The MathSAT 3 System ⋆

Marco Bozzano[1], Roberto Bruttomesso[1], Alessandro Cimatti[1], Tommi Junttila[2], Peter van Rossum[3], Stephan Schulz[4], and Roberto Sebastiani[5]

[1] ITC-IRST, Via Sommarive 18, 38050 Povo, Trento, Italy
[2] Helsinki University of Technology, P.O.Box 5400, FI-02015 TKK, Finland
[3] Radboud University, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands.
[4] University of Verona, Strada le Grazie 15, 37134 Verona, Italy
[5] Università di Trento, Via Sommarive 14, 38050 Povo, Trento, Italy

1 Introduction

Satisfiability Modulo Theories (SMT) can be seen as an extended form of propositional satisfiability, where propositions are either simple boolean propositions or quantifier-free atomic constraints in a specific theory. In this paper we present MATHSAT version 3 [6–8], a DPLL-based decision procedure for the SMT problem for various theories, including those of Equality and Uninterpreted Functions ($\mathcal{EUF}$) [1], of Separation Logic ($\mathcal{SEP}$), and of Linear Arithmetic on the Reals ($\mathcal{LA}(\mathbb{R})$) and on the integers ($\mathcal{LA}(\mathbb{Z})$). MATHSAT is also able to solve the SMT problem for combined $\mathcal{EUF}$ +$\mathcal{SEP}$, $\mathcal{EUF}$ +$\mathcal{LA}(\mathbb{R})$, and $\mathcal{EUF}$ +$\mathcal{LA}(\mathbb{Z})$, either by means of Ackermann's reduction [1] or using our new approach called Delayed Theory Combination (DTC) [6], which is alternative to the classical Nelson-Oppen or Shostak integration schemata.

MATHSAT is based on the approach of integrating a state-of-the-art SAT solver with a hierarchy of dedicated theory solvers. It is a re-implementation of an older version of the same tool [3, 4], supporting more extended theories, and their combination, and implementing a number of important optimization techniques [2].

MATHSAT has been and is currently used in many projects, both as a platform for experimenting new automated reasoning techniques, and as a workhorse reasoning procedure onto which to develop formal verification tools. Our main target application domains are those of formal verification of RTL circuit designs, and of timed and hybrid systems. MATHSAT has been and is currently widely used by many authors for empirical tests on SMT problems (see, e.g., [11, 12, 2]).

For lack of space, in this paper we omit any description of empirical results, which can be found in [6–8]. A Linux executable of the solver, together with the papers [6–8] and the benchmarks used there, is available from `http://mathsat.itc.it/`.

⋆ This work has been partly supported by ISAAC, an European sponsored project, contract no. AST3-CT-2003-501848, by ORCHID, a project sponsored by Provincia Autonoma di Trento, and by BOWLING, a project sponsored by a grant from Intel Corporation. The work of T. Junttila has also been supported by the Academy of Finland, projects 53695 and 211025.

[1] More precisely, $\mathcal{EUF}$ with some extensions for arithmetic predicates and constants.

[2] We notice that some ideas related to the mathematical solver(s) presented in this paper (i.e., layering, stack-based interfaces, theory-deduction) are to some extent similar to ideas pioneered by Constraint Logic Programming (see, e.g., [14]).

2 The main procedure

MATHSAT is built on top of the standard "online" lazy integration schema used in many SMT tools (see, e.g., [3, 11, 2]). In short: after some preprocessing to the input formula ϕ, a DPLL-based SAT solver is used as an enumerator of (possibly partial) truth assignments for (the boolean abstraction of) ϕ; the consistency in $\mathcal{T}$ of (the set of atomic constraints corresponding to) each assignment is checked by a solver $\mathcal{T}$-SOLVER. This is done until either one $\mathcal{T}$-consistent assignment is found, or all assignments have been checked.

