TSAT++: an Open Platform for Satisfiability Modulo Theories [1]

Alessandro Armando Claudio Castellini Enrico Giunchiglia
Massimo Idini Marco Maratea

MRG-DIST
University of Genova
Genova, Italy

Abstract

This paper describes TSAT++, an open platform which realizes the *lazy SAT-based approach* to Satisfiability Modulo Theories (SMT). SMT is the problem of determining satisfiability of a propositional combination of T-literals, where T is a first-order theory for which a satisfiability procedure for a set of ground atoms is known. TSAT++ enjoys a modular design in which an enumerator and a theory-specific satisfiability checker cooperate in order to solve SMT. Modularity allows both different enumerators, and satisfiability checkers for different theories (or combinations of theories), to be plugged in, as far as they comply to a simple and well-defined interface. A number of optimization techniques are also implemented in TSAT++, which are independent of the modules used (and of the corresponding theory). Some experimental results are presented, showing that TSAT++, instantiated for *Separation Logic*, is competitive with, or faster than, state-of-the-art solvers for that very logic.

Key words: Boolean Satisfiability, Ground Decision Procedures, Separation Logic, Hardware Verification, Formal Methods

1 Introduction

Satisfiability Modulo Theories (SMT, see [15]) is the problem of determining satisfiability of a propositional combination of T-literals, where T is a simple first-order theory of practical interest, such as, e.g., the theory of arrays, of

[1] We thank Ofer Strichman and Gilles Audemard for valuable suggestions about their solvers. This work is partially supported by COFIN and FIRB projects, and also by MIUR (Italian Ministry of Education, University and Research) under the project RoboCare – A Multi-Agent System with Intelligent Fixed and Mobile Robotic Components.
Email: {armando,drwho,enrico,idini,marco}@mrg.dist.unige.it

This is a preliminary version. The final version will be published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

lists, full linear arithmetic (real and integer) and Separation Logic [17]. By the term "simple" we mean that the theory must have a known satisfiability procedure for a conjunctive set of ground atoms. A number of systems and techniques for SMT have been recently presented (e.g., [4,17,8]), showing that the problem is of great interest. In fact, the behavior of complex infinite-state systems (e.g., real-time hardware and programs) can be rigorously specified in SMT, and automated reasoning can be used to solve the related problems. In other words SMT seems an interesting compromise between expressivity and tractability.

One of the most promising approaches to SMT is the so-called *lazy SAT-based approach* [1,2,8]. The idea is that of managing the search for a model via an efficient machine for Boolean Satisfiability (SAT), e.g., the Davis-Logemann-Loveland algorithm (see, e.g., the seminal paper [7]), and delegating first-order reasoning to an ad-hoc satisfiability procedure for the theory T [2].

As long as this de-coupling is handled smartly, this approach benefits from

(i) the possibility of re-using, for the search, most of the technology and skill achieved in years of research on SAT;

(ii) the possibility to upgrade the T-satisfiability procedure, improving performance and/or extending the range of theories tackled with reasonable effort.

As far as we understand, while great attention has so far been devoted to the theoretical and practical issues of combining decision procedures for various theories, little or no care has been put in building a *practical*, *open* architecture, in which SAT reasoners and satisfiability procedures can be smoothly combined, while retaining good performance. (An attempt is represented by the forthcoming [10].)

Therefore, in this paper we propose a schema for realizing the approach, which modularly combines an enumerator and a satisfiability checker. These two modules take care, in turn, of the search and the first-order reasoning required. Their combination is realized via C++ abstract classes. The system we have built along these lines, TSAT++, is an *open platform* for SMT, in which any such modules can be plugged, as far as they comply with the interfaces defined by the classes.

We also show that a number of optimization techniques, both borrowed from the AI and Formal Methods literature and new, can be smoothly implemented in TSAT++, and that they are *theory-independent*, in that either a technique is implemented by means of the interfaces provided by the abstract classes only, or it is realized entirely within one module; such techniques, then, can be applied to all theories for which a suitable T-satisfiability module, com-

[2] really, just a satisfiability procedure for a conjunctive set of T-atoms is required here; but we will keep the term for the sake of simplicity.

pliant with the interface, is available.

Lastly we show that TSAT++, instantiated for *Separation Logic* (SL, see [17]), performs better than state-of-the-art solvers for SL, both on synthetic and real-world problems coming from industrial test-cases.

