Satisfiability modulo theories

In computer science and mathematical logic, **satisfiability modulo theories** (**SMT**) is the problem of determining whether a mathematical formula is satisfiable. It generalizes the Boolean satisfiability problem (SAT) to more complex formulas involving real numbers, integers, and/or various data structures such as lists, arrays, bit vectors, and strings. The name is derived from the fact that these expressions are interpreted within ("modulo") a certain formal theory in first-order logic with equality (often disallowing quantifiers). **SMT solvers** are tools that aim to solve the SMT problem for a practical subset of inputs. SMT solvers such as Z3 and cvc5 have been used as a building block for a wide range of applications across computer science, including in automated theorem proving, program analysis, program verification, and software testing.

Since Boolean satisfiability is already NP-complete, the SMT problem is typically NP-hard, and for many theories it is undecidable. Researchers study which theories or subsets of theories lead to a decidable SMT problem and the computational complexity of decidable cases. The resulting decision procedures are often implemented directly in SMT solvers; see, for instance, the decidability of Presburger arithmetic. SMT can be thought of as a constraint satisfaction problem and thus a certain formalized approach to constraint programming.

Terminology and examples

Formally speaking, an SMT instance is a formula in first-order logic, where some function and predicate symbols have additional interpretations, and SMT is the problem of determining whether such a formula is satisfiable. In other words, imagine an instance of the Boolean satisfiability problem (SAT) in which some of the binary variables are replaced by predicates over a suitable set of non-binary variables. A predicate is a binary-valued function of non-binary variables. Example predicates include linear inequalities (e.g., $3x + 2y - z \geq 4$) or equalities involving uninterpreted terms and function symbols (e.g., $f(f(u, v), v) = f(u, v)$ where f is some unspecified function of two arguments). These predicates are classified according to each respective theory assigned. For instance, linear inequalities over real variables are evaluated using the rules of the theory of linear real arithmetic, whereas predicates involving uninterpreted terms and function symbols are evaluated using the rules of the theory of uninterpreted functions with equality (sometimes referred to as the empty theory). Other theories include the theories of arrays and list structures (useful for modeling and verifying computer programs), and the theory of bit vectors (useful in modeling and verifying hardware designs). Subtheories are also possible: for example, difference logic is a sub-theory of linear arithmetic in which each inequality is restricted to have the form $x - y > c$ for variables x and y and constant c.

The examples above show the use of Linear Integer Arithmetic over inequalities. Other examples include:

- Satisfiability: Determine if $x \vee (y \wedge \neg z)$ is satisfiable.
- Array access: Find a value for array A such that A[0]=5.
- Bit vector arithmetic: Determine if *x* and *y* are distinct 3-bit numbers.
- Uninterpreted functions: Find values for *x* and *y* such that $f(x) = 2$ and $g(x) = 3$.

Most SMT solvers support only quantifier-free fragments of their logics.

Relationship to automated theorem proving

There is substantial overlap between SMT solving and automated theorem proving (ATP). Generally, automated theorem provers focus on supporting full first-order logic with quantifiers, whereas SMT solvers focus more on supporting various theories (interpreted predicate symbols). ATPs excel at problems with lots of quantifiers, whereas

SMT solvers do well on large problems without quantifiers.[1] The line is blurry enough that some ATPs participate in SMT-COMP, while some SMT solvers participate in CASC.[2]

Expressive power

An SMT instance is a generalization of a Boolean SAT instance in which various sets of variables are replaced by predicates from a variety of underlying theories. SMT formulas provide a much richer modeling language than is possible with Boolean SAT formulas. For example, an SMT formula allows one to model the datapath operations of a microprocessor at the word rather than the bit level.

By comparison, answer set programming is also based on predicates (more precisely, on atomic sentences created from atomic formulas). Unlike SMT, answer-set programs do not have quantifiers, and cannot easily express constraints such as linear arithmetic or difference logic—answer set programming is best suited to Boolean problems that reduce to the free theory of uninterpreted functions. Implementing 32-bit integers as bitvectors in answer set programming suffers from most of the same problems that early SMT solvers faced: "obvious" identities such as $x+y=y+x$ are difficult to deduce.

Constraint logic programming does provide support for linear arithmetic constraints, but within a completely different theoretical framework. SMT solvers have also been extended to solve formulas in higher-order logic.[3]

Solver approaches

Early attempts for solving SMT instances involved translating them to Boolean SAT instances (e.g., a 32-bit integer variable would be encoded by 32 single-bit variables with appropriate weights and word-level operations such as 'plus' would be replaced by lower-level logic operations on the bits) and passing this formula to a Boolean SAT solver. This approach, which is referred to as *the eager approach* (or *bitblasting*), has its merits: by pre-processing the SMT formula into an equivalent Boolean SAT formula existing Boolean SAT solvers can be used "as-is" and their performance and capacity improvements leveraged over time. On the other hand, the loss of the high-level semantics of the underlying theories means that the Boolean SAT solver has to work a lot harder than necessary to discover "obvious" facts (such as $x + y = y + x$ for integer addition.) This observation led to the development of a number of SMT solvers that tightly integrate the Boolean reasoning of a DPLL-style search with theory-specific solvers (*T-solvers*) that handle conjunctions (ANDs) of predicates from a given theory. This approach is referred to as *the lazy approach*.[4]

Dubbed DPLL(T),[5] this architecture gives the responsibility of Boolean reasoning to the DPLL-based SAT solver which, in turn, interacts with a solver for theory T through a well-defined interface. The theory solver only needs to worry about checking the feasibility of conjunctions of theory predicates passed on to it from the SAT solver as it explores the Boolean search space of the formula. For this integration to work well, however, the theory solver must be able to participate in propagation and conflict analysis, i.e., it must be able to infer new facts from already established facts, as well as to supply succinct explanations of infeasibility when theory conflicts arise. In other words, the theory solver must be incremental and backtrackable.

