OptiMathSAT: A Tool for Optimization Modulo Theories

Roberto Sebastiani and Patrick Trentin(✉)

DISI, University of Trento, Trento, Italy
patrick.trentin@unitn.it

Abstract. Many SMT problems of interest may require the capability of finding models that are *optimal* wrt. some objective functions. These problems are grouped under the umbrella term of *Optimization Modulo Theories – OMT*. In this paper we present OptiMathSAT, an OMT tool extending the MathSAT5 SMT solver. OptiMathSAT allows for solving a list of optimization problems on SMT formulas with linear objective functions –on the Boolean, the rational and the integer domains, and on their combination thereof– including MaxSMT. Multiple objective functions can be combined together and handled either independently, or lexicographically, or in a min-max/max-min fashion.

OptiMathSAT ships with an extended SMT-LIBV2 input syntax and C API bindings, and it preserves the incremental attitude of its underlying SMT solver.

1 Introduction

SMT solvers are currently used as backend engines in many formal verification (FV) tools for Hardware, Software and Hybrid Systems. Many SMT problems of interest for FV or for other disciplines, however, require the capability of finding models that are *optimal* wrt. some objective functions [6,8,9,11,13–16,18–20]. These problems are grouped under the umbrella term of *Optimization Modulo Theories – OMT*.

For instance, in SMT-based model checking with timed or hybrid systems, you may want to find executions which optimize the value of some parameter while fulfilling/violating some property –e.g., to find the minimum opening time interval for a railcrossing causing a safety violation. (See e.g. [19] for some examples.) Also, a recent application of OMT is the SMT-based computation of the worst-case execution time (WCET) of loop-free programs [12], which finds tighter over-approximations of the WCET than other state-of-the-art approaches. A longer list of OMT applications in formal verification and in other disciplines can be found in [8,11,14–16,19].

This work is supported by Semiconductor Research Corporation (SRC) under GRC Research Project 2012-TJ-2266 WOLF. We thank Alberto Griggio for support with MathSAT5 code.

D. Kroening and C.S. Păsăreanu (Eds.): CAV 2015, Part I, LNCS 9206, pp. 447–454, 2015.
DOI: 10.1007/978-3-319-21690-4_27

In this paper we present OptiMathSAT, an OMT tool extending the MathSAT5 SMT solver [3,10], implementing the OMT procedures described in [18–20]. OptiMathSAT allows for solving a list of optimization problems on SMT formulas with linear objective functions –on the Boolean, the rational and the integer domains, and on their combination thereof– including MaxSMT. Multiple objective functions can be combined together and handled either independently, or lexicographically, or in a min-max/max-min fashion. Like MathSAT5, it is freely available for research and evaluation purposes [4], and it is currently used in some innovative projects (see Sect. 5).

Related Tools. Currently few other OMT tools exist. Closest to OptiMathSAT are Symba [14] and the very-recent νZ [6,7], which are both built on top of Z3. [14] considered the problem of optimizing multiple rational cost functions at the same time. Symba uses the underlying SMT solver as black-box, and it features additional ad hoc techniques for detecting unbounded costs and optimization. νZ supports both single-objective linear optimization –over a real, integer or bit-vector term – and multi-objective optimization in either boxed, lexicographic or Pareto-optimization mode. It ships with several specialized engines for MaxSMT and with pre-processing techniques that re-encode the 0-1 integer variables of the input formula into Pseudo-Boolean or MaxSMT constraints. We refer the reader to the related work section of [19] for a more-detailed analysis of other OMT-related approaches and tools.

Content. This paper is structured as follows. Section 2 provides a brief outline of OptiMathSAT architecture, followed by a description of its optimization functionalities and interfaces in Sect. 3. Section 4 presents a short example, and Sect. 5 reviews some recent interesting applications of OptiMathSAT. Section 6 concludes the paper with hints of some future developments. An extended version of this paper, containing a performance evaluation and some more details, is available from OptiMathSAT web page [4].

2 Architecture

OptiMathSAT is written in *C++* and it is built as an extension of MathSAT5, which implements the standard lazy SMT paradigm (see [5]). Unlike the OMT algorithms in [6,14], which are based on an *offline* architecture –in which the SMT Solver is incrementally called multiple times as a black-box– OptiMathSAT is based on an *inline architecture* –in which the SMT solver is run only once and its internal SAT solver is modified to handle the search for the optima [18–20]. Although harder to implement, the *inline* architecture has showed better performance for OptiMathSAT than the *offline* one [18,19]. (We refer the reader to [19] for a comparison of the two architectures.)

The optimization algorithm can explore the search space in *linear-search* mode, by pruning one intermediate solution at a time, or in *binary-search* mode, by introducing cuts bisecting the search space, or in *adaptive-search* mode, which uses adaptive heuristics to choose among the linear- and binary-search modes at each search step.