The paper is organized as follows: after some preliminaries (Section 2), we present TSAT++'s architecture (Section 3), some of the optimizations implemented (Section 4) and some comparative experimental results (Section 5). Lastly, Section 6 describes conclusions and future work.

2 Background

We assume a basic knowledge of first-order and propositional logic. Assume T is a first-order theory, for which a satisfiability procedure for a conjunctive set of ground atoms is known; then a *T-atom* is an atomic formula of T; a *SMT-atom* is either a T-atom or a propositional atom, and a *SMT-formula* is a Boolean combination of SMT-atoms and SMT-formulas via standard propositional connectives such as $\neg$, $\wedge$ and $\vee$. A *SMT-literal* is a SMT-atom or its negation.

A *SMT-assignment* σ (or simply *assignment*, when it is not ambiguous) is a mapping from variables in a SMT-formula to values in the appropriate domain(s), and propositional atoms to the truth values $\{\bot, \top\}$.

A SMT-assignment σ is extended to map a SMT-formula to $\{\bot, \top\}$ by defining *(1)* $\sigma(\alpha) = \top$ (with α being a T-atom) if and only if α, evaluated under σ, is true with respect to the standard interpretation of the theory T, and *(2)* $\sigma(\phi) = \top$ (with ϕ being a SMT-formula) according to the truth tables of propositional logic. In this respect, an assignment is also viewed as a conjunctive set of SMT-literals in the obvious way.

Let ϕ be a SMT-formula. A SMT-assignment σ *satisfies* ϕ if and only if $\sigma(\phi) = \top$; in that case σ will be called a *SMT-model* of ϕ, or simply a *model*. ϕ is *satisfiable* if and only if there exists a model for it. Here we deal with the decision problem for SMT, that is, the problem of determining whether a SMT-formula is satisfiable or not.

3 System description

According to the so-called *lazy SAT-based approach* (see [1,2,8]), a decision procedure for SMT can be built, which enforces three phases:

(i) *Abstraction.* ϕ is first mapped to a Boolean formula ψ via a bijective map η from distinct T-atoms to (fresh) propositional atoms;

(ii) *Check.* If ψ is unsatisfiable the procedure stops with a negative answer. Otherwise, a Boolean model μ of ψ is determined and the set of T-atoms $\eta^{-1}(\mu)$ is checked for T-satisfiability; if the check succeeds, the procedure stops with a positive answer; otherwise,

(iii) *Refinement.* An unsatisfiable subset of $\eta^{-1}(\mu)$ (called from now on *reason*) is detected and added to ψ; the procedure then goes back to step ii.

We have implemented the lazy SAT-based approach to SMT in a system, called TSAT++, whose architecture is visible in Figure 1.

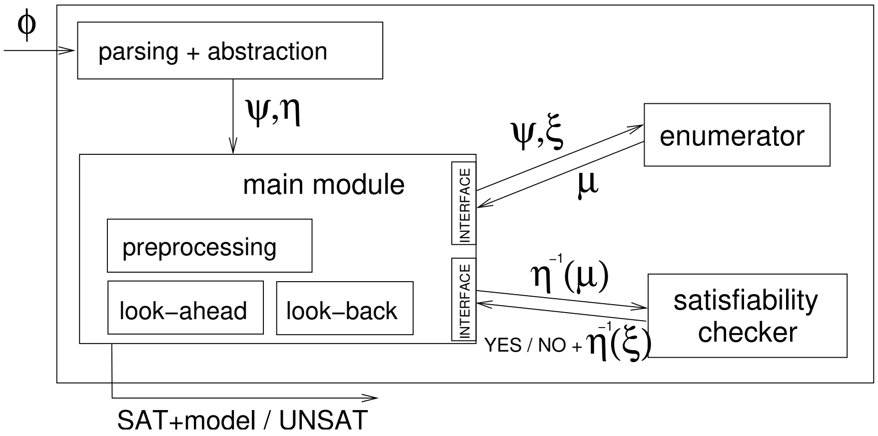

Fig. 1. High-level view of TSAT++.

We now give a brief explanation of the architecture, with respect to the aforementioned three phases. An overview of the interfaces among the modules is then given.

3.1 Functional view

The abstraction phase is quickly carried out by a parsing / abstraction module. As is usually the case, abstraction in the lazy approach basically consists of replacing each distinct T-atom with a fresh propositional variable.