Decidable theories

Researchers study which theories or subsets of theories lead to a decidable SMT problem and the computational complexity of decidable cases. Since full first-order logic is only semidecidable, one line of research attempts to find efficient decision procedures for fragments of first-order logic such as effectively propositional logic.[6]

Another line of research involves the development of specialized decidable theories, including linear arithmetic over rationals and integers, fixed-width bitvectors,[7] floating-point arithmetic (often implemented in SMT solvers via *bit-blasting*, i.e., reduction to bitvectors),[8][9] strings,[10] (co)-datatypes,[11] sequences (used to model dynamic arrays),[12] finite sets and relations,[13][14] separation logic,[15] finite fields,[16] and uninterpreted functions among others.

Boolean monotonic theories are a class of theory that support efficient theory propagation and conflict analysis, enabling practical use within DPLL(T) solvers.[17] Monotonic theories support only boolean variables (boolean is the only *sort*), and all their functions and predicates p obey the axiom

$$p(\ldots, b_{i-1}, 0, b_{i+1}, \ldots) \implies p(\ldots, b_{i-1}, 1, b_{i+1}, \ldots)$$

Examples of monotonic theories include graph reachability, collision detection for convex hulls, minimum cuts, and computation tree logic.[18] Every Datalog program can be interpreted as a monotonic theory.[19]

SMT for undecidable theories

Most of the common SMT approaches support decidable theories. However, many real-world systems, such as an aircraft and its behavior, can only be modelled by means of non-linear arithmetic over the real numbers involving transcendental functions. This fact motivates an extension of the SMT problem to non-linear theories, such as determining whether the following equation is satisfiable:

$$(\sin(x)^3 = \cos(\log(y) \cdot x) \vee b \vee -x^2 \geq 2.3y) \wedge \left(\neg b \vee y < -34.4 \vee \exp(x) > \tfrac{y}{x}\right)$$

where

$$b \in \mathbb{B}, x, y \in \mathbb{R}.$$

Such problems are, however, undecidable in general. (On the other hand, the theory of real closed fields, and thus the full first order theory of the real numbers, are decidable using quantifier elimination. This is due to Alfred Tarski.) The first order theory of the natural numbers with addition (but not multiplication), called Presburger arithmetic, is also decidable. Since multiplication by constants can be implemented as nested additions, the arithmetic in many computer programs can be expressed using Presburger arithmetic, resulting in decidable formulas.

Examples of SMT solvers addressing Boolean combinations of theory atoms from undecidable arithmetic theories over the reals are ABsolver,[20] which employs a classical DPLL(T) architecture with a non-linear optimization packet as (necessarily incomplete) subordinate theory solver, iSAT (http://isat.gforge.avacs.org/), building on a unification of DPLL SAT-solving and interval constraint propagation called the iSAT algorithm,[21] and cvc5.[22]

Solvers

The table below summarizes some of the features of the many available SMT solvers. The column "SMT-LIB" indicates compatibility with the SMT-LIB language; many systems marked 'yes' may support only older versions of SMT-LIB, or offer only partial support for the language. The column "CVC" indicates support for the CVC language. The column "DIMACS" indicates support for the DIMACS format (http://www.satcompetition.org/2009/format-benchmarks2009.html).

Projects differ not only in features and performance, but also in the viability of the surrounding community, its ongoing interest in a project, and its ability to contribute documentation, fixes, tests and enhancements.

Platform			Features						Notes
Name	**OS**	**License**	**SMT-LIB**	**CVC**	**DIMACS**	**Built-in theories**	**API**	**SMT-COMP [1] (http://www.smtcomp.org/)**	
ABsolver	Linux	CPL	v1.2	No	Yes	linear arithmetic, non-linear arithmetic	C++	no	DPLL-based
Alt-Ergo	Linux, Mac OS, Windows	CeCILL-C (roughly equivalent to LGPL)	partial v1.2 and v2.0	No	No	empty theory, linear integer and rational arithmetic, non-linear arithmetic, polymorphic arrays, enumerated datatypes, AC symbols, bitvectors, record datatypes, quantifiers	OCaml	2008	Polymorphic first-order input language à la ML, SAT-solver based, combines Shostak-like and Nelson-Oppen like approaches for reasoning modulo theories
Barcelogic	Linux	Proprietary	v1.2			empty theory, difference logic	C++	2009	DPLL-based, congruence closure
Beaver	Linux, Windows	BSD	v1.2	No	No	bitvectors	OCaml	2009	SAT-solver based
Boolector	Linux	MIT	v1.2	No	No	bitvectors, arrays	C	2009	SAT-solver based
CVC3	Linux	BSD	v1.2	Yes		empty theory, linear arithmetic, arrays, tuples, types, records, bitvectors, quantifiers	C/C++	2010	proof output to HOL
CVC4	Linux, Mac OS, Windows, FreeBSD	BSD	Yes	Yes		rational and integer linear arithmetic, arrays, tuples, records, inductive data types, bitvectors, strings, and equality over uninterpreted function symbols	C++	2021	version 1.8 released May 2021
cvc5	Linux, Mac OS, Windows	BSD	Yes	Yes		rational and integer linear arithmetic, arrays, tuples, records, inductive	C++, Python, Java	2021	version 1.0 released April 2022

