A Proposal for an OMT Extension to SMT-LIB

Nestan Tsiskaridze[1], Andrew Reynolds[2], Cesare Tinelli[2] and Clark Barrett[1]

[1]Stanford University, Stanford, USA
[2]The University of Iowa, Iowa City, USA

Abstract

Optimization Modulo Theories (OMT) has emerged as a prominent extension of the Satisfiability Modulo Theories (SMT) paradigm, bringing optimization objectives into first-order logic constraint solving. Unlike SMT, which focuses solely on satisfiability with respect to a given theory, OMT additionally seeks to optimize a specified objective function. Several state-of-the-art SMT solvers have integrated OMT capabilities. However, no official SMT-LIB extension has yet been adopted for OMT. As a result, OMT benchmarks lack standardization, which hinders broader adoption and progress in the field. In this paper, we propose an extension to SMT-LIB that supports all of the different flavors of OMT found in the literature. Our goal is to foster the development of OMT solvers and applications, to enable more robust, reusable, and comparable OMT solutions, and to promote the creation of standardized OMT benchmarks in SMT-LIB format for systematic and meaningful evaluation.

Keywords

SMT-LIB syntax, Optimization Modulo Theories, OMT

1. Introduction

Optimization Modulo Theories (OMT) builds on the highly successful Satisfiability Modulo Theories (SMT) [1] paradigm: but while SMT focuses on finding a model for a first-order formula, OMT extends this by introducing an objective term that must be optimized according to some order on the term's domain.

This added expressiveness has established OMT as a powerful tool fueling progress across a wide range of applications. These include formal verification and model checking [2, 3], program analysis [4, 5, 6, 7, 8], security analysis [9, 10, 11, 12], resource-constrained scheduling and planning [13, 14, 15, 16, 17, 18, 19, 20, 21, 22], requirements engineering and specification synthesis [23, 24, 25, 26], system design and configuration [27, 28, 29, 30, 31, 32, 33, 34], as well as applications in machine learning [35, 36] and quantum annealing [37].

To address the wide range of optimization goals and underlying theories encountered in practice, the OMT community has developed specialized procedures for various theories, such as arithmetic and bitvectors, different types of optimization—including single- and multi-objective optimization, and specific search strategies including linear, binary, and hybrid approaches. The result is a fragmented landscape of theory-specific solutions, each tailored to particular combinations of goals and theories. Our goal in this work is to start a discussion with the aim of eventually achieving consensus in the community around an SMT-LIB standard extension that provides a uniform interface for representing a large class of OMT problems. We draw inspiration from extensive previous work on OMT and OMT extensions to SMT-LIB (e.g., [38, 39]). Our specific goal of unifying previous approaches draws on a recent effort on *Generalized Optimization Modulo Theories* (GOMT) [40], which provides an abstract formalization. Specific OMT approaches can then be seen as specific instances of GOMT.

The rest of the paper is organized as follows. After an overview of related work, below, we introduce the necessary formal preliminaries in Section 2 followed by our proposed SMT-LIB extension in Section 3. We provide an extensive set of examples in Section 4 and conclude in Section 5.

SMT 2025: 23rd International Workshop on Satisfiability Modulo Theories, August 10–11, 2025, Glasgow, UK
nestan@cs.stanford.edu (N. Tsiskaridze); andrew.j.reynolds@gmail.com (A. Reynolds); cesare-tinelli@uiowa.edu (C. Tinelli); barrett@cs.stanford.edu (C. Barrett)
0000-0002-4729-9770 (N. Tsiskaridze); 0000-0002-3529-8682 (A. Reynolds); 0000-0002-6726-775X (C. Tinelli); 0000-0002-9522-3084 (C. Barrett)

Related Work Domain-specific OMT solving techniques have been developed based on the SMT-LIB theories and objective types of optimization problems involved.

The OMT problem was first introduced in [41]. Various specialized techniques have since been proposed to handle both theory-specific and objective-specific variants of the OMT problem.

Theory-specific OMT techniques address objectives involving linear real arithmetic [42, 43], linear integer arithmetic[42, 44, 45] pseudo-Booleans [46, 43, 47, 48], bitvectors [49, 50], bitvectors combined with floating-point arithmetic [50], and nonlinear real and integer arithmetic [51].

Objective-specific techniques were developed for lexicographic optimization [42, 45], Pareto optimization [52, 42], Box [53, 42, 45], MinMax [38], MaxSMT [41, 54], and All-OMT [55]. For more details of theory-specific and objective-specific OMT approaches, we refer the reader to [55].