The check phase goes on as follows: a main module receives ψ and η and initially feeds ψ to an enumerator. This module has the task of enumerating (Boolean) models of ψ, which we will indicate generically by μ. In case the formula is propositionally unsatisfiable, that is, there are no (more) models to be enumerated, TSAT++ stops with a negative answer.

Otherwise, the main module evaluates the set of T-atoms $\eta^{-1}(\mu)$ and sends it off to a satisfiability checker module. This module determines whether the set is T-satisfiable or not.

If $\eta^{-1}(\mu)$ is satisfiable, then TSAT++ exits with a positive answer. (In case it is required, it is usually easy then to show a model of ϕ.) Otherwise, a reason, called $\eta^{-1}(\xi)$ in the Figure, is evaluated and sent back to the main module. ξ is then sent to the enumerator and a new μ is requested. The loop goes back to the check phase.

3.2 Architectural view

TSAT++ is written in C++ and naturally exploits the mechanism of *abstract classes* to realize the interfaces to the enumerator and satisfiability checker modules. An interface is defined by a C++ abstract class, and every module must be a concrete instance of the related abstract class (see, for instance, [9]). Each interface consists of *methods* each module can use to interact with the system, and *data structures*, which must be made compatible (i.e., models, sets of T-atoms and so on must be translated from the common data structure defined in TSAT++ to those pertaining to the modules).

Operationally, the code of the module is separately compiled into a static library, which is later on linked to TSAT++. This has two beneficial effects: (a) it completely de-couples TSAT++ from the modules, also at compilation/shipping level, making the development of the modules independent from that of the system; and (b) it enables any module which can be turned into an instance of the related abstract class to be plugged into TSAT++ with a reasonable effort. In principle, incomplete or non-DPLL-based enumerators can be used (at the price of giving up completeness, of course), as well as satisfiability checkers for different theories or combinations of theories.

The driving idea is that of enabling foreign code to be seamlessly integrated into TSAT++, minimizing the effort required to turn any T-satisfiability procedure (such as, e.g., those present in equational theorem provers) or SAT machines into TSAT++ modules. The problem of how to design the interfaces has been solved so far via a criterion of maximum essentiality; in other words, the requirements on the modules have been kept minimal.

In detail, the interface between TSAT++ and the enumerator defines the following services:

(i) getAssignment searches for, and returns, a model μ of ψ; in case ψ is unsatisfiable, it returns the empty assignment.

(ii) isSatisfied returns whether ψ is currently satisfied or not.

(iii) augmentFormula conjoins a Boolean formula (in the signature of ψ) to ψ. So far we work with SMT formulas in Conjunctive Normal Form, so that this method really adds a set of Boolean clauses to ψ.

On the other hand, the interface between TSAT++ and the sat checker defines the following services:

(i) isSatisfied returns whether $\eta^{-1}(\mu)$ is satisfiable or not.

(ii) deduceConstraints deduces new constraints out of the current $\eta^{-1}(\mu)$. In principle the method can return any set of constraints which can be deduced from $\eta^{-1}(\mu)$, also in case it is satisfiable (this could happen, e.g., if *early pruning* is used, see below); so far, it returns one single reason out of an unsatisfiable $\eta^{-1}(\mu)$.

4 Optimization techniques

The interfaces mechanism leaves totally unspecified not only the inner workings of the modules, but also the quality, with respect to the theory tackled, of the objects returned. For instance, TSAT++ cannot possibly influence the way the enumerator searches for models. If on one side this limits the control TSAT++ has over its components, on the other it gives the module developer maximum freedom in choosing heuristics and optimizations.

A wide range of such techniques can then be realized, which are independent of the actual modules, either because they are enforced by TSAT++ via the interfaces or because they are embedded inside a single module. A description of some of the techniques actually implemented in TSAT++ follows; each time we also sketch how the technique is implemented.

(Full) IS(2) preprocessing

This technique detects unsatisfiable *pairs* of T-atoms (T-literals in the full version) and adds them offline to ψ, before the search is started, as binary clauses. IS(2) proved to be quite effective at least in a synthetic class of problems (see [2]). Of course, the technique could be extended from pairs to arbitrary tuples of T-atoms (literals).

IS(2) preprocessing is done completely offline by repeatedly calling the satisfiability checker's method isSatisfied, each time $\eta^{-1}(\mu)$ being a pair of T-atoms (literals).

