# The MathSAT 4 SMT Solver (Tool Paper)

Roberto Bruttomesso[1], Alessandro Cimatti[1], Anders Franzén[1], Alberto Griggio[2], and Roberto Sebastiani[2]

[1] FBK-IRST, Povo, Trento, Italy. {bruttomesso,cimatti,franzen}@fbk.eu
[2] DISI, Università di Trento, Italy. {griggio,rseba}@disi.unitn.it

**Abstract.** We present MathSAT 4, a state-of-the-art SMT solver. MathSAT 4 handles several useful theories: (combinations of) equality and uninterpreted functions, difference logic, linear arithmetic, and the theory of bit-vectors. It was explicitly designed for being used in formal verification, and thus provides functionalities which extend the applicability of SMT in this setting. In particular: model generation (for counterexample reconstruction), model enumeration (for predicate abstraction), an incremental interface (for BMC), and computation of unsatisfiable cores and Craig interpolants (for abstraction refinement).

## 1 Introduction

In this paper we present MathSAT 4, a modern Satisfiability Modulo Theories (SMT) solver. Despite its "traditional" name, MathSAT 4 has been completely redesigned and reimplemented from scratch, and it is thus a completely new system wrt. MathSAT 3 [3]. Unlike its predecessors, MathSAT 4 has been explicitly designed for being used in a formal verification setting: in fact, besides extending the set of "traditional" theories of interest with the theory of bit-vectors [4], it also provides novel functionalities which are particularly targeted for usage in FV. To this extent, MathSAT 4 has recently been integrated within the NuSMV model checker [7] as a workhorse engine for formal verification of word-level circuits and of timed and hybrid systems.

MathSAT 4 is available at its web page (`http://mathsat4.disi.unitn.it`), with documentation, related papers and some performance figures.

## 2 Architecture

MathSAT 4 is based on the lazy integration schema used in many SMT tools (see, e.g., [16]). The high-level architecture of the system is shown in Figure 1.
**Interface.** MathSAT 4 is written in C++. Interaction with MathSAT can be performed either via files or via a rich C API. The system supports three different input formats: a native (MSAT) one, the standard SMT-LIB one, and the one used by the FOCI [13] interpolating prover. (The API supports also the possibility to use MathSAT 4 as a Theory Context Checker (TCC) [6], see below.) MathSAT 4 can also interface with external Boolean unsat-core extractors by exchanging purely-Boolean CNF formulas as DIMACS files.

**Preprocessor.** After the input formula $\varphi$ is parsed (or generated through the API), a preprocessing step is performed, consisting of three parts. First, the problem is simplified by encoding equivalent theory atoms ($\mathcal{T}$-atoms) into a unique representation, and by propagating top-level information.[1] Second, the formula is converted to CNF. Finally, MATHSAT applies static learning [3], i.e. it adds to the formula small clauses representing $\mathcal{T}$-valid lemmas (e.g. transitivity constraints) which can speed up the Boolean reasoning process.

**DPLL Engine.** The core of the solver is the DPLL Engine. It receives as input the CNF conversion of the original problem, and drives the search by enumerating its propositional models and invoking the $\mathcal{T}$-solver(s) to check them for consistency, until either a model is found or all of them are found inconsistent. The DPLL Engine is based on the highly-efficient MINISAT 2 SAT solver.

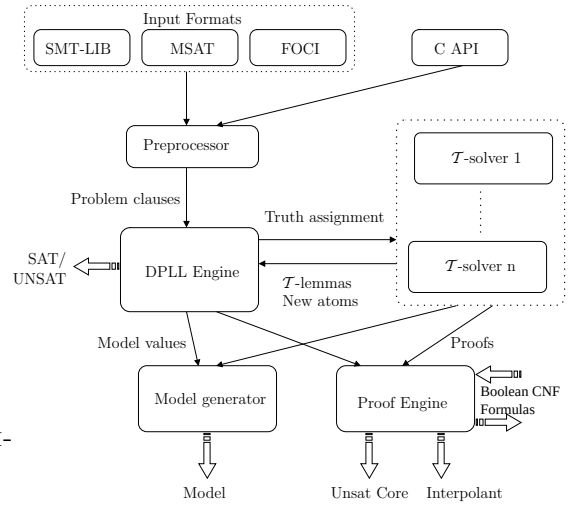


**Fig. 1.** MATHSAT 4 architecture.

Like its predecessors, MATHSAT 4 implements most of the techniques for optimizing the interaction of DPLL and $\mathcal{T}$-solvers (see [16] for a survey). Notably, it implements also a novel adapting heuristic for controlling the interleaving between DPLL steps and $\mathcal{T}$-solver calls.