						data types, bitvectors, strings, sequences, bags, and equality over uninterpreted function symbols			
Decision Procedure Toolkit (DPT)	Linux	Apache	No				OCaml	no	DPLL-based
iSAT	Linux	Proprietary	No			non-linear arithmetic		no	DPLL-based
MathSAT	Linux, Mac OS, Windows	Proprietary	Yes		Yes	empty theory, linear arithmetic, nonlinear arithmetic, bitvectors, arrays	C/C++, Python, Java	2010	DPLL-based
MiniSmt	Linux	LGPL	partial v2.0			non-linear arithmetic	OCaml	2010	SAT-solver based, Yices-based
Norn									SMT solver for string constraints
OpenCog	Linux	AGPL	No	No	No	probabilistic logic, arithmetic. relational models	C++, Scheme, Python	no	subgraph isomorphism
OpenSMT	Linux, Mac OS, Windows	GPLv3	partial v2.0		Yes	empty theory, differences, linear arithmetic, bitvectors	C++	2011	lazy SMT Solver
raSAT	Linux	GPLv3	v2.0			real and integer nonlinear arithmetic		2014, 2015	extension of the Interval Constraint Propagation with Testing and the Intermediate Value Theorem
SatEEn	?	Proprietary	v1.2			linear arithmetic, difference logic	none	2009	
SMTInterpol	Linux, Mac OS, Windows	LGPLv3	v2.5			uninterpreted functions, linear real arithmetic, and linear integer arithmetic	Java	2012	Focuses on generating high quality, compact interpolants.
SMCHR	Linux, Mac OS, Windows	GPLv3	No	No	No	linear arithmetic, nonlinear arithmetic, heaps	C	no	Can implement new theories using Constraint Handling Rules.
SMT-RAT	Linux, Mac OS	MIT	v2.0	No	No	linear arithmetic,	C++	2015	Toolbox for strategic and parallel SMT

						nonlinear arithmetic			solving consisting of a collection of SMT compliant implementations.
SONOLAR	Linux, Windows	Proprietary	partial v2.0			bitvectors	C	2010	SAT-solver based
Spear	Linux, Mac OS, Windows	Proprietary	v1.2			bitvectors		2008	
STP	Linux, OpenBSD, Windows, Mac OS	MIT	partial v2.0	Yes	No	bitvectors, arrays	C, C++, Python, OCaml, Java	2011	SAT-solver based
SWORD	Linux	Proprietary	v1.2			bitvectors		2009	
UCLID	Linux	BSD	No	No	No	empty theory, linear arithmetic, bitvectors, and constrained lambda (arrays, memories, cache, etc.)		no	SAT-solver based, written in Moscow ML. Input language is SMV model checker. Well-documented!
veriT	Linux, OS X	BSD	partial v2.0			empty theory, rational and integer linear arithmetics, quantifiers, and equality over uninterpreted function symbols	C/C++	2010	SAT-solver based, can produce proofs
Yices	Linux, Mac OS, Windows, FreeBSD	GPLv3	v2.0	No	Yes	rational and integer linear arithmetic, bitvectors, arrays, and equality over uninterpreted function symbols	C	2014	Source code is available online
Z3 Theorem Prover	Linux, Mac OS, Windows, FreeBSD	MIT	v2.0		Yes	empty theory, linear arithmetic, nonlinear arithmetic, bitvectors, arrays, datatypes, quantifiers, strings	C/C++, .NET, OCaml, Python, Java, Haskell	2011	Source code is available online

Standardization and the SMT-COMP solver competition

There are multiple attempts to describe a standardized interface to SMT solvers (and automated theorem provers, a term often used synonymously). The most prominent is the SMT-LIB standard, which provides a language based on S-expressions. Other standardized formats commonly supported are the DIMACS format supported by many Boolean SAT solvers, and the CVC format used by the CVC automated theorem prover.

The SMT-LIB format also comes with a number of standardized benchmarks and has enabled a yearly competition between SMT solvers called SMT-COMP. Initially, the competition took place during the Computer Aided Verification conference (CAV),[23][24] but as of 2020 the competition is hosted as part of the SMT Workshop, which is affiliated with the International Joint Conference on Automated Reasoning (IJCAR).[25]

Applications

SMT solvers are useful both for verification, proving the correctness of programs, software testing based on symbolic execution, and for synthesis, generating program fragments by searching over the space of possible programs. Outside of software verification, SMT solvers have also been used for type inference[26][27] and for modelling theoretic scenarios, including modelling actor beliefs in nuclear arms control.[28]

Verification

Computer-aided verification of computer programs often uses SMT solvers. A common technique is to translate preconditions, postconditions, loop conditions, and assertions into SMT formulas in order to determine if all properties can hold.