Augmenting, backjumping and learning

In order to comply with the augmentFormula method, the enumerator must be able to on-the-fly augment its formula. As is well-known from the literature on lazy SAT-based theorem proving, this can either be realized by using an off-the-shelf SAT solver (see [8]) or by engineering the DPLL method in order for it to be more open and flexible. The second option is better from the point of view of efficiency, of course, but it is usually no trivial task to re-engineer an off-the-shelf SAT solver this way.

The enumerator must also be able to "forget" clauses which have been added via augmentFormula during the search, according to some policy, since their number is potentially exponential — but this is left entirely to the module itself.

The enumerator should also be able to take maximum advantage from the constraints which augment its formula on the run; especially, it should be able to *backjump* and *learn* (these techniques are well-known in the SAT literature and are not explained here; the reader can look at [14,6]).

These optimizations are realized entirely inside the enumerator itself.

Reduction of μ

Given μ, we evaluate a *prime implicant* of ψ, that is, a subset $\rho \subseteq \mu$, minimal under set inclusion, which is still a non-contradictory assignment to ψ; then $\eta^{-1}(\rho)$ is sent to the satisfiability checker in place of $\eta^{-1}(\mu)$.

Another way of effectively reducing μ consists of using *triggering*, that is, removing any literal in μ which does not appear in ψ. It can be easily proved that this does not alter the soundness of the approach. Triggering has been introduced in [20] and is also used in a tool called MathSAT [4,5].

In general, reduction of μ is usually useful: if $\eta^{-1}(\mu)$ is satisfiable, so is $\eta^{-1}(\rho)$; if $\eta^{-1}(\mu)$ is unsatisfiable and $\eta^{-1}(\rho)$ is satisfiable, we have anyway found a model of ψ; lastly, if $\eta^{-1}(\mu)$ and $\eta^{-1}(\rho)$ are both unsatisfiable, checking the satisfiability of the latter, rather than of the former, can cause exponentially many fewer satisfiability checks. In fact, any assignment extending ρ is also an abstract model and could potentially be generated and then rejected. (It must be remarked that this technique potentially slows the system down if used in combination with *early pruning*, see below).

Reduction of μ is completely devoted to the main module and is transparent to the modules.

Early pruning

If we relax the requirement that the getAssignment service return *models* of ψ, but rather let it produce *non-contradictory assignments*, that is, models of a subset of ψ, we possibly exploit the obvious fact that an unsatisfiable set of T-literals cannot be made satisfiable by adding more literals.

Feeding such an assignment to the satisfiability checker might produce valuable sets of constraints earlier than if we let the enumerator produce a model, therefore potentially saving time. This technique is similar (but more powerful than) what is usually called *early pruning* and/or *forward checking* in the AI literature (see, e.g., [16,13,19]).

Early pruning is realized entirely inside the enumerator.

Evaluation of ξ

There is usually a large number of constraints that the sat checker can return. In general one must put great care in selecting what to get back to the enumerator, be $\eta^{-1}(\mu)$ satisfiable of not.

Let us concentrate on the latter case so far; a generally useful strategy seems that of looking for *small* reasons; this is motivated by the obvious fact that small clauses can usually prune a larger portion of the search space than long ones. This is especially true if the enumerator maintains a search tree, as is the case of all DPLL implementations.

In the latter case a further beneficial strategy can be that of evaluating the $\eta^{-1}(\mu)$ that is minimal with respect to the ordering induced on T-literals by the search tree — what could be called the "shallowest" set. Such constraints might help the enumerator backjump as high as possible.

Generalizing a little, note that TSAT++ is not limited to augmenting ψ with one single clause; therefore, even in the simple case of detecting reasons from an unsatisfiable set of T-literals, one idea is that of returning *a subset* of all reasons.

The evaluation of ξ is realized entirely inside the satisfiability checker.

5 Experimental results

We have instantiated the architecture described in Section 3 for *Separation Logic* (SL), a decidable theory combining Boolean logic with a fragment of linear arithmetic able to compactly capture the behavior of a large class of infinite-state systems. Recently a number of interesting problems of AI (planning, scheduling, temporal reasoning, e.g., in [19,4]) and Formal Methods (safety of hardware, bounded model checking of real-time systems, e.g., in [17]) have been recast as decision problems of SL-formulas. Hence, the need of efficient decision procedures for this logic.