MATHSAT 3 [6–8] is a complete re-implementation of the previous versions described in [3, 4], supporting more theories (e.g., $\mathcal{EUF}$, $\mathcal{EUF}+\mathcal{LA}(\mathbb{R})$, $\mathcal{EUF}+\mathcal{LA}(\mathbb{Z})$) and a richer input language (e.g., non-clausal forms, if-then-else). It features a new preprocessor, a new SAT solver, and a much more sophisticate theory solver, which we describe in this section. It also features new optimization techniques, which we describe in the next sections. [3]

2.1 The Preprocessor

First, MATHSAT allows the input formula to be in non-clausal form and to include operations such as if-then-else's over non-boolean terms. The preprocessor translates the input formula into conjunctive normal form by using a standard linear-time satisfiability preserving translation. Second, the input formula may contain constraints that mix theories in a way that cannot be handled either by the $\mathcal{EUF}$ solver or the linear arithmetic solver alone (e.g., $f(x) + f(z) = 4$). To handle these constraints, the preprocessor either (i) eliminates them by applying Ackermann's reduction [1] or (ii) purifies them into a normal form if the Delayed Theory Combination scheme [6] is applied.

2.2 The SAT Solver

In MATHSAT, the boolean solver is built upon the MINISAT solver [10]. Thus it inherits for free conflict-driven learning and backjumping [15], restarts [13], optimized boolean constraint propagation based on the two-watched literal scheme [16], and an effective splitting heuristics VSIDS [16]. It communicates with $\mathcal{T}$-SOLVER through a stack-based interface that passes assigned literals, $\mathcal{T}$-consistency queries and backtracking commands to $\mathcal{T}$-SOLVER, and gets back answers to the queries, $\mathcal{T}$-inconsistent sets (theory conflict sets) and $\mathcal{T}$-implied literals.

2.3 The Theory Solver $\mathcal{T}$-SOLVER

$\mathcal{T}$-SOLVER gets in input a set of quantifier-free constraints μ and checks whether μ is $\mathcal{T}$-satisfiable or not. In the first case, it also tries to perform and return deductions in the form $\mu' \models_{\mathcal{T}} l$ s.t. l is a literal representing a truth assignment to a not-yet-assigned atom occurring in the input formula, and $\mu' \subseteq \mu$ in the (possibly minimal) set of literals

[3] In order to distinguish what is new in MATHSAT 3 wrt. previous versions, in the next sections we label by "[3, 4]" those techniques which were already present in previous versions, with "[7, 8]" the new techniques, and with "[3, 4, 7, 8]" those techniques which have been proposed in earlier versions and have been signifi cantly improved or extended in MATHSAT 3.

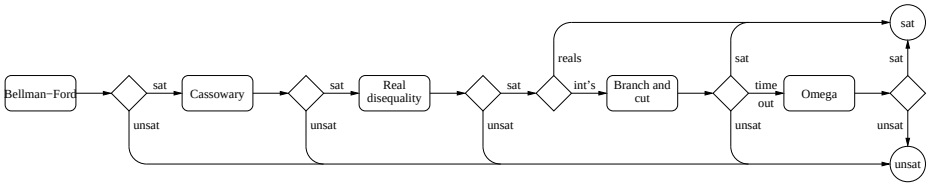

Fig. 1. The control fbw of the linear solver.

entailing l. In the second case, it returns the (possibly minimal) sub-assignment $\mu' \subseteq \mu$ which caused the inconsistency (*conflict set*). Due to the early pruning step (see § 3), $\mathcal{T}$-SOLVER is typically invoked on *incremental* assignments. When a conflict is found, the search backtracks to a previous point, and $\mathcal{T}$-SOLVER then restarts from a previously visited state. Based on these considerations, $\mathcal{T}$-SOLVER has a persistent state, and is *incremental* and *backtrackable*: incremental means that it avoids restarting the computation from scratch whenever it is given as input an assignment μ' such that $\mu' \supset \mu$ and μ has already been proved satisfiable; backtrackable means that it is possible to return to a previous state on the stack in a relatively efficient manner.

$\mathcal{T}$-SOLVER consists mainly on two main layers: an *Equational Satisfiability Procedure* for $\mathcal{EUF}$ and a *Linear Arithmetic Procedure for* $\mathcal{SEP}$, $\mathcal{LA}(\mathbb{R})$ *and* $\mathcal{LA}(\mathbb{Z})$.