**Theory solvers.** In MATHSAT 4 the $\mathcal{T}$-solvers are organized as a *layered hierarchy* of solvers of increasing expressivity and complexity [3, 4]: if a higher-level solver finds a conflict, then this conflict is used to prune the search at the Boolean level; if it does not, the lower level solvers are activated. These $\mathcal{T}$-solvers implement state-of-the-art procedures for the theories of equality and uninterpreted functions and predicates ($\mathcal{EUF}$) plus numeric constants and some numeric relations [14], for difference logic ($\mathcal{DL}$) [10], for the theory of linear arithmetic ($\mathcal{LA}$) over the rationals ($\mathcal{LA}(\mathbb{R})$) and over the integers ($\mathcal{LA}(\mathbb{Z})$) [12, 3], and a basic procedure for a fragment of the theory of bit vectors ($\mathcal{BV}$). [2] We are currently on the way of integrating in MATHSAT 4 also the $\mathcal{T}$-solver for the theory of unbounded reachability ($\mathcal{HMP}$) of [15].

A $\mathcal{T}$-solver gets in input a set of quantifier-free literals $\mu$ and checks whether $\mu$ is $\mathcal{T}$-satisfiable or not. In the first case, it also tries to perform deductions in the form $\mu' \models_{\mathcal{T}} l$, where $\mu' \subseteq \mu$ and $l$ is a literal representing a truth assignment to a

[1] For example, the formula $x = 5 \wedge f(x) < 3$ is rewritten into $f(5) < 3$.

[2] Currently, the $\mathcal{T}$-solver for bit-vectors can not be used in DTC with other theories.

not-yet-assigned atom occurring in the input formula. In both cases, the $\mathcal{T}$-solver generates a compact explanation for the conflict or for the implication, whose negation (a $\mathcal{T}$-valid clause called $\mathcal{T}$-lemma) is then used by the DPLL engine for backjumping and learning. The $\mathcal{T}$-solvers can also generate new atoms, which allows for implementing techniques like Delayed Theory Combination (DTC) [2], Splitting On Demand [1] or Dynamic Ackermann's Expansion [11].

In order to handle problems expressed in a combination $\mathcal{T}_1 \cup \mathcal{T}_2$ of theories, MATHSAT 4 applies the Delayed Theory Combination (DTC) procedure (see [2]), which is more suitable than the traditional Nelson-Oppen procedure for exploiting the synergy with the underlying DPLL engine.When $\mathcal{T}_1$ is the $\mathcal{EUF}$ theory, an alternative approach is that of reducing to a problem in $\mathcal{T}_2$ only by applying Ackermann's expansion to all the uninterpreted function symbols. In such cases, MATHSAT 4 applies a simple but effective heuristic [5] to decide whether to use DTC or Ackermann's expansion.

## 3 Novel Functionalities

MATHSAT 4 was designed primarily to be used in formal verification settings, where very often a simple "SAT/UNSAT" answer for an SMT problem is not enough, and extra information is required. Therefore, several extended functionalities are provided.

**Producing models.** For all theories and their combination, when $\varphi$ is satisfiable, MATHSAT 4 returns a satisfying interpretation $\mathcal{I}$ on domain variables with a congruent partial interpretation of uninterpreted functions and predicates. [3]

**Extracting $\mathcal{T}$-unsatisfiable cores.** MATHSAT 4 provides two distinct techniques for extracting a $\mathcal{T}$-unsatisfiable subset of an input clause set (unsat core) described in [8]. The first ("proof-based") computes a resolution proof of $\mathcal{T}$-unsatisfiability and returns all its leaf clauses which are not $\mathcal{T}$-lemmas. The second ("lemma-lifting") invokes an external Boolean unsat-core extractor on the Boolean abstraction of the original clauses plus all $\mathcal{T}$-lemmas computed, discharging all $\mathcal{T}$-lemmas from the result. This benefits from every size-reduction techniques implemented in Boolean unsat-core extractors available off-the-shelf.