There are many verifiers built on top of the Z3 SMT solver. Boogie (http://research.microsoft.com/en-us/projects/boogie/) is an intermediate verification language that uses Z3 to automatically check simple imperative programs. The VCC (https://www.microsoft.com/en-us/research/project/vcc-a-verifier-for-concurrent-c/) verifier for concurrent C uses Boogie, as well as Dafny (http://research.microsoft.com/en-us/projects/dafny/) for imperative object-based programs, Chalice (http://research.microsoft.com/en-us/projects/chalice/) for concurrent programs, and Spec# (http://research.microsoft.com/en-us/projects/specsharp/) for C#. F* (http://research.microsoft.com/en-us/projects/fstar/) is a dependently typed language that uses Z3 to find proofs; the compiler carries these proofs through to produce proof-carrying bytecode. The Viper verification infrastructure (http://viper.ethz.ch) encodes verification conditions to Z3. The sbv (https://hackage.haskell.org/package/sbv) library provides SMT-based verification of Haskell programs, and lets the user choose among a number of solvers such as Z3, ABC, Boolector, cvc5, MathSAT and Yices.

There are also many verifiers built on top of the Alt-Ergo (http://alt-ergo.ocamlpro.com/) SMT solver. Here is a list of mature applications:

- Why3 (http://why3.lri.fr/), a platform for deductive program verification, uses Alt-Ergo as its main prover;
- CAVEAT, a C-verifier developed by CEA and used by Airbus; Alt-Ergo was included in the qualification DO-178C of one of its recent aircraft;
- Frama-C, a framework to analyse C-code, uses Alt-Ergo in the Jessie and WP plugins (dedicated to "deductive program verification");
- SPARK uses CVC4 and Alt-Ergo (behind GNATprove) to automate the verification of some assertions in SPARK 2014;
- Atelier-B can use Alt-Ergo instead of its main prover (increasing success from 84% to 98% on the ANR Bware project benchmarks (http://alt-ergo.lri.fr/documents/ABZ-2014.pdf));
- Rodin, a B-method framework developed by Systerel, can use Alt-Ergo as a back-end;
- Cubicle (http://cubicle.lri.fr/), an open source model checker for verifying safety properties of array-based transition systems.
- EasyCrypt (https://www.easycrypt.info/), a toolset for reasoning about relational properties of probabilistic computations with adversarial code.

Many SMT solvers implement a common interface format called SMTLIB2 (http://smt-lib.org/) (such files usually have the extension "`.smt2`"). The LiquidHaskell (https://ucsd-progsys.github.io/liquidhaskell-blog/) tool implements a refinement type based verifier for Haskell that can use any SMTLIB2 compliant solver, e.g. cvc5, MathSat, or Z3.

Symbolic-execution based analysis and testing

An important application of SMT solvers is symbolic execution for analysis and testing of programs (e.g., concolic testing), aimed particularly at finding security vulnerabilities. Example tools in this category include SAGE (http://research.microsoft.com/en-us/um/people/pg/public_psfiles/ndss2008.pdf) from Microsoft Research, KLEE (https://klee.github.io/), S2E (http://s2e.epfl.ch/), and Triton (https://triton.quarkslab.com). SMT solvers that have been used for symbolic-execution applications include Z3 (https://github.com/Z3Prover/z3), STP (https://sites.google.com/site/stpfastprover/) Archived (https://web.archive.org/web/20150406115407/https://sites.google.com/site/stpfastprover/) 2015-04-06 at the Wayback Machine, the Z3str family of solvers (https://z3string.github.io/), and Boolector (http://fmv.jku.at/boolector/).

Interactive theorem proving

SMT solvers have been integrated with proof assistants, including Coq[29] and Isabelle/HOL.[30]

See also

- Answer set programming
- Automated theorem proving
- SAT solver
- First-order logic
- Theory of pure equality

Notes

1. Blanchette, Jasmin Christian; Böhme, Sascha; Paulson, Lawrence C. (2013-06-01). "Extending Sledgehammer with SMT Solvers" (https://doi.org/10.1007/s10817-013-9278-5). *Journal of Automated Reasoning*. **51** (1): 109–128. doi:10.1007/s10817-013-9278-5 (https://doi.org/10.1007%2Fs10817-013-9278-5). ISSN 1573-0670 (https://search.worldcat.org/issn/1573-0670). "ATPs and SMT solvers have complementary strengths. The former handle quantifiers more elegantly, whereas the latter excel on large, mostly ground problems."
2. Weber, Tjark; Conchon, Sylvain; Déharbe, David; Heizmann, Matthias; Niemetz, Aina; Reger, Giles (2019-01-01). "The SMT Competition 2015–2018" (https://doi.org/10.3233%2FSAT190123). *Journal on Satisfiability, Boolean Modeling and Computation*. **11** (1): 221–259. doi:10.3233/SAT190123 (https://doi.org/10.3233%2FSAT190123). S2CID 210147712 (https://api.semanticscholar.org/CorpusID:210147712). "In recent years, we have seen a blurring of lines between SMT-COMP and CASC with SMT solvers competing in CASC and ATPs competing in SMT-COMP."
3. Barbosa, Haniel; Reynolds, Andrew; El Ouraoui, Daniel; Tinelli, Cesare; Barrett, Clark (2019). "Extending SMT solvers to higher-order logic" (https://hal.archives-ouvertes.fr/hal-02300986/document). *Automated Deduction – CADE 27: 27th International Conference on Automated Deduction, Natal, Brazil, August 27–30, 2019, Proceedings*. Springer. pp. 35–54. doi:10.1007/978-3-030-29436-6_3 (https://doi.org/10.1007%2F978-3-030-29436-6_3). ISBN 978-3-030-29436-6. S2CID 85443815 (https://api.semanticscholar.org/CorpusID:85443815). hal-02300986.