A recent orthogonal approach [40] unifies all the above OMT problems into a Generalized OMT (GOMT) framework and proposes an abstract, theory- and objective-agnostic calculus for solving them. Like SMT-LIB itself, GOMT supports arbitrary combinations of theories. It also supports objectives of any sort with at least a partial order defined on it (including user-defined orders).

On the SMT-LIB syntax side, several SMT solvers have extended the SMT-LIB language to support OMT, although no official standard has yet been adopted. Notably, OptiMathSAT [38] has introduced commands to encode optimization objectives and soft constraints within SMT-LIB v2, with details available in its documentation. Likewise, Z3 offers optimization functionality through its νZ [39] extension, which adopts a similar syntax.

Efforts have been made to integrate modeling languages with OMT to bridge the gap between high-level problem formulations and advanced solver capabilities. Notably, [56] investigates the integration between MiniZinc [57]—a solver-independent, high-level modeling language for constraint satisfaction and optimization problems—with OMT. To enable OMT solving from MiniZinc models, the authors extended OptiMathSAT solver with a FlatZinc interface and developed a tool to translate SMT and OMT problems into MiniZinc format. FlatZinc [58], MiniZinc's low-level, solver-agnostic target language, enables communication with constraint and optimization solvers. OptiMathSAT includes experimental support for a subset of the FlatZinc 1.6 language, allowing users to solve MiniZinc-defined optimization problems with OMT techniques. Further supporting solver interoperability, the open-source tool fzn2omt [59] translates FlatZinc models into SMT-LIB format, making them compatible with solvers such as OptiMathSAT, Barcelogic [60], Z3 [61], and CVC5 [62].

These efforts seek to provide interfaces for formulating and solving optimization problems with OMT solvers. However, the lack of a general SMT-LIB standard for OMT continues to limit the broader adoption and seamless integration of OMT across tools and solvers in the community. This lack of standardization hampers collaboration, makes it difficult to share benchmarks and tools, and is an obstacle to building a comprehensive OMT benchmark library.

2. Formal Preliminaries

We work within the standard many-sorted first-order logic setting for SMT, with the usual notions of signatures, terms, and interpretations. A formula is a term of sort `Bool`. We write $\mathcal{I} \models \phi$ to mean that formula ϕ holds in or is *satisfied* by an interpretation $\mathcal{I}$. A *theory* is a pair $\mathcal{T} = (\Sigma, \mathbf{I})$, where Σ is a signature and $\mathbf{I}$ is a class of Σ-interpretations. We call the elements of $\mathbf{I}$ *$\mathcal{T}$-interpretations*. We write $\Gamma \models_{\mathcal{T}} \phi$, where Γ is a formula (or a set of formulas), to mean that Γ *$\mathcal{T}$-entails* ϕ, i.e., every $\mathcal{T}$-interpretation that satisfies (each formula in) Γ satisfies ϕ as well. A Σ-formula ϕ is *satisfiable* (resp., *unsatisfiable*) *in* $\mathcal{T}$ if it is satisfied by some (resp., no) $\mathcal{T}$-interpretation. For convenience, for the rest of the paper, *we assume an arbitrary but fixed background theory $\mathcal{T}$ with equality and with signature Σ*. We also fix an infinite set $\mathcal{X}$ of sorted variables with sorts from Σ and assume $\prec_{\mathcal{X}}$ is some total order on $\mathcal{X}$. We assume all terms and formulas are Σ-terms and Σ-formulas with free variables from $\mathcal{X}$. Since $\mathcal{T}$ is fixed, we will often abbreviate $\models_{\mathcal{T}}$ as $\models$ and consider only interpretations that are $\mathcal{T}$-interpretations assigning a value to every variable in $\mathcal{X}$.

Let s be a Σ-term. We denote by $s^{\mathcal{I}}$ the value of s in an interpretation $\mathcal{I}$, defined as usual by

recursively determining the values of sub-terms. We denote by $FV(s)$ the set of all variables occurring in s. Similarly, we write $FV(\phi)$ to denote the set of all the free variables occurring in a formula ϕ. If $FV(\phi) = \{v_1, \ldots, v_n\}$, where for each $i \in [1, n), v_i \prec_{\mathcal{X}} v_{i+1}$, then the relation *defined by* ϕ (in $\mathcal{T}$) is $\{(v_1^{\mathcal{I}}, \ldots, v_n^{\mathcal{I}}) \mid \mathcal{I} \models \phi$ for some $\mathcal{T}$-interpretation $\mathcal{I}\}$. A relation is *definable in* $\mathcal{T}$ if there is some formula that defines it.