Some functionalities, such as the control loop for *lexicographic* optimization and the assertion of soft clauses, are handled at a higher level of abstraction by means of a combination of MathSAT5 and OptiMathSAT API calls.

3 Optimization Functionalities

OptiMathSAT is mainly a tool for (single- and multiple-objective) OMT with linear objective functions OMT($\mathcal{LA} \cup \mathcal{T}$) s.t. "$\mathcal{LA}$" denotes linear arithmetic over either the rationals ($\mathcal{LRA}$), or the integers ($\mathcal{LIA}$) or their combination $\mathcal{LRIA}$, and $\mathcal{T}$ denotes any other Nelson-Oppen theory supported by MathSAT5. For each objective it is possible to specify both global and local bounds, if known.[1] OptiMathSAT can use this information to explore the search space in binary or in adaptive search mode, which might improve the overall performance of the solver. We support objective functions over the rational, integer and Boolean domains[2], or their combinations.

Here we provide a brief list of OptiMathSAT optimization functionalities, omitting the functionalities inherited from MathSAT5 [10]. A detailed description of the implemented algorithms is presented in [18–20].

3.1 Single-Objective Optimization

We discuss first the case in which we have only one objective function, namely *obj*.

Linear Arithmetic Optimization over $\mathcal{LRA}$, $\mathcal{LIA}$ and $\mathcal{LRIA}$. Given some term *obj* on $\mathcal{LA}$, OptiMathSAT finds a solution (if any) which makes the term *obj* minimum/maximum. This is based on a combination of SMT and linear [integer] programming techniques.

Partial Weighted MaxSMT and SMT with Pseudo-Boolean Objectives (PB-SMT). Given an input formula $\varphi_h \wedge \varphi_s$, where φ_h contains hard constraints and φ_s contains soft constraints with positive weights, the goal of *partial weighted* MaxSMT [9,16] is to find a model M s.t. $M \models \varphi_h \wedge \varphi_s^M$ and φ_s^M is a subset of φ_s in which the soft-constraints have the largest cumulative weight possible. Similarly, OptiMathSAT allows also for defining Pseudo-Boolean objective functions in the form $\sum_i w_i \psi_i$, where w_i are numerical constants and ψ_i are sub-formulas.

Unlike with the procedures in [6,8,9,16], which use specialized algorithms for MaxSMT/PB-SMT, OptiMathSAT works by encoding the problem into the optimization of an $\mathcal{LRA}$ term, as described in [19]. This allows for combining the MaxSMT terms with other objectives, as we describe in Sect. 3.2.

[1] Local bounds have a special use in boxed multi-objective optimization (see Sect. 3.2). In single-objective and lexicographic optimization, they coincide with global bounds.

[2] i.e. MaxSMT and SMT with Pseudo-Boolean objective functions.

Notice that it is possible to interrupt the search of OptiMathSAT (e.g., by setting a timeout) and to still have access to the current sub-optimal solutions and its model.

3.2 Multi-objective Combination

The interface of OptiMathSAT allows for combining multiple objective functions obj_1, ..., obj_N in various ways.

Multiple Independent Objectives [14]**.** (Aka Boxed Optimization [6].) OptiMathSAT can solve simultaneously N independent optimization problems $\langle \varphi, obj_1 \rangle, ..., \langle \varphi, obj_N \rangle$, optionally building the corresponding optimum models $M_0, ..., M_N$.[3] (In the empirical evaluation presented in [20], we showed that using this optimization strategy can be considerably more efficient than solving N single-objective optimization problems.) This option is the default configuration.

Lexicographic Optimization. OptiMathSAT optimizes lexicographically the objectives $obj_1, ..., obj_N$ by decreasing level of priority. If any objective obj_i is unsatisfiable or unbounded, the search returns.

Min-max and Max-min. The goal of a min-max problem is to find the maximum value of an obj s.t. $\bigwedge_{i=0}^{N} (obj \leq obj_i) \wedge \bigvee_{i=0}^{N} (obj_i = obj)$, obj being a fresh variable.[4] Max-min is dual. OptiMathSAT provides syntactic-sugar extensions to SMT-LIBv2 that allow for encoding this type of objectives.

Linear Combination. Obviously, one can also create objectives that are a linear combination of other objectives $obj_1, ..., obj_N$, i.e., $obj = \sum_{i=1}^{N} w_i \cdot obj_i$.

We remark that all the above combinations hold for obj_i cost functions over every domain, including Boolean. For instance, you can combine together MaxSMT with OMT optimization over Integer or Real objectives.