The Equational Satisfiability Procedure The first layer of $\mathcal{T}$-SOLVER is provided by the equational solver, a satisfiability checker for $\mathcal{EUF}$ with minor extensions for arithmetic predicates and constants. The solver is based on the congruence closure algorithm presented in [17], and reuses some of the data structures of the theorem prover E [19] to store and process terms and atoms. It is incremental and supports efficient backtracking. The solver generates conflict sets and produces deductions for equational literals. It also implicitly knows that syntactically different numerical constants are semantically distinct, and efficiently detects and signals if a new equation forces the identification of distinct domain elements.

The Linear Arithmetic Procedure The second layer of $\mathcal{T}$-SOLVER is given by a procedure for the satisfiability of sets of linear arithmetic constraints in the form $(\sum_i c_i v_i \bowtie c_j)$, with $\bowtie \in \{=, \neq, >, <, \geq, \leq\}$. The linear solver is *layered*, running faster, more general solvers first and using slower, more specialized solvers only if the early ones do not detect an inconsistency. The control flow through the linear solver is given in Fig. 1.

First, we consider only those constraints that are in the difference logic fragment, i.e., the subassignment of μ consisting of all constraints of the forms $v_i - v_j \bowtie c$ and $v_i \bowtie c$, with $\bowtie \in \{=, \neq, <, >, \leq, \geq\}$. Satisfiability checking for this subassignment is performed by an incremental version of the Bellman-Ford algorithm [9], which allows for deriving minimal conflict sets. Second, we try to determine if the current assignment μ is consistent over the reals, by means of the Cassowary constraint solver. Cassowary [5] is a simplex-based solver over the reals, using slack variables to efficiently allow the addition and removal of constraints and the generation of a minimal conflict set. Cassowary has been extended by an ad-hoc technique to handle disequalities on $\mathbb{R}$ and with arbitrary precision arithmetic.

If the variables are interpreted over the integers, and the problem is unsatisfiable in the reals, then it is so in the integers. When the problem is satisfiable in the reals, a simple form of branch-and-cut is carried out, to search for solutions over the integers, using Cassowary's incremental and backtrackable machinery. If branch-and-cut does not find either an integer solution or a conflict within a small, predetermined amount of search, the Omega constraint solver [18] is called on the current assignment.

3 Tightly-integrated SAT and Theory Solvers

In MATHSAT the naive DPLL+$\mathcal{T}$-SOLVER integration schema is enriched by the following optimization techniques [3, 4, 7, 8]. Apart from theory-driven backjumping and learning, all these optimizations can be disabled/enabled by the user.

Early pruning [3, 4] Before every boolean decision step, $\mathcal{T}$-SOLVER is invoked on the current assignment μ. If this is found unsatisfiable, then there is no need to proceed, and the procedure backtracks.

Theory-driven backjumping [3, 4, 7, 8] When $\mathcal{T}$-SOLVER finds the assignment μ to be $\mathcal{T}$-unsatisfiable, it also returns a conflict set η causing the unsatisfiability. This enables MINISAT to backjump in its search to the most recent branching point in which at least one literal $l \in \eta$ is not assigned a truth value, pruning the search space below.

Theory-driven learning [3, 4, 7, 8] When $\mathcal{T}$-SOLVER returns a conflict set η, the clause $\neg\eta$ can be added in conjunction to φ: this will prevent MINISAT from generating again any branch containing η.

Theory-driven deduction (and learning) [3, 4, 7, 8] With early pruning, if a call to $\mathcal{T}$-SOLVER produces some deduction in the form $\mu' \models_{\mathcal{T}} l$ (e.g., by the $\mathcal{EUF}$ solver), then l is returned to the SAT solver, which uses it for boolean constraint propagation, triggering new boolean simplification. Moreover, the implication clause $\mu \rightarrow l$ can be learned and added to the main formula, pruning the remaining boolean search.

Static learning [7, 8] Before the main solver is invoked, short clauses valid in $\mathcal{T}$ like, e.g., $\neg(t = 1) \vee \neg(t = 2)$, $\neg(t_1 - t_2 \leq 3) \vee (t_2 - t_1 > -4)$, $(t_1 - t_2 \leq 3) \wedge (t_2 - t_3 < 5) \rightarrow (t_1 - t_3 < 9)$, are added off-line to the input formula ϕ if their atoms occur in ϕ. This helps pruning the search space at the boolean level.