4. Bruttomesso, Roberto; Cimatti, Alessandro; Franzén, Anders; Griggio, Alberto; Hanna, Ziyad; Nadel, Alexander; Palti, Amit; Sebastiani, Roberto (2007). "A Lazy and Layered SMT($\mathcal{BV}$) Solver for Hard Industrial Verification Problems" (https://link.springer.com/chapter/10.1007/978-3-540-73368-3_54). In Damm, Werner; Hermanns, Holger (eds.). *Computer Aided Verification*. Lecture Notes in Computer Science. Vol. 4590. Berlin, Heidelberg: Springer. pp. 547–560. doi:10.1007/978-3-540-73368-3_54 (https://doi.org/10.1007%2F978-3-540-73368-3_54). ISBN 978-3-540-73368-3.
5. Nieuwenhuis, R.; Oliveras, A.; Tinelli, C. (2006), "Solving SAT and SAT Modulo Theories: From an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T)" (http://homepage.cs.uiowa.edu/~tinelli/papers/NieOT-JACM-06.pdf) (PDF), *Journal of the ACM*, vol. 53, pp. 937–977, doi:10.1145/1217856.1217859 (https://doi.org/10.1145%2F1217856.1217859), S2CID 14058631 (https://api.semanticscholar.org/CorpusID:14058631)
6. de Moura, Leonardo; Bjørner, Nikolaj (August 12–15, 2008). "Deciding Effectively Propositional Logic Using DPLL and Substitution Sets" (https://link.springer.com/chapter/10.1007/978-3-540-71070-7_35). In Armando, Alessandro; Baumgartner, Peter; Dowek, Gilles (eds.). *Automated Reasoning*. 4th International Joint Conference on Automated Reasoning, Sydney, NSW, Australia. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. pp. 410–425. doi:10.1007/978-3-540-71070-7_35 (https://doi.org/10.1007%2F978-3-540-71070-7_35). ISBN 978-3-540-71070-7.
7. Hadarean, Liana; Bansal, Kshitij; Jovanović, Dejan; Barrett, Clark; Tinelli, Cesare (2014). "A Tale of Two Solvers: Eager and Lazy Approaches to Bit-Vectors" (https://link.springer.com/chapter/10.1007/978-3-319-08867-9_45). In Biere, Armin; Bloem, Roderick (eds.). *Computer Aided Verification*. Lecture Notes in Computer Science. Vol. 8559. Cham: Springer International Publishing. pp. 680–695. doi:10.1007/978-3-319-08867-9_45 (https://doi.org/10.1007%2F978-3-319-08867-9_45). ISBN 978-3-319-08867-9.
8. Brain, Martin; Schanda, Florian; Sun, Youcheng (2019). "Building Better Bit-Blasting for Floating-Point Problems". In Vojnar, Tomáš; Zhang, Lijun (eds.). *Tools and Algorithms for the Construction and Analysis of Systems*. 25th International Conference, Tools and Algorithms for the Construction and Analysis of Systems 2019, Prague, Czech Republic, April 6–11, 2019, Proceedings, Part I. Lecture Notes in Computer Science. Cham: Springer International Publishing. pp. 79–98. doi:10.1007/978-3-030-17462-0_5 (https://doi.org/10.1007%2F978-3-030-17462-0_5). ISBN 978-3-030-17462-0. S2CID 92999474 (https://api.semanticscholar.org/CorpusID:92999474).
9. Brain, Martin; Niemetz, Aina; Preiner, Mathias; Reynolds, Andrew; Barrett, Clark; Tinelli, Cesare (2019). "Invertibility Conditions for Floating-Point Formulas". In Dillig, Isil; Tasiran, Serdar (eds.). *Computer Aided Verification*. 31st International Conference, Computer Aided Verification 2019, New York City, July 15–18, 2019. Lecture Notes in Computer Science. Cham: Springer International Publishing. pp. 116–136. doi:10.1007/978-3-030-25543-5_8 (https://doi.org/10.1007%2F978-3-030-25543-5_8). ISBN 978-3-030-25543-5. S2CID 196613701 (https://api.semanticscholar.org/CorpusID:196613701).
10. Liang, Tianyi; Tsiskaridze, Nestan; Reynolds, Andrew; Tinelli, Cesare; Barrett, Clark (2015). "A Decision Procedure for Regular Membership and Length Constraints over Unbounded Strings" (https://link.springer.com/chapter/10.1007/978-3-319-24246-0_9). In Lutz, Carsten; Ranise, Silvio (eds.). *Frontiers of Combining Systems*. Lecture Notes in Computer Science. Vol. 9322. Cham: Springer International Publishing. pp. 135–150. doi:10.1007/978-3-319-24246-0_9 (https://doi.org/10.1007%2F978-3-319-24246-0_9). ISBN 978-3-319-24246-0.
11. Reynolds, Andrew; Blanchette, Jasmin Christian (2015). "A Decision Procedure for (Co)datatypes in SMT Solvers" (https://link.springer.com/chapter/10.1007/978-3-319-21401-6_13). In Felty, Amy P.; Middeldorp, Aart (eds.). *Automated Deduction - CADE-25*. Lecture Notes in Computer Science. Vol. 9195. Cham: Springer International Publishing. pp. 197–213. doi:10.1007/978-3-319-21401-6_13 (https://doi.org/10.1007%2F978-3-319-21401-6_13). ISBN 978-3-319-21401-6.
12. Sheng, Ying; Nötzli, Andres; Reynolds, Andrew; Zohar, Yoni; Dill, David; Grieskamp, Wolfgang; Park, Junkil; Qadeer, Shaz; Barrett, Clark; Tinelli, Cesare (2023-09-15). "Reasoning About Vectors: Satisfiability Modulo a Theory of Sequences" (https://doi.org/10.1007/s10817-023-09682-2). *Journal of Automated Reasoning*. **67** (3): 32. doi:10.1007/s10817-023-09682-2 (https://doi.org/10.1007%2Fs10817-023-09682-2). ISSN 1573-0670 (https://search.worldcat.org/issn/1573-0670). S2CID 261829653 (https://api.semanticscholar.org/CorpusID:261829653).
13. Bansal, Kshitij; Reynolds, Andrew; Barrett, Clark; Tinelli, Cesare (2016). "A New Decision Procedure for Finite Sets and Cardinality Constraints in SMT" (https://link.springer.com/chapter/10.1007/978-3-319-40229-1_7). In Olivetti, Nicola; Tiwari, Ashish (eds.). *Automated Reasoning*. Lecture Notes in Computer Science. Vol. 9706. Cham: Springer International Publishing. pp. 82–98. doi:10.1007/978-3-319-40229-1_7 (https://doi.org/10.1007%2F978-3-319-40229-1_7). ISBN 978-3-319-40229-1.