3.3 Interfaces

Input Language. OptiMathSAT functions are accessible through a list of commands, extending the SMT-LIBv2 syntax, which is shown in a concise description in Fig. 1. Notice that, differently from νZ [7], in case of a MaxSMT problem we require the user to build *explicitly* a minimization objective using the ID associated with the asserted soft clauses, i.e., by writing "`(minimize ID)`". The advantage of this requirement is that we allow for arbitrary composition of the MaxSMT objective with other linear arithmetic functions, which can be useful in particular contexts (for instance, to build obj functions on mixed Boolean/numeric domains, as with Linear Generalized Disjunctive Programming (LGDP) problems [17]).

[3] Since the N input problems are independent to one another, the local bounds of each objective obj_i do not have any side effect on the feasible solutions of all other objectives obj_j, as if the N problems were solved separately.

[4] Notice that in the actual encoding we drop the "$\bigvee$" part of the formula, since it is unnecessary and may cause extra Boolean search.

```
  ; assert soft clauses for MaxSMT (weight/dweight
  ;    defaults to 1, id defaults to 'I')
(assert-soft <term> [:weight <numeral>] [:id <string>])
(assert-soft <term> [:dweight <decimal>] [:id <string>])
  ; push new objectives on the stack
(minimize|maximize <term> [:local-lb <decimal>]
                          [:local-ub <decimal>])
(minmax|maxmin <term>...<term> [:local-lb <decimal>]
                               [:local-ub <decimal>])
  ; set multi-objective optimization mode
(set-option :opt.priority box|lex)
  ; optimize objective(s) in the stack
(check-sat)
  ; load model of objective with stack index <numeral>
(set-model <numeral>)
  ; retrieve model values
(get-model)
(get-value <VAR>)
```

Fig. 1. SMT-LIBv2 Optimization Extensions, square brackets corresponds to optional parameters, whereas "|" stands for alternative choices.

```
; set goods quantity
(assert (= 250 (+ q1 q2 q3 q4)))
; set goods offered by each supplier
(assert (and (or (= q1 0) (and (<=  50 q1) (<= q1 250)))
             (or (= q2 0) (and (<= 100 q2) (<= q2 150)))
             (or (= q3 0) (and (<= 100 q3) (<= q3 100)))
             (or (= q4 0) (and (<=  50 q4) (<= q4 100)))))
; a supplier is 'used' if sends more than zero items
(assert (and (=> s1 (not (= q1 0))) (=> s2 (not (= q2 0)))
           (=> s3 (not (= q3 0))) (=> s4 (not (= q4 0)))))
; supply from the largest number of suppliers
(assert-soft s1 :id ignored_suppliers)
(assert-soft s2 :id ignored_suppliers)
(assert-soft s3 :id ignored_suppliers)
(assert-soft s4 :id ignored_suppliers)
; set goal (A)
(minimize (+ (* q1 23) (* q2 21) (* q3 20) (* q4 10)))
; set goal (B)
(minimize ignored_suppliers)
; optimize lexicographically
(set-option :opt.priority lex)
(check-sat)
; print model
(set-model 1)
(get-model)
```

Fig. 2. SMT-LIBv2 encoding of the problem.

C API. The optimization functions of OptiMathSAT are also available through its C API, which extends that of MathSAT5 [3]. A detailed documentation of the *C API*, the SMT-LIBv2 language extensions and some usage examples are accessible on OptiMathSAT website [4].

Incremental Interface. Like MathSAT5, OptiMathSAT provides a push/pop interface for adding and removing objectives and pieces of formulas from the formula stack, which allows for reusing information from one optimization search to another to improve the global performance of the search [20].

4 Example

In Fig. 2 we present a toy example that illustrates how to encode a problem into the extended SMT-LIBv2 language of OptiMathSAT. A small company urgently needs 250 units of some goods. Suppliers s_1, s_2, s_3, s_4 offer to supply up to $250, 150, 100, 100$ units of goods starting from the minimum quantity of $50, 100, 100, 50$ units respectively. Their prices are $23\$, 21\$, 20\$, 10\$$ per unit respectively. Our goal is (A) to minimize the overall purchase cost and, at cost tie, (B) to maximize the number of suppliers.

A simple OMT encoding of the problem is shown in Fig. 2. In this example there are two combinations of suppliers –s_2, s_4 and s_1, s_3, s_4– from which we can purchase the goods at the minimum cost of 4150\$. Therefore, the tie is broken by our secondary goal (B), which imposes our preference on the second solution. The optimum model of a lexicographic optimization is always associated with the top-most objective on the internal stack. Since in this example there are only two objectives, this objective can be selected by passing 1 to the `set-model` command. As mentioned in Sect. 3.3, notice that we explicitly ask for `ignored_suppliers`, the label of the MaxSMT constraints, to be minimized. OptiMathSAT solved the problem in 10ms.

5 Applications

We briefly mention two examples of recent applications –which are very innovative in their respective domains– that have been technologically enabled by OMT and use OptiMathSAT as backend automated-reasoning engine.

Structured Learning Modulo Theories. In Machine Learning applications, performing inference and learning in hybrid domains –characterized by both continuous and Boolean/discrete variables– is a particularly daunting task. *Structured Learning Modulo Theories (SLMT)* [21] addresses the problem by combining (Structured-Output) Support Vector Machines (SVNs) with OMT, so that the latter plays the role of inference and separation oracle for the former. The tool LMT implementing the SLMT method [2] uses OptiMathSAT as backend OMT engine.

Automated Reasoning on Constrained Goal Models. Goal Models (GM) are used in Requirements Engineering to represent software requirements, objectives, and design qualities [22]. *Constrained Goal Models (CGM)* are a novel, formal version of GM which are enriched with constraints so that to handle preferences, numerical attributes and resources (e.g., scores, financial cost, workforce, etc.). OptiMathSAT is used as a backend reasoning engine of CGM-Tool [1], a tool for building and reasoning on *CGM*s, allowing for automatically verifying the realizability of a CGM and for finding optimal realizations according to some specified criterion.

6 Future Developments

We plan to extend OptiMathSAT capabilities along several directions. For instance, we are interested into generalizing our implementation to support objective functions extended on other theories, i.e. bit-vector. We are also considering to add the possibility of combining multiple objectives for Pareto-optimization. Finally, we plan to parallelize OMT so that to exploit the multi-core architectures of modern CPUs.

References

1. CGM-Tool. www.cgm-tool.eu
2. LMT. http://disi.unitn.it/teso/lmt/lmt.tgz
3. MathSAT 5. http://mathsat.fbk.eu/
4. OptiMathSAT. http://optimathsat.disi.unitn.it/
5. Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. In: Handbook of Satisfiability, chap. 26, pp. 825–885. IOS Press (2009)
6. Bjorner, N., Phan, A.-D.: νZ - Maximal satisfaction with Z3. In: Proceedings of SCSS. Invited presentation., Gammart, Tunisia, December 2014. EasyChair Proceedings in Computing (EPiC). http://www.easychair.org/publications/?page=862275542
7. Bjørner, N., Phan, A.-D., Fleckenstein, L.: νZ - An optimizing SMT solver. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 194–199. Springer, Heidelberg (2015, to appear)
8. Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R., Stenico, C.: Satisfiability modulo the theory of costs: foundations and applications. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 99–113. Springer, Heidelberg (2010)
9. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: A modular approach to maxSAT modulo theories. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 150–165. Springer, Heidelberg (2013)
10. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The mathSAT5 SMT solver. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013 (ETAPS 2013). LNCS, vol. 7795, pp. 93–107. Springer, Heidelberg (2013)
11. Dillig, I., Dillig, T., McMillan, K.L., Aiken, A.: Minimum satisfying assignments for SMT. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 394–409. Springer, Heidelberg (2012)

12. Henry, J., Asavoae, M., Monniaux, D., Maïza, C.: How to compute worst-case execution time by optimization modulo theory and a clever encoding of program semantics. SIGPLAN Not. **49**(5), 43–52 (2014)
13. Larraz, D., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: Minimal-model-guided approaches to solving polynomial constraints and extensions. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 333–350. Springer, Heidelberg (2014)
14. Li, Y., Albarghouthi, A., Kincad, Z., Gurfinkel, A., Chechik, M.: Symbolic optimization with SMT solvers. In: POPL. ACM Press (2014)
15. Manolios, P., Papavasileiou, V.: ILP modulo theories. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 662–677. Springer, Heidelberg (2013)
16. Nieuwenhuis, R., Oliveras, A.: On SAT modulo theories and optimization problems. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 156–169. Springer, Heidelberg (2006)
17. Raman, R., Grossmann, I.: Modelling and computational techniques for logic based integer programming. Comput. Chem. Eng. **18**(7), 563–578 (1994)
18. Sebastiani, R., Tomasi, S.: Optimization in SMT with $\mathcal{LA}\mathbb{Q}$ Cost Functions. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 484–498. Springer, Heidelberg (2012)
19. Sebastiani, R., Tomasi, S.: Optimization modulo theories with linear rational costs. ACM Trans. Comput. Logics 16(2) (2015). doi:10.1145/2699915
20. Sebastiani, R., Trentin, P.: Pushing the envelope of optimization modulo theories with linear-arithmetic cost functions. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 335–349. Springer, Heidelberg (2015)
21. Teso, S., Sebastiani, R., Passerini, A.: Structured learning modulo theories. Artificial Intelligence (2015). http://disi.unitn.it/rseba/publist.html
22. Van Lamsweerde, A.: Goal-oriented requirements engineering: a guided tour. In: Proceedings of the Fifth IEEE International Conference on Requirements Engineering, RE 2001, pp. 249–262. IEEE Computer Society (2001)