Currently, the enumerator is a modified version of SIMO [11], a Chaff-like SAT solver; the satisfiability checker module for SL relies on the Bellman-Ford algorithm. It is easy for it to return the smallest reason after detection of unsatisfiability.

We compare TSAT++ with the two SL decision procedures *MathSAT* [4] and *SEP* [17]. The chosen benchmarks are: (1) the *diamonds* problems as defined in [17]; and (2) a set of "real-world" problems, representing bounded model checking for timed automata (see [4]) and hardware verification problems originally generated by UCLID (see [12]) and appearing in the SMT library. Experiments were run on a Pentium IV 2.4GHz, with 1GHz of RAM. We report search CPU time in seconds. In the following Tables, TIME means that the process was stopped after 1000 seconds.

Diamonds.

Given a parameter D, these problems are characterized by an exponentially large (2^D) number of Boolean models, some of which correspond to SL-models; hard instances with a unique SL-model can be generated. A second parameter, S, is used to make SL models larger, further increasing the difficulty. Variables range over the reals.

Table 1 shows comparative results on the diamonds problems. The third column denotes whether the problem has a unique SL-model; the remaining columns show CPU times for TSAT++ with prime implicants reduction and smallest reason detection, SEP with and without conjunction matrix and MathSAT.[3] TSAT++ clearly performs best, often by orders of magnitude;

[3] The configurations employed were suggested by the authors of SEP and MathSAT.

Table 1
Diamonds problems.

D	S	unique	TSAT++	SEP	SEP (no c.m.)	MathSAT
50	4	N	0	0.03	0.12	0.05
50	4	Y	0.01	0.84	0.07	TIME
100	5	N	0.01	0.13	1.18	0.61
100	5	Y	0.04	10.20	0.17	TIME
250	5	N	0.08	0.95	52.20	5.4
250	5	Y	0.21	288.30	0.77	TIME
500	5	N	0.29	5.92	742.99	21.22
500	5	Y	1.05	TIME	4.85	TIME

Table 2
Real-world problems.

Instance (cat.)	TSAT++	SEP (no c.m.)	MathSAT
abz5-900 (a)	1.9	TIME	0.82
ring2-10 (a)	0.02	0.01	0.01
ring2-100 (a)	2.07	0.18	1.07
post-office 4/10 (a)	0.51	TIME	1.06
post-office 4/11 (a)	1.01	TIME	2.13
post-office 4/12 (a)	0.58	TIME	0.91
LD-ST-neg.3step (b)	0.03	0.42	-
OOO-neg.3step (b)	0.01	0.23	-

instances with a unique solution are more difficult than non-unique ones, as expected, except for SEP without conjunction matrix.

Real-world problems.

These problems represent (a) scheduling and bounded model checking for timed automata, (b) verification of hardware models, including a load-store unit and an out-of-order execution unit.

Consider Table 2: we compare TSAT++ (full IS(2) preprocessing, prime implicants reduction and detection of the smallest reason for the post-office problem and early-pruning plus smallest reason for the others), SEP without conjunction matrix and MathSAT, on the biggest problems SEP could tackle, found in either category. In category (b) variables are restricted to take integer values, and MathSAT has been excluded from the comparison. As one can see, TSAT++ is competitive with, or faster than, its competitors uniformly. It is worth remarking that MathSAT is customized for the post-office problem, and SEP is customized for SL.

6 Conclusions

SMT is an interesting and challenging problem, mainly thanks to its expressivity; and the SAT-based approach represents a general solving paradigm for this problem, at the same time being able to achieve high performance. The essence of the approach is the interplay between the search, managed by an enumerator module, and the first-order reasoning, delegated to a specialized satisfiability procedure. Much research has recently been done along these lines, see, e.g., the results achieved by such systems as *Tsat* [2], *CVC* [18], *ICS* [8], *MathSAT* [4] and UCLID [12] (SEP was actually born as a back-end to UCLID).

We think openness and modularity are crucial here in order to gain in terms of flexibility, upgradability and extendability. This is why we have built the system TSAT++, in which the two facets of the lazy SAT-based approach to SMT are embedded in two dedicated modules, managed thanks to simple, well-defined C++ abstract classes. An open question, of course, is *how open* the system can be kept without loosing performance.