14. Meng, Baoluo; Reynolds, Andrew; Tinelli, Cesare; Barrett, Clark (2017). "Relational Constraint Solving in SMT" (https://link.springer.com/chapter/10.1007/978-3-319-63046-5_10). In de Moura, Leonardo (ed.). *Automated Deduction – CADE 26*. Lecture Notes in Computer Science. Vol. 10395. Cham: Springer International Publishing. pp. 148–165. doi:10.1007/978-3-319-63046-5_10 (https://doi.org/10.1007%2F978-3-319-63046-5_10). ISBN 978-3-319-63046-5.
15. Reynolds, Andrew; Iosif, Radu; Serban, Cristina; King, Tim (2016). "A Decision Procedure for Separation Logic in SMT" (https://hal.archives-ouvertes.fr/hal-01418883/file/Atva2016-2.pdf) (PDF). In Artho, Cyrille; Legay, Axel; Peled, Doron (eds.). *Automated Technology for Verification and Analysis*. Lecture Notes in Computer Science. Vol. 9938. Cham: Springer International Publishing. pp. 244–261. doi:10.1007/978-3-319-46520-3_16 (https://doi.org/10.1007%2F978-3-319-46520-3_16). ISBN 978-3-319-46520-3. S2CID 6753369 (https://api.semanticscholar.org/CorpusID:6753369).
16. Ozdemir, Alex; Kremer, Gereon; Tinelli, Cesare; Barrett, Clark (2023). "Satisfiability Modulo Finite Fields" (https://link.springer.com/chapter/10.1007/978-3-031-37703-7_8). In Enea, Constantin; Lal, Akash (eds.). *Computer Aided Verification*. Lecture Notes in Computer Science. Vol. 13965. Cham: Springer Nature Switzerland. pp. 163–186. doi:10.1007/978-3-031-37703-7_8 (https://doi.org/10.1007%2F978-3-031-37703-7_8). ISBN 978-3-031-37703-7. S2CID 257235627 (https://api.semanticscholar.org/CorpusID:257235627).
17. Bayless, Sam; Bayless, Noah; Hoos, Holger; Hu, Alan (2015-03-04). "SAT Modulo Monotonic Theories" (https://ojs.aaai.org/index.php/AAAI/article/view/9755). *Proceedings of the AAAI Conference on Artificial Intelligence*. **29** (1). arXiv:1406.0043 (https://arxiv.org/abs/1406.0043). doi:10.1609/aaai.v29i1.9755 (https://doi.org/10.1609%2Faaai.v29i1.9755). ISSN 2374-3468 (https://search.worldcat.org/issn/2374-3468). S2CID 9567647 (https://api.semanticscholar.org/CorpusID:9567647).
18. Klenze, Tobias; Bayless, Sam; Hu, Alan J. (2016). "Fast, Flexible, and Minimal CTL Synthesis via SMT" (https://link.springer.com/chapter/10.1007/978-3-319-41528-4_8). In Chaudhuri, Swarat; Farzan, Azadeh (eds.). *Computer Aided Verification*. Lecture Notes in Computer Science. Vol. 9779. Cham: Springer International Publishing. pp. 136–156. doi:10.1007/978-3-319-41528-4_8 (https://doi.org/10.1007%2F978-3-319-41528-4_8). ISBN 978-3-319-41528-4.
19. Bembenek, Aaron; Greenberg, Michael; Chong, Stephen (2023-01-11). "From SMT to ASP: Solver-Based Approaches to Solving Datalog Synthesis-as-Rule-Selection Problems" (https://doi.org/10.1145%2F3571200). *Proceedings of the ACM on Programming Languages*. **7** (POPL): 7:185–7:217. doi:10.1145/3571200 (https://doi.org/10.1145%2F3571200). S2CID 253525805 (https://api.semanticscholar.org/CorpusID:253525805).
20. Bauer, A.; Pister, M.; Tautschnig, M. (2007), "Tool-support for the analysis of hybrid systems and models", *Proceedings of the 2007 Conference on Design, Automation and Test in Europe (DATE'07)*, IEEE Computer Society, p. 1, CiteSeerX 10.1.1.323.6807 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.323.6807), doi:10.1109/DATE.2007.364411 (https://doi.org/10.1109%2FDATE.2007.364411), ISBN 978-3-9810801-2-4, S2CID 9159847 (https://api.semanticscholar.org/CorpusID:9159847)
21. Fränzle, M.; Herde, C.; Ratschan, S.; Schubert, T.; Teige, T. (2007), "Efficient Solving of Large Non-linear Arithmetic Constraint Systems with Complex Boolean Structure" (http://jsat.ewi.tudelft.nl/content/volume1/JSAT1_11_Fraenzle.pdf) (PDF), *Journal on Satisfiability, Boolean Modeling and Computation*, **1** (3–4 JSAT Special Issue on SAT/CP Integration): 209–236, doi:10.3233/SAT190012 (https://doi.org/10.3233%2FSAT190012)
22. Barbosa, Haniel; Barrett, Clark; Brain, Martin; Kremer, Gereon; Lachnitt, Hanna; Mann, Makai; Mohamed, Abdalrhman; Mohamed, Mudathir; Niemetz, Aina; Nötzli, Andres; Ozdemir, Alex; Preiner, Mathias; Reynolds, Andrew; Sheng, Ying; Tinelli, Cesare (2022). "cvc5: A Versatile and Industrial-Strength SMT Solver" (https://link.springer.com/chapter/10.1007/978-3-030-99524-9_24). In Fisman, Dana; Rosu, Grigore (eds.). *Tools and Algorithms for the Construction and Analysis of Systems, 28th International Conference*. Lecture Notes in Computer Science. Vol. 13243. Cham: Springer International Publishing. pp. 415–442. doi:10.1007/978-3-030-99524-9_24 (https://doi.org/10.1007%2F978-3-030-99524-9_24). ISBN 978-3-030-99524-9. S2CID 247857361 (https://api.semanticscholar.org/CorpusID:247857361).
23. Barrett, Clark; de Moura, Leonardo; Stump, Aaron (2005). "SMT-COMP: Satisfiability Modulo Theories Competition" (https://link.springer.com/chapter/10.1007%2F11513988_4). In Etessami, Kousha; Rajamani, Sriram K. (eds.). *Computer Aided Verification*. Lecture Notes in Computer Science. Vol. 3576. Springer. pp. 20–23. doi:10.1007/11513988_4 (https://doi.org/10.1007%2F11513988_4). ISBN 978-3-540-31686-2.