Clause Discharge [7, 8] MATHSAT inherits MINISAT's feature of periodically discarding some of the learned clauses to prevent explosion of the formula size. However, because the clauses generated by theory-driven learning and theory deduction mechanisms may have required a lot of work in $\mathcal{T}$-SOLVER, as a default option they are never discarded.

Control on split literals [3, 4, 7, 8] In MATHSAT it is possible to initialize the weights of the literals in the VSIDS splitting heuristics so that either boolean or mathematical atoms are preferred as splitting choices early in the DPLL search.

4 An Optimized Theory Solver

In MATHSAT, $\mathcal{T}$-SOLVER benefits of the following optimization techniques [3, 4, 7, 8]. All these optimizations can be disabled/enabled by the user.

Clustering [7, 8] At the beginning of the search, the set of all atoms occurring in the formula is partitioned into disjoint *clusters* $L_1, ..., L_k$: intuitively, two atoms (literals) belong to the same cluster if they share a variable. Thus, instead of having a single, monolithic solver for linear arithmetic, k different solvers are constructed, each responsible for the handling of a single cluster. This allows for "dividing-and-conquering" the mathematical component of search, and for generating shorter conflict sets.

EQ-Layering [7, 8] The equational solver can be used not only as a solver for $\mathcal{EUF}$, but also as a layer in the arithmetic reasoning process. In that case, *all* constraints, including those involving arithmetic operators, are passed to the equational solver. Arithmetic function symbols are treated as fully uninterpreted. However, the solver has a limited interpretation of the predicates $<$ and $\leq$, knowing only that $s < t$ implies $s \neq t$, and $s = t$ implies $s \leq t$ and $\neg(s < t)$. Thus, the equational interpretation is a (rough) approximation of the arithmetic interpretation, and all conflicts and deductions found by the equational solver under $\mathcal{EUF}$ semantics are valid under fully interpreted semantics. Hence, they can be used to prune the search. Thus, given the efficiency and the deduction capabilities of the equality solver, this process in many cases significantly improves the overall performances [8].

Filtering [3, 4, 7, 8] MATHSAT simplifies the set of constraints passed to $\mathcal{T}$-SOLVER by "filtering" unnecessary literals. If an atom ψ which occurs only positively (resp., negatively) in the input formula ϕ is assigned to false (resp., true) in the current truth assignment μ, then it is dropped from μ without loosing correctness and completeness. If an atom ψ is assigned by unit propagation on clauses resulting from theory-driven learning, theory-driven deduction, or static learning, then it is dropped from the set of constraints μ to check because it is a consequence in $\mathcal{T}$ of other literals in μ.

Weakened Early Pruning [7, 8] During early pruning calls, $\mathcal{T}$-SOLVER does not have to detect *all* inconsistencies; as long as calls to $\mathcal{T}$-SOLVER at the end of a search branch faithfully detect inconsistency, correctness is guaranteed. We exploit this by using a faster, but less powerful version of $\mathcal{T}$-SOLVER for early pruning calls. Specifically, as the theory of linear arithmetic on $\mathbb{Z}$ is much harder, in theory and in practice, than that on $\mathbb{R}$, during early pruning calls, $\mathcal{T}$-SOLVER looks for a solution on the reals only.

5 Delayed Theory Combination of $\mathcal{T}$-solvers

In the standard Nelson-Oppen's (or Shostak's) approaches, the $\mathcal{T}$-solvers for two different theories T_1, T_2 interact with each other by deducing and exchanging (disjunctions of) *interface equalities* in the form $(v_i = v_j)$, v_i, v_j being variables labeling terms in the different theories; the SAT solver interacts with this integrated $T_1 \cup T_2$-solver according to the standard SMT paradigm.