In this paper, in particular, we have described the architecture of the system, giving an overview of some optimization techniques we have implemented. These techniques are theory-independent, in that they are either realized by using the services provided by the interfaces, or they are confined to the modules.

Lastly, we have shown some experimental results in which TSAT++, instantiated for Separation Logic, is competitive with or faster than specialized solvers for that logic; this is at least an initial indication that keeping a system modular does not necessarily degrade its performance. More experimental results obtained by TSAT++ can be seen in [3]; moreover, up-to-date information about the system, the benchmarks used, and an executable of TSAT++ can be found at `http://www.mrg.dist.unige.it/Tsat`. We plan to make the source code available to the community in a reasonable time, therefore enlarging the spectrum of possible applications.

References

[1] A. Armando and E. Giunchiglia. Embedding Complex Decision Procedures inside an Interactive Theorem Prover. *Annals of Mathematics and Artificial Intelligence*, 8(3–4):475–502, 1993.

[2] Alessandro Armando, Claudio Castellini, and Enrico Giunchiglia. SAT-based procedures for temporal reasoning. In Susanne Biundo and Maria Fox, editors, *Proceedings of the 5th European Conference on Planning (Durham, UK)*, volume 1809 of *Lecture Notes in Computer Science*, pages 97–108. Springer, 2000.

[3] Alessandro Armando, Claudio Castellini, Enrico Giunchiglia, and Marco Maratea. A sat-based decision procedure for the boolean combination of difference constraints, May 2004. To appear at SAT 2004, Vancouver, BC, Canada.

[4] Gilles Audemard, Piergiorgio Bertoli, Alessandro Cimatti, Artur Kornilowicz, and Roberto Sebastiani. A SAT based approach for solving formulas over boolean and linear mathematical propositions. In Andrei Voronkov, editor, *Automated Deduction – CADE-18*, volume 2392 of *Lecture Notes in Computer Science*, pages 195–210. Springer-Verlag, July 27-30 2002.

[5] Gilles Audemard, Alessandro Cimatti, Artur Kornilowicz, and Roberto Sebastiani. Bounded model checking for timed systems. In *IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE), LNCS*, volume 22, 2002.

[6] R. J. Bayardo, Jr. and D. P. Miranker. A complexity analysis of space-bounded learning algorithms for the constraint satisfaction problem. In *Proc. AAAI*, pages 298–304, 1996.

[7] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7), 1962.

[8] Leonardo de Moura, Harald Rueß, and Maria Sorea. Lazy theorem proving for bounded model checking over infinite domains. *Lecture Notes in Computer Science*, 2392:438–455, 2002.

[9] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison Wesley, Reading, Mass., 1995.

[10] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T): Fast decision procedures. july 2004. to appear at CAV 2004.

[11] Enrico Giunchiglia, Marco Maratea, and Armando Tacchella. Look-ahead vs. look-back techniques in a modern sat solver. Accepted at SAT03 - Sixth International Conference on Theory and Applications of Satisfiability Testing, Portofino, Italy, May 2003.

[12] Shuvendu K. Lahiri, Sanjit A. Seshia, and Randal E. Bryant. Modeling and verification of out-of-order microprocessors in UCLID. *Lecture Notes in Computer Science*, 2517:142–159, 2002.

[13] A. Oddi and A. Cesta. Incremental forward checking for the disjunctive temporal problem. In *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI-2000)*, pages 108–112, Berlin, 2000.

[14] Patrick Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, 1993.

[15] Silvio Ranise, Cesare Tinelli, et al. The SMT library (Satisfiability Modulo Theory). http://www.smtlib.org.

[16] Kostas Stergiou and Manolis Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120(1):81–117, 2000.

[17] Ofer Strichman, Sanjit A. Seshia, and Randal E. Bryant. Deciding separation formulas with SAT. *Lecture Notes in Computer Science*, 2404:209–222, 2002.

[18] Aaron Stump, Clark W. Barrett, and David L. Dill. CVC: a cooperating validity checker. In J. C. Godskesen, editor, *Proceedings of the International Conference on Computer-Aided Verification*, Lecture Notes in Computer Science, 2002.

[19] Ioannis Tsamardinos and Martha Pollack. Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence*, 2003. To appear.

[20] Steven Wolfman and Daniel Weld. The LPSAT-engine & its application to resource planning. In *Proc. IJCAI-99*, 1999.