SMT-like Queries in Maple

Stephen A. Forrest

Maplesoft Europe Ltd., Cambridge, UK
`sforrest@maplesoft.com`

Abstract. The recognition that Symbolic Computation tools could benefit from techniques from the world of Satisfiability Checking was a primary motive for the founding of the SC^2 community. These benefits would be further demonstrated by the existence of "SMT-like" queries in legacy computer algebra systems; that is, computations which seek to decide satisfiability or identify a satisfying witness.
The Maple CAS has been under continuous development since the 1980s and its core symbolic routines incorporate many heuristics. We describe ongoing work to compose an inventory of such "SMT-like" queries extracted from the built-in Maple library, most of which were added long before the inclusion in Maple of explicit software links to SAT/SMT tools. Some of these queries are expressible in the SMT-LIB format using an existing logic, and it is hoped that those that are not could help inform future development of the SMT-LIB standard.

1 Introduction

1.1 Maple

Maple is a computer algebra system originally developed by members of the Symbolic Computation Group in the Faculty of Mathematics at the University of Waterloo. Since 1988, it has been developed and commercially distributed by Maplesoft (formally Waterloo Maple Inc.), a company based in Waterloo, Ontario, Canada, with ongoing contributions from affiliated research centres. The core Maple language is implemented in a kernel written in C++ and much of the computational library is written in the Maple language, though the system does employ external libraries such as LAPACK and the GNU Multiprecision Library (GMP) for special-purpose computations.

2 The commands `is` and `coulditbe`

Consistent with Maple's roots as a computer algebra system, its core symbolic solvers (such as `solve`, `dsolve`, and `int`) generally aim to provide a general solution to a posed problem which is both compact and useful. Further transformation or simplification of such solutions using simplifiers based on heuristic methods [3] is often necessary.

Nevertheless the approach of posing queries as questions about satisfiability or requests for a satisfying witness is not unknown in Maple. The most obvious example is in the commands `is` and `coulditbe`. These are the standard general-purpose commands in Maple for querying universal and existential properties, respectively, about a given expression. [5] They are widely used by other symbolic commands in Maple (e.g. `solve`, `int`).

The `is` command accepts an expression `p` and asks if `p` evaluates to the value `true` for every possible assignment of values to the symbols in `p`. The `coulditbe` command operates similarly but asks if there is any assignment of values to the symbols in `p` which could cause `p` to evaluate to `true`.

Both `is` and `coulditbe` return results in ternary logic: `true`, `false`, or `FAIL`. Both also make use of the "assume facility", which is a system for associating Boolean properties with symbolic variables. This provides limits on the range of possible assignments considered by `is` and `coulditbe` and is roughly analogous to a type declaration. For example, the expression `is(x^2>=0)` evaluates to `false` because there are many possible values of `x` which do not evaluate to nonnegative real numbers, in particular the imaginary unit $\sqrt{-1}$. By contrast, the expression `is(x^2>=0) assuming x::real` returns `true` because the range of possible values of `x` has been constrained to real numbers.

An illustrative example is found in the function `product`. In the evaluation of the expression `product(f(n),n=a..b)`, the system seeks to compute a symbolic formula for the product $\prod_{n=a}^{b} f(n)$. As one can verify by inspecting the source code with `showstat(product)`, the implementation of `product` computes a set of roots of $f(n)$ and, if neither a nor b is infinite, checks whether there exists a root r such that r is an integer and $a \leq r \leq b$. If so, it returns zero as the result of the product. (Similar logic is applied if either of a or b is infinite.)

As evidence of the ubiquity of such queries, Table 1 summarizes the distinct invocations of `is` and `coulditbe` encountered during a complete run through Maplesoft's internal test suite for the Maple library performed on 24 April 2018. (An investigation into an earlier version of this dataset was published in [4]). This includes both instances in which the test case explicitly calls `is`/`coulditbe` and instances in which `is`/`coulditbe` are invoked by other library functions such as `product`, as shown previously.