**Computing Craig interpolants.** MATHSAT 4 allows for computing Craig interpolants of pairs of input SMT formulas [9]. This feature include an optimized interpolant generator for the full theory $\mathcal{LA}(\mathbb{R})$, an ad hoc interpolant generator for $\mathcal{DL}$, and an interpolant generator for combined theories based on DTC.

**Working incrementally.** MATHSAT 4 can work incrementally, that is, when invoked in sequence on similar SMT formulas, it reuses information from one run to the other so that to avoid restarting the search from scratch. This feature is very important when extending to SMT SAT-based techniques like BMC or induction-based model checking.

**Enumerating all consistent assignments.** MATHSAT 4 implements an "All-SMT" functionality: in case of a $\mathcal{T}$-satisfiable input formula $\varphi$, it can enumerate

[3] E.g., in $\mathcal{EUF} \cup \mathcal{LA}$, if $\varphi$ is $x = 5 \wedge f(x) < 3$, then $\mathcal{I}$ may assign $x$ to 5 and $f(5)$ to 2.

a complete set of partial assignments satisfying $\varphi$ which are consistent with the theory $\mathcal{T}$; this feature is useful for performing predicate abstraction in a SMT-based Counter-Example-Guided Abstraction-Refinement (CEGAR) context [6].
**Externalizing the control of Boolean search.** By means of the TCC interface, MATHSAT 4 allows for an external control of the variable selection during the Boolean search in DPLL. E.g, this has been used in [6] to allow an external OBDD package for driving the enumeration of $\mathcal{T}$-consistent assignments.

## References


1. C. Barrett, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Splitting on Demand in SAT Modulo Theories. In *LPAR*, volume 4246 of *LNCS*. Springer, 2006.
2. M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Ranise, and R. Sebastiani. Efficient Theory Combination via Boolean Search. *Information and Computation*, 204(10), 2006.
3. M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani. MathSAT: Tight Integration of SAT and Mathematical Decision Procedures. *Journal of Automated Reasoning*, 35(1-3), 2005.
4. R. Bruttomesso, A. Cimatti, A. Franzen, A. Griggio, Z. Hanna, A. Nadel, A. Palti, and R. Sebastiani. A Lazy and Layered SMT(BV) Solver for Hard Industrial Verification Problems. In *Proc. CAV'07*, volume 4590 of *LNCS*. Springer, 2007.
5. R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, A. Santuari, and R. Sebastiani. To Ackermann-ize or Not to Ackermann-ize? On Efficiently Handling Uninterpreted Function Symbols in SMT($\mathcal{EUF} \cup \mathcal{T}$). In *LPAR*, volume 4246 of *LNCS*. Springer, 2006.
6. R. Cavada, A. Cimatti, A. Franzén, K. Kalyanasundaram, M. Roveri, and R. Shyamasundar. Computing Predicate Abstractions by Integrating BDDs and SMT Solvers. In *FMCAD*. IEEE Computer Society, 2007.
7. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *CAV*, volume 2404 of *LNCS*. Springer, 2002.
8. A. Cimatti, A. Griggio, and R. Sebastiani. A Simple and Flexible Way of Computing Small Unsatisfiable Cores in SAT Modulo Theories. In *SAT*, volume 4501 of *LNCS*. Springer, 2007.
9. A. Cimatti, A. Griggio, and R. Sebastiani. Efficient Interpolant Generation in Satisfiability Modulo Theories. In *TACAS*, volume 4963 of *LNCS*. Springer, 2008.
10. S. Cotton and O. Maler. Fast and Flexible Difference Constraint Propagation for DPLL(T). In *SAT*, volume 4121 of *LNCS*, 2006.
11. L. de Moura and N. Bjorner. Model-based theory combination. In *SMT'07*, 2007.
12. B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *CAV*, volume 4144 of *LNCS*. Springer, 2006.
13. K. McMillan. An interpolating theorem prover. *Theor. Comp. Sci.*, 345(1), 2005.
14. R. Nieuwenhuis and A. Oliveras. Proof-producing congruence closure. In *RTA*, volume 3467 of *LNCS*. Springer, 2005.
15. Z. Rakamarić, R. Bruttomesso, A. J. Hu, and A. Cimatti. Verifying Heap Manipulating Programs in an SMT Framework. In *ATVA*, volume 4762 of *LNCS*. Springer, 2007.
16. R. Sebastiani. Lazy Satisfiability Modulo Theories. *Journal on Satisfiability, Boolean Modeling and Computation, JSAT.*, 3:141–224, 2007.