE, 2013.
   - R. Williams and B. Konev. *Propositional temporal proving with reductions to a SAT problem.* In Maria Paola Bonacina, editor, *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, volume 7898 of *Lecture Notes in Computer Science*, pages 421–435. Springer, 2013.
   - D. Kapur, R. Nieuwenhuis, A. Voronkov, C. Weidenbach, and R. Wilhelm. *Harald ganzinger's legacy: Contributions to logics and programming.* In Andrei Voronkov and Christoph Weidenbach, editors, *Programming Logics - Essays in Memory of Harald Ganzinger*, volume

7797 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2013.

- R. Goré. *And-or tableaux for fixpoint logics with converse: LTL, CTL, PDL and CPDL.* In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *Automated Reasoning - 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings*, volume 8562 of *Lecture Notes in Computer Science*, pages 26–45. Springer, 2014.
- J. Li, S. Zhu, G. Pu, and M. Y. Vardi. *Sat-based explicit LTL reasoning.* In Nir Piterman, editor, *Hardware and Software: Verification and Testing - 11th International Haifa Verification Conference, HVC 2015, Haifa, Israel, November 17-19, 2015, Proceedings*, volume 9434 of *Lecture Notes in Computer Science*, pages 209–224. Springer, 2015.

3. **M. Suda.** *Variable and clause elimination for LTL satisfiability checking.* In Marek Košta and Thomas Sturm, editors, *MACIS 2013 Nanning, China*, pages 60–74, 2013.

4. **M. Suda.** *Property directed reachability for automated planning. J. Artif. Intell. Res. (JAIR)*, 50:265–319, 2014.

5. **M. Suda.** *Variable and clause elimination for LTL satisfiability checking. Mathematics in Computer Science*, 9(3):327–344, 2015.

Other cited publications relevant for the thesis

1. **M. Suda.** *Triggered clause pushing for IC3. CoRR*, abs/1307.4966, 2013. *Cited in:*

   - K. L. McMillan. *Lazy annotation revisited.* In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 243–259. Springer, 2014.
   - T. Lange, M. R. Neuhäußer, and T. Noll. *IC3 Software Model Checking on Control Flow Automata.* In *Formal Methods in Computer-Aided Design, FMCAD 2015, Austin, Texas, USA, September 27-30, 2015*, pages 97–104. IEEE, 2013.
   - T. Isenberg and H. Wehrheim. *Timed automata verification via IC3 with zones.* In Stephan Merz and Jun Pang, editors, *Formal Methods and Software Engineering - 16th International Conference on Formal Engineering Methods, ICFEM 2014, Luxembourg, Luxembourg, November 3-5, 2014. Proceedings*, volume 8829 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2014.

2. **M. Suda.** *Duality in STRIPS planning. CoRR*, abs/1304.0897, 2013.
Cited in:

   - V. Alcázar and Á. Torralba. *A reminder about the importance of computing and exploiting invariants in planning.* In Ronen I. Brafman, Carmel Domshlak, Patrik Haslum, and Shlomo Zilberstein, editors, *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.*, pages 2–6. AAAI Press, 2015.