Note that in the above table, whenever logic $L2$ is an extension of logic $L1$, the listed results for logic $L2$ refer only to those queries which are expressible in $L2$ but not in $L1$. For example, the 2888 `is` queries expressible in `QF_LIA` are not included among the 1449 queries expressible in `QF_LIRA`, even though all of them are expressible in the more general logic.

In total, 24006 distinct `is` and 5701 distinct `coulditbe` queries were issued during the course of the test run. The inputs vary considerably in size and in the complexity of the underlying theory, and for both `is` and `coulditbe` approximately 11% of queries cannot be decided (i.e. return `FAIL` rather than `true` or `false`). A complete list of queries encountered may be viewed at https://doi.org/10.5281/zenodo.943349.

Description	is	coulditbe
Total expressible in SMT-LIB	15572	4690
Expressible with `QF_LIA`	2888	1686
Expressible with `QF_NIA`	2129	744
Expressible with `QF_LRA`	1542	505
Expressible with `QF_NRA`	284	41
Expressible with `AUFLIRA`	1449	687
Expressible with `AUFNIRA`	7230	1027
Total not expressible in SMT-LIB	8434	1011
Expressible with complex arithmetic	4134	565
Linear arithmetic with Gaussian integers ("QF_LICA")	258	171
Nonlinear arithmetic with Gaussian integers ("QF_NICA")	259	88
Linear arithmetic with complex numbers ("AUFLIRCA")	165	32
Nonlinear arithmetic with complex numbers ("AURNIRCA")	2728	207
All "special" functions	3248	441
Exponential functions and logarithms	461	76
RootOf expressions	232	40
RootOf with exponential and trigometric functions	231	24
Remaining queries with Boolean structure	599	5
Total distinct queries	24006	5701

Table 1. Distinct `is` and `coulditbe` queries encountered in a full library test run

Of the total, 15572 of the `is` queries and 4690 of the `coulditbe` queries can be assigned to one of the SMT-LIB2 predefined logics. Of the queries which cannot be so assigned, the reasons include the use of special functions unsupported by SMT-LIB, as well as complex arithmetic.

3 Future Work

Recent versions of Maple have seen the addition of explicit links to SAT and SMT solvers: Maple 2018 is distributed with both the SAT solver MapleSAT [1] and the SMT solver Z3 [7]. In future, we aim to examine the effectiveness of using these packaged solvers on SMT instances which arise during evaluation of symbolic expressions.

An important factor in this assessment will be whether this implementation offers better performance and meaningful answers (not `FAIL`) for a larger class of such queries than existing tools in Maple.

References

1. Hui Liang, Vijay Ganesh. MapleSAT development site. https://sites.google.com/a/gsd.uwaterloo.ca/maplesat/
2. Clark Barrett, Pascal Fontaine, and Cesare Tinelli. *The Satisfiability Modulo Theories Library (SMT-LIB)*, http://www.smt-lib.org, 2016.

3. Jacques Carette. 2004. Understanding Expression Simplification. In *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation, Santander, Spain* (ISSAC 2004), ACM, New York, NY, USA, 72-79. doi:10.1145/1005285.1005298.
4. Stephen A. Forrest. 2017. Integration of SMT-LIB Support into Maple. Second Annual SC^2 Workshop, ISSAC 2017, Kaiserslautern, Germany. http://www.sc-square.org/CSA/workshop2-papers/EA5-FinalVersion.pdf
5. *The Assume Facility in Maple*, Maple Online Help : The Assume Facility.
6. *Logics in SMT-LIB*, http://smtlib.org/logics.shtml.
7. de Moura, L. M., and Bjørner, N. *Z3: an efficient SMT solver.* In TACAS (2008), vol. 4963 of Lecture Notes in Computer Science, Springer, pp. 337340. https://github.com/Z3Prover/z3.