24. Barrett, Clark; de Moura, Leonardo; Ranise, Silvio; Stump, Aaron; Tinelli, Cesare (2011). "The SMT-LIB Initiative and the Rise of SMT: (HVC 2010 Award Talk)". In Barner, Sharon; Harris, Ian; Kroening, Daniel; Raz, Orna (eds.). *Hardware and Software: Verification and Testing*. Lecture Notes in Computer Science. Vol. 6504. Springer. p. 3. Bibcode:2011LNCS.6504....3B (https://ui.adsabs.harvard.edu/abs/2011LNCS.6504....3B). doi:10.1007/978-3-642-19583-9_2 (https://doi.org/10.1007%2F978-3-642-19583-9_2). ISBN 978-3-642-19583-9.
25. "SMT-COMP 2020" (https://smt-comp.github.io/2020/). *SMT-COMP*. Retrieved 2020-10-19.
26. Hassan, Mostafa; Urban, Caterina; Eilers, Marco; Müller, Peter (2018). "MaxSMT-Based Type Inference for Python 3" (https://link.springer.com/chapter/10.1007/978-3-319-96142-2_2). *Computer Aided Verification*. Lecture Notes in Computer Science. Vol. 10982. pp. 12–19. doi:10.1007/978-3-319-96142-2_2 (https://doi.org/10.1007%2F978-3-319-96142-2_2). ISBN 978-3-319-96141-5.
27. Loncaric, Calvin, et al. "A practical framework for type inference error explanation." (https://manu.sridharan.net/files/mycroft-preprint.pdf) ACM SIGPLAN Notices 51.10 (2016): 781-799.
28. Beaumont, Paul; Evans, Neil; Huth, Michael; Plant, Tom (2015). "Confidence Analysis for Nuclear Arms Control: SMT Abstractions of Bayesian Belief Networks". In Pernul, Günther; Y A Ryan, Peter; Weippl, Edgar (eds.). *Computer Security -- ESORICS 2015*. Lecture Notes in Computer Science. Vol. 9326. Springer. pp. 521–540. doi:10.1007/978-3-319-24174-6_27 (https://doi.org/10.1007%2F978-3-319-24174-6_27). ISBN 978-3-319-24174-6.
29. Ekici, Burak; Mebsout, Alain; Tinelli, Cesare; Keller, Chantal; Katz, Guy; Reynolds, Andrew; Barrett, Clark (2017). "SMTCoq: A Plug-In for Integrating SMT Solvers into Coq" (https://hal.archives-ouvertes.fr/hal-01669345/file/main.pdf) (PDF). In Majumdar, Rupak; Kunčak, Viktor (eds.). *Computer Aided Verification, 29th International Conference*. Lecture Notes in Computer Science. Vol. 10427. Cham: Springer International Publishing. pp. 126–133. doi:10.1007/978-3-319-63390-9_7 (https://doi.org/10.1007%2F978-3-319-63390-9_7). ISBN 978-3-319-63390-9. S2CID 206701576 (https://api.semanticscholar.org/CorpusID:206701576).
30. Blanchette, Jasmin Christian; Böhme, Sascha; Paulson, Lawrence C. (2013-06-01). "Extending Sledgehammer with SMT Solvers" (https://doi.org/10.1007/s10817-013-9278-5). *Journal of Automated Reasoning*. **51** (1): 109–128. doi:10.1007/s10817-013-9278-5 (https://doi.org/10.1007%2Fs10817-013-9278-5). ISSN 1573-0670 (https://search.worldcat.org/issn/1573-0670).

