Satisfiability Modulo Theories

Silvio Ranise* Cesare Tinelli†

Many applications of formal methods rely on generating formulas of First-Order Logic (FOL) and proving or disproving their validity. Despite the great progress in the last twenty years in automated theorem proving (and disproving) in FOL, general-purpose theorem provers, such as for instance provers based on the resolution calculus, are typically inadequate to work with the sort of formulas generated by formal methods tools. The main reason is that these tools are not interested in validity in general but in validity with respect to some *background theory*, a logical theory that fixes the interpretations of certain predicates and function symbols. For instance, in formal methods involving the integers, one is only interested in showing that the formula

$$\forall x\ \forall y\ (x < y\ \Rightarrow\ x < y + y)$$

is true in those interpretations in which the symbol $<$ denotes the usual ordering over the integers and $+$ denotes the addition function. When proving the (in)validity of a formula, general-purpose reasoning methods have only one way to consider only the interpretations allowed by a background theory: add as a premise to the formula a conjunction of the theory's axioms. When this is possible at all[1] the performance of generic theorem provers is usually unacceptable for realistic formal method applications. A more viable alternative is the use of specialized reasoning methods for the background theory of interest. This is particularly the case for *ground* formulas, FOL formulas with no variables (and so also with no quantifiers), but possibly with *free constants*—constant symbols not in the background theory.

For many theories, specialized methods actually yield *decision procedures* for the validity of ground formulas or some subset of them. This is for instance the case, thanks to classical results in mathematics, for the theory of integer numbers (and formulas with no multiplication symbols) or the theory of real numbers. In the last two decades however, specialized decision

*LORIA & INRIA-Lorraine and Università degli Studi di Milano
†University of Iowa
[1]Some background theories cannot be captured by a finite set of FOL formulas.

procedures have also been discovered for a long, and still growing, list of theories of other important data types such as certain theories of arrays and of strings, several variants of the theory of finite sets, some theories of lattices, the theories of finite, regular and infinite trees, of lists, tuples, records, queues, hash tables, and bit vectors of a fixed or arbitrary finite size.

The literature on these procedures often describes them in terms of *satisfiability* in a theory—relying on the fact that a formula is valid in a theory T exactly when no interpretation of T satisfies the formula's negation. For this reason we refer to the field as *Satisfiability Modulo Theories* or SMT, for short, and call those procedures *SMT solvers*.

The use of SMT solvers in formal methods is not new. It was championed in the early 1980s by Greg Nelson and Derek Oppen at Stanford University, by Robert Shostak at SRI, and by Robert Boyer and J Moore at the University of Texas at Austin. Building on this work, however, the last ten years have seen an explosion of interest and research on the foundations and practical aspects of SMT. Several SMT solvers have been developed in academia and industry with continually increasing scope and performance. Some of them have or are being integrated into: interactive theorem provers for high-order logic (such as HOL and Isabelle); extended static checkers (such as CAsCaDE, Boogie, and ESC/Java); verification systems (such as ACL2, Caduceus, SAL, and UCLID); formal CASE environments (such as KeY); model checkers (such as BLAST, MAGIC and SLAM); certifying compilers (such as Touchstone); unit test generators (such as CUTE and MUTT). In industry, there are currently SMT-related research projects at Cadence, Intel, Microsoft, and NEC, just to name some.

Main SMT Approaches

The design, proof of correctness, and implementation of SMT methods poses several challenges. First, formal methods naturally involve more than one data type, each with its own background theory, so suitable combination techniques are necessary. Second, satisfiability procedures must be proved sound and complete. While soundness is usually easy, completeness requires specific model construction arguments showing that, whenever the procedure finds a formula satisfiable, a satisfying theory interpretation for it does indeed exist. This means that each new procedure in principle requires a new completeness proof. Third, data structures and algorithms for a new procedure, precisely for being specialized, are often implemented from scratch,

with little software reuse. There are currently three major approaches for implementing SMT solvers, each addressing the challenges above differently and with their own pros and cons.

The SAT Encodings Approach

This approach is based on ad-hoc translations that convert an input formula and relevant consequences of its background theory into an equisatisfiable propositional formula (see for instance [4]). The approach applies in principle to all theories whose ground satisfiability problem is decidable, possibly however at the cost of an exponential blow-up in the translation. The approach is nevertheless appealing because SAT solvers today are able to quickly process extremely large formulas. Proving soundness and completeness is relatively simple because it reduces to proving that the translation is satisfiability preserving. Also, the implementation effort is relatively small for being limited to the translator—after that, one can use any off-the-shelf SAT solver. Eager versions of the approach, which first generate the complete translation and then pass it to a SAT solver, have produced in the recent past very competitive solvers for the theory of equality and for certain fragments of the integers theory. The main disadvantage of the SAT encodings developed so far is that they do not scale up as well as SMT solvers based on the next approach because of the exponential blowup of the eager translation and the difficulty of combining encodings for different theories.

The Small Engines Approach

This approach is currently the most popular and consists in building procedures implementing an inference systems specialized on a theory T. The lure of these "small engines" is that one can use whatever algorithms and data structures are best for T, which typically leads to better performance. A disadvantage is that proving the correctness of an ad-hoc procedure may be non-trivial. A possibly bigger disadvantage is that one has to write an entire solver for each new theory of interest, possibly duplicating internal functionalities and implementation effort.

One way to address the latter problem is to reduce a theory solver to its essence by separating generic Boolean reasoning from theory reasoning proper. The common practice is to write theory solvers just for conjunctions of ground *literals*—atomic formulas and negation. These pared down solvers are then embedded as separate submodules into an efficient SAT

solver, allowing the joint system to accept arbitrary ground formulas. Such a scheme (formulated generally and abstractly in [2]) allows one to plug-in a new theory solver into the same SAT engine as long as the solver conforms to a simple common interface.

Another way to reduce development costs is to decompose, when possible, a background theory into two or more component theories, write a solver for each of the smaller theories, and then use the solvers cooperatively. A general and highly influential method for doing this is due to Nelson and Oppen. Several enhancements and variations on this methods are used today in all major SMT solvers based of the small engines approach.

The Big Engines Approach

This is a rather recent approach that can be applied to theories T that admit a finite FOL axiomatization, capitalizing on the power and flexibility of current automated theorem provers for FOL—in particular, provers based on the *superposition calculus*, a modern version of resolution with a built-in treatment of the equality predicate and powerful techniques for reducing the search space. The approach consists in instrumenting a superposition prover with specialized control strategies which, together with the axioms of T, effectively turn this "big engine" into a decision procedure for ground satisfiability in T. A big plus of the approach is a simplified proof of correctness, which reduces down to a routine termination proof for the rules of the superposition calculus (see, e.g., [1] for details). Another advantage is that, when the approach applies to two theories, obtaining a decision procedure for their union is, under reasonable assumptions, as simple as feeding the union of the axioms to the prover. A further advantage is the reuse of efficient data structures and algorithms for automated deduction implemented in state-of-the-art provers. The main disadvantage is that to get additional functionalities (such as incrementality, proof production, or model building), one may need to modify the prover in ways that the original implementors did not foresee, which may require a considerable (re)implementation effort.

Standardization Efforts

Because of their specialized nature, it is very common for different SMT solvers to be based on different variants of FOL, work with different theories, deal with different classes of formulas, and have different interfaces and input formats. Until a few years ago this made it arduous to assess the relative

merits of existing SMT solvers and techniques theoretically or in practice. In fact, it was difficult even to test and evaluate a single solver in isolation because of the dearth of available benchmarks.

To mitigate these problems the SMT community launched in 2002 the *SMT-LIB initiative*, a standardization effort co-led by these authors and currently backed by the vast majority of research groups in SMT. The initiative's main goals are to define standard input/output formats and interfaces for SMT solvers, and build a large on-line repository of benchmarks for several theories. The current version of the input format is now supported by several SMT solvers worldwide and is being adopted by a number of formal methods applications. The repository, which is still growing, presently includes about 40,000 benchmarks from academia and industry. (See [3] for more details on SMT-LIB and on SMT-COMP, the affiliated solver competition.)

Future Directions

While SMT tools are proving increasingly useful in formal methods applications, more work is needed to improve the trade-off between their efficiency and their expressiveness. For example, software verification problems often require the handling of formulas with quantifiers, something that SMT solvers do not do satisfactorily yet. Finding good heuristics for lifting current SMT techniques from ground to quantified formulas is a key challenge for the future.

Other lines of further work come from the need to enhance further the interface functionalities of SMT solvers. For instance, to validate the results of a solver or integrate it within interactive provers, it is necessary to have solvers that can produce a machine-checkable proof every time they declare a formula to be unsatisfiable. Similarly, to be useful to larger tools like static checker, model checkers and test set generators, an SMT solver must be able to produce, for formula it finds satisfiable, a concrete and finite representation of the interpretation that satisfies it. Other potentially very useful functionalities are the generation of unsatisfiable cores of unsatisfiable formulas, or of interpolants for pairs of jointly unsatisfiable formulas. More research on efficiently realizing all these functionalities is currently under way.

References

[1] Alessandro Armando, Silvio Ranise, and Michäel Rusinowitch. A Rewriting Approach to Satisfiability Procedures. *Information and Computation*, 183(2):140–164, June 2003.

[2] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Abstract DPLL and Abstract DPLL Modulo Theories. In F. Baader and A. Voronkov, editors, *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'04)*, volume 3452 of *Lecture Notes in Computer Science*, pages 36–50. Springer, 2005.

[3] Silvio Ranise and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2006.

[4] Ofer Strichman, Sanjit A. Seshia, and Randal E. Bryant. Deciding separation formulas with SAT. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV'02)*, Lecture Notes in Computer Science, pages 209–222. Springer, 2002.