In Delayed Theory Combination (DTC) [6], instead, the SAT solver is in charge of finding suitable truth value assignments not only to the atoms occurring in the formula, but also for their relative interface equalities. Each $\mathcal{T}$-solver works in isolation, without direct exchange of information, and interacts only with the SAT solver, which gives it as input not only the set of atomic constraints for its specific theory, but also the truth value assignment to the interface equalities. Under such conditions, two theory-specific models found by the two $\mathcal{T}$-solvers can be merged into a model for the input formula.

We notice the following facts [6]. DTC does not require the direct combination of $\mathcal{T}$-solvers for T_1 and T_2; the construction of conflict sets involving multiple theories is straightforward; with DTC the $\mathcal{T}$-solvers are not required to have deduction capabilities (though, the integration benefits from them); DTC extends naturally to the case of more than two theories; DTC does not suffer in the case of non-convex theories, like $\mathcal{LA}(\mathbb{Z})$.

References

1. W. Ackermann. *Solvable Cases of the Decision Problem*. North Holland Pub. Co., Amsterdam, 1954.
2. A. Armando, C. Castellini, E. Giunchiglia, and M. Maratea. A SAT-based Decision Procedure for the Boolean Combination of Difference Constraints, 2004. In SAT 2004 conference.
3. G. Audemard, P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani. A SAT Based Approach for Solving Formulas over Boolean and Linear Mathematical Propositions. In *CADE-18*, volume 2392 of *LNAI*, pages 195–210. Springer, 2002.
4. G. Audemard, P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani. Integrating boolean and mathematical solving: Foundations, basic algorithms and requirements. In *AISC 2002*, volume 2385 of *LNAI*, pages 231–245. Springer, 2002.
5. G. J. Badros, A. Borning, and P. J. Stuckey. The Cassowary Linear Arithmetic Constraint Solving Algorithm. *ACM Trans. on Computer-Human Interaction*, 8(4):267–306, 2001.
6. M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, S. Ranise, P. van Rossum, and R. Sebastiani. Effi cient Satisfi ability Modulo Theories via Delayed Theory Combination. In *CAV 2005*, LNCS. Springer, 2005.
7. M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani. An incremental and Layered Procedure for the Satisfi ability of Linear Arithmetic Logic. In *TACAS 2005*, volume 3440 of *LNCS*, pages 317–333. Springer, 2005.
8. M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani. MathSAT: Tight Integration of SAT and Mathematical Decision Procedure. *Journal of Automated Reasoning*, to appear.
9. B. V. Cherkassky and A. V. Goldberg. Negative-Cycle Detection Algorithms. *Mathematical Programming*, 85(2):277–311, 1999.
10. N. Eén and N. Sörensson. An extensible SAT-solver. In *SAT 2003*, volume 2919 of *LNCS*, pages 502–518. Springer, 2004.
11. C. Flanagan, R. Joshi, X. Ou, and J.B. Saxe. Theorem Proving using Lazy Proof Explication. In *CAV 2003*, volume 2725 of *LNCS*, pages 355–367. Springer, 2003.
12. H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In *CAV 2004*, volume 3114 of *LNCS*, pages 175–188. Springer, 2004.
13. C. P. Gomes, B. Selman, and H. A. Kautz. Boosting Combinatorial Search Through Randomization. In *AAAI/IAAI 1998*, pages 431–437. AAAI Press / The MIT Press, 1998.
14. J. Jaffar, S. Michaylov, P. J. Stuckey, and R. H. C. Yap. The CLP(R) Language and System. *ACM Trans. on Programming Languages and Systems*, 14(3):339–395, 1992.
15. J. P. Marques-Silva and K. A. Sakallah. GRASP: A Search Algorithm for Propositional Satisfi ability. *IEEE Trans. on Computers*, 48(5):506–521, 1999.
16. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Effi cient SAT Solver. In *DAC 2001*, pages 530–535. ACM, 2001.
17. R. Nieuwenhuis and A. Oliveras. Congruence Closure with Integer Offsets. In *LPAR 2003*, volume 2850 of *LNAI*, pages 78–90. Springer, 2003.
18. Omega. `http://www.cs.umd.edu/projects/omega`.
19. S. Schulz. E – A Brainiac Theorem Prover. *AI Communications*, 15(2/3):111–126, 2002.