References

- Barrett, C.; Sebastiani, R.; Seshia, S.; Tinelli, C. (2009). "Satisfiability Modulo Theories" (https://books.google.com/books?id=shLvAgAAQBAJ&q=%22Satisfiability+Modulo+Theories%22). In Biere, A.; Heule, M.J.H.; van Maaren, H.; Walsh, T. (eds.). *Handbook of Satisfiability*. Frontiers in Artificial Intelligence and Applications. Vol. 185. IOS Press. pp. 825–885. ISBN 9781607503767.
- Ganesh, Vijay (September 2007). *Decision Procedures for Bit-Vectors, Arrays and Integers* (https://ece.uwaterloo.ca/~vganesh/Publications_files/vg2007-PhD-STANFORD.pdf) (PDF) (PhD). Computer Science Department, Stanford University.
- Jha, Susmit; Limaye, Rhishikesh; Seshia, Sanjit A. (2009). "Beaver: Engineering an efficient SMT solver for bit-vector arithmetic". *Proceedings of 21st International Conference on Computer-Aided Verification*. pp. 668–674. doi:10.1007/978-3-642-02658-4_53 (https://doi.org/10.1007%2F978-3-642-02658-4_53). ISBN 978-3-642-02658-4.
- Bryant, R.E.; German, S.M.; Velev, M.N. (1999). "Microprocessor Verification Using Efficient Decision Procedures for a Logic of Equality with Uninterpreted Functions" (https://kilthub.cmu.edu/articles/Microprocessor_Verification_Using_Efficient_Decision_Procedures_for_a_Logic_of_Equality_with_Uninterpreted_Functions/6607286/files/12097826.pdf) (PDF). *Analytic Tableaux and Related Methods*. pp. 1–13., pp. , .
- Davis, M.; Putnam, H. (1960). "A Computing Procedure for Quantification Theory" (https://doi.org/10.1145%2F321033.321034). *Journal of the Association for Computing Machinery*. **7** (3): 201–215. doi:10.1145/321033.321034 (https://doi.org/10.1145%2F321033.321034). S2CID 31888376 (https://api.semanticscholar.org/CorpusID:31888376).
- Davis, M.; Logemann, G.; Loveland, D. (1962). "A Machine Program for Theorem-Proving". *Communications of the ACM*. **5** (7): 394–397. doi:10.1145/368273.368557 (https://doi.org/10.1145%2F368273.368557). hdl:2027/mdp.39015095248095 (https://hdl.handle.net/2027%2Fmdp.39015095248095). S2CID 15866917 (https://api.semanticscholar.org/CorpusID:15866917).

- Kroening, D.; Strichman, O. (2008). *Decision Procedures — an algorithmic point of view* (https://books.google.com/books?id=anJsH3Dq5BIC&q=SMT). Theoretical Computer Science series. Springer. ISBN 978-3-540-74104-6.
- Nam, G.-J.; Sakallah, K.A.; Rutenbar, R. (2002). "A New FPGA Detailed Routing Approach via Search-Based Boolean Satisfiability". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. **21** (6): 674–684. doi:10.1109/TCAD.2002.1004311 (https://doi.org/10.1109%2FTCAD.2002.1004311).
- SMT-LIB: The Satisfiability Modulo Theories Library (http://smtlib.org/)
- SMT-COMP: The Satisfiability Modulo Theories Competition (http://www.smtcomp.org)
- Decision procedures - an algorithmic point of view (http://www.decision-procedures.org:)
- Sebastiani, R. (2007). "Lazy Satisfiability Modulo Theories". *Journal on Satisfiability, Boolean Modeling and Computation*. **3** (3–4): 141–224. CiteSeerX 10.1.1.100.221 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.100.221). doi:10.3233/SAT190034 (https://doi.org/10.3233%2FSAT190034).

This article was originally adapted from a column in the ACM SIGDA (http://www.sigda.org) e-newsletter (https://web.archive.org/web/20070208034716/http://www.sigda.org/newsletter/index.html) by Prof. Karem A. Sakallah. Original text is available here (http://archive.sigda.org/newsletter/2006/061215.txt)