We adopt the standard notion of *strict partial order* $\prec$ on a set A, that is, a relation in $A \times A$ that is irreflexive, asymmetric, and transitive. The relation $\prec$ is a *strict total order* if, in addition, $a_1 \prec a_2$ or $a_2 \prec a_1$ for every pair a_1, a_2 of distinct elements of A. As usual, we call $\prec$ *well-founded* over a subset A' of A if A' contains no infinite descending chains. An element $m \in A$ is *minimal (with respect to* $\prec$*)* if there is no $a \in A$ such that $a \prec m$. If A has a unique minimal element, it is called a *minimum*.

2.1. Generalized Optimization Modulo Theories

We rely on the *Generalized Optimization Modulo Theories* framework, which conveniently unifies many OMT variants under a single umbrella. The definition and explanation below are taken from [40].

Definition 1 (Generalized Optimization Modulo Theories (GOMT))**.** *A Generalized Optimization Modulo Theories problem is a tuple* $\mathcal{GO} := \langle t, \prec, \phi \rangle$*, where:*

- t*, a* Σ*-term of some sort* σ*, is an objective term to optimize;*
- $\prec$ *is a strict partial order definable in* $\mathcal{T}$*, whose defining formula has two free variables, each of sort* σ*; and*
- ϕ *is a* Σ*-formula.*

For any GOMT problem $\mathcal{GO}$ and $\mathcal{T}$-interpretations $\mathcal{I}$ and $\mathcal{I}'$, we say that:

- $\mathcal{I}$ is *$\mathcal{GO}$-consistent* if $\mathcal{I} \models \phi$;
- $\mathcal{I}$ *$\mathcal{GO}$-dominates* $\mathcal{I}'$, denoted by $\mathcal{I} <_{\mathcal{GO}} \mathcal{I}'$, if $\mathcal{I}$ and $\mathcal{I}'$ are $\mathcal{GO}$-consistent and $t^{\mathcal{I}} \prec t^{\mathcal{I}'}$; and
- $\mathcal{I}$ is a *$\mathcal{GO}$-solution* if $\mathcal{I}$ is $\mathcal{GO}$-consistent and no $\mathcal{T}$-interpretation $\mathcal{GO}$-dominates $\mathcal{I}$.

Informally, the term t represents the *objective function*, whose value we want to optimize. The order $\prec$ is used to compare values of t, with a value a being considered *better* than a value a' if $a \prec a'$. Finally, the formula ϕ imposes constraints on the values that t can take. It is easy to see that the value of $t^{\mathcal{I}}$ assigned by a $\mathcal{GO}$-solution $\mathcal{I}$ is always minimal. As a special case, if $\prec$ is a total order, then $t^{\mathcal{I}}$ is also unique (i.e., it is a minimum). Once we have fixed a GOMT problem $\mathcal{GO}$, we will informally refer to a $\mathcal{GO}$-consistent interpretation as a *solution (of* ϕ*)* and to a $\mathcal{GO}$-solution as an *optimal solution*.

3. An OMT Extension for SMT-LIB

This section introduces an extension to SMT-LIB v2.7 [63] that provides options, commands, and attributes for expressing the general class of OMT problems captured by the GOMT formalization. This extension is informed and inspired by previous work on OMT extensions in OptiMathSAT and Z3.

We also provide special syntax for specific OMT subproblems, for convenience. For example, though it can be encoded using GOMT, we include specific commands for the MaxSMT problem. Namely, following previous work, we provide commands for asserting *soft* constraints with assigned weights, the goal being to satisfy all hard (i.e., non-soft) constraints while maximizing the total weight of the satisfied soft ones.

Figure 1 shows the proposed extensions to the grammar of SMT-LIBv2.7. It should be understood in the context of the grammar in the SMT-LIB v2.7 reference document [63], i.e., although some pieces of the grammar listed there are repeated here for convenience, the extension is built on the full standard, and thus a complete understanding requires understanding that document.

We first note that in our proposal, OMT capabilities are not enabled by default. To enable them, the `:enable-omt` option must be set to `true`. In other words, the SMT-LIB script must include the line: `(set-option :enable-omt true)`. We now explain the remainder of Figure 1. We focus on the new commands, explaining the other pieces as they become relevant for understanding the commands.

Tokens

Command names: `define-objective` `define-multi-objective` `define-maxsmt-objective` `assert-soft` `optimize-sat` `optimize-sat-next`

Identifiers

⟨*objective-kind*⟩ ::= `OBJECTIVE_MIN` | `OBJECTIVE_MAX`
⟨*m-objective-kind*⟩ ::= `OBJECTIVE_LEX` | `OBJECTIVE_PARETO` | `OBJECTIVE_BOX` | `OBJECTIVE_MINMAX` | `OBJECTIVE_MAXMIN`
⟨*strategy*⟩ ::= `STRATEGY_LINEAR` | `STRATEGY_BINARY`

Attributes

⟨*strategy_attribute*⟩ ::= `:strategy` ⟨*strategy*⟩ | ⟨*attribute*⟩
⟨*order_attribute*⟩ ::= `:order` ⟨*symbol*⟩
⟨*bound_attribute*⟩ ::= `:lower` ⟨*term*⟩ | `:upper` ⟨*term*⟩
⟨*assumption_attribute*⟩ ::= `:assumption` ⟨*term*⟩
⟨*objective_attribute*⟩ ::= ⟨*strategy_attribute*⟩ | ⟨*order_attribute*⟩ | ⟨*bound_attribute*⟩ | ⟨*assumption_attribute*⟩
⟨*assert_attribute*⟩ ::= `:objective` ⟨*symbol*⟩ | `:weight` ⟨*term*⟩

Info flags

⟨*info_flag*⟩ ::= `:limit-optimal` | `:unbounded`

Command options

⟨*option*⟩ ::= `:enable-omt` ⟨*b_value*⟩

Commands

⟨*command*⟩ ::= `(` `define-objective` ⟨*symbol*⟩ ⟨*objective-kind*⟩ ⟨*term*⟩ ⟨*objective_attribute*⟩* `)`
| `(` `define-multi-objective` ⟨*symbol*⟩ ⟨*m-objective-kind*⟩ ⟨*symbol*⟩$^+$ ⟨*attribute*⟩* `)`
| `(` `define-maxsmt-objective` ⟨*symbol*⟩ ⟨*strategy_attribute*⟩* `)`
| `(` `assert-soft` ⟨*term*⟩ ⟨*assert_attribute*⟩$^+$ `)`
| `(` `optimize-sat` ⟨*symbol*⟩ ⟨*assumption_attribute*⟩* ⟨*attribute*⟩* `)`
| `(` `optimize-sat-next` `)`
| `(` `get-value` `(` ⟨*symbol*⟩$^+$ `)` `)`
| `(` `get-model` `)`
| `(` `get-info` ⟨*symbol*⟩$^?$ ⟨*info_flag*⟩ `)`

Command responses

⟨*optimize_sat_response*⟩ ::= `optimal` | `limit-optimal` | `non-optimal` | `unbounded` | `unsat` | `unknown`
⟨*optimize_sat_next_response*⟩ ::= ⟨*optimize_sat_response*⟩
⟨*get_info_response*⟩ ::= `(` ⟨*info_response*⟩$^+$ `)`
⟨*info_response*⟩ ::= `:limit-optimal` ⟨*term*⟩ | `:unbounded` ⟨*term*⟩

Figure 1: OMT extension to the SMT-LIB language: tokens, identifiers, attributes, info flags, command options, commands, and command responses. Some entries that are identical to those in the SMT-LIB standard are listed here for convenience. All other entries are assumed to extend those in the SMT-LIB standard.

3.1. Defining Objectives

We start by describing two commands, `define-objective` and `define-multi-objective`, which enable the definition of single- and multi-objective optimization problems, respectively.

Single Objective The `define-objective` command defines a single objective by specifying a name

for the objective, the kind of the objective, either OBJECTIVE_MIN or OBJECTIVE_MAX, the objective term to be optimized, and one or more optional instances of *objective_attribute*. The *objective_attribute* list can include at most one :strategy attribute, any number of arbitrary attributes, at most one of each of the :order, :lower, and :upper attributes, and zero or more :assumption attributes.

Lower and upper bounds for the objective can be specified using the :lower and :upper attributes. The provided term must have the same sort as the objective term, and it must denote a value for that sort. Bounds are *inclusive* and are used to inform certain strategies (see below). The :assumption attribute can be followed by any arbitrary formula. Bounds and assumptions are required to be satisfied by any solution found for the objective. As with check-sat-assuming, assumptions are local to queries involving their accompanying objective (i.e., they are pushed into the context when optimizing the corresponding objective and popped once that optimization completes). This mechanism is particularly useful when working with multiple objectives, each with their own relevant assumptions.

If an *order_attribute* is given, it must be a function symbol f that takes two arguments of sort τ and returns Bool, where τ is the sort of the objective term. The function f should either be a built-in symbol or have been defined using define-fun, and the term $(f\ x\ y)$, for variables x and y of sort τ, must define a strict partial order. If the :order attribute is omitted, then for the following sorts, a default is used. The standard arithmetic less-than operator < is the default for Int or Real sorts, the bvult operator for bitvector sorts, fp.lt for floating-point sorts, and str.< for string sorts. For other sorts, if no order is provided, solvers may either use a solver-specific default or respond with unsupported.

Finally, a *strategy_attribute* can be specified. Strategies include STRATEGY_LINEAR and STRATEGY_BINARY. The first instructs the solver to use a linear strategy, meaning that it repeatedly tries to improve the value of the objective term simply by adding a constraint that it be better than the previous value. The binary strategy uses a binary search, which can be more efficient than linear search in some cases. If upper and lower bounds are not provided, the solver can use any method to establish a starting point for the search. Solver-specific strategy attributes are also allowed.

This command defines a GOMT problem $\langle t, \prec, \phi \rangle$, where t is the given objective term, $\prec$ is the given or default order, if the kind is OBJECTIVE_MIN, and the converse of the given or default order, if the kind is OBJECTIVE_MAX. The constraint ϕ is the conjunction of all bounds and assumptions..

Example 1. *Suppose we want to find the maximum value that an unsigned 8-bit bitvector can take, subject to the constraint that its value must be even—that is, its least significant bit must be* 0*—and its most significant bit must be* 1.

We present three solutions. The first version includes the evenness constraint as part of the objective, uses a define-fun command to define the order, and specifies that a binary search should be used. It enforces that the most significant bit must be 1 by specifying lower and upper bounds for the search.

```
(declare-const bv_var (_ BitVec 8))

(define-fun ord ((x (_ BitVec 8)) (y (_ BitVec 8))) Bool
                (bvult x y))

(define-objective obj1a OBJECTIVE_MAX bv_var
                        :order ord
                        :assumption (= (bvand bv_var #b00000001) #b00000000)
                        :strategy STRATEGY_BINARY
                        :lower #b10000000
                        :upper #b11111111)
```

The second version uses a linear strategy, specifically specifies that bvult is to be used for the order, and moves all constraints to background assertions.

```
(declare-const bv_var (_ BitVec 8))

(assert (= (bvand bv_var #b00000001) #b00000000))
(assert (= ((_ extract 7 7) bv_var) #b1))

(define-objective obj1b OBJECTIVE_MAX bv_var
                        :order bvult
                        :strategy STRATEGY_LINEAR)
```

The final version is the same as the previous one, except that we don't specify the order at all. The solver uses `bvult` for the order since that is the default for bitvector sorts.

```
(declare-const bv_var (_ BitVec 8))

(assert (= (bvand bv_var #b00000001) #b00000000))
(assert (= ((_ extract 7 7) bv_var) #b1))

(define-objective obj1c OBJECTIVE_MAX bv_var
                        :strategy STRATEGY_LINEAR)
```

Example 2. *Consider a single-objective optimization problem using linear real arithmetic: given the line* $3x + 5y + 1 = 0$ *and an interval constraint* $x \in [-3, 3]$*, find the smallest possible value of* y *such that the point* (x, y) *lies on the line and satisfies the interval constraint on* x*.*

We can encode the example as follows. Again, no order is specified, so the solver uses the arithmetic less-than operator. We also do not specify a strategy, so the solver will use its default strategy.

```
(declare-const x () Real)
(declare-const y () Real)

(assert (= (+ (* 3 x) (* 5 y) 1) 0))
(assert (and (>= x (- 3)) (<= x 3)))

(define-objective obj2 OBJECTIVE_MIN y)
```

Multi-Objective The `define-multi-objective` command defines a multi-objective optimization problem. The arguments are: a name for the objective, a multi-objective kind, and a sequence of previously declared objectives. Solver-specific attributes can also be optionally supplied as arguments.

This command also defines a GOMT problem $\langle t, \prec, \phi \rangle$, formed by composing the objectives provided according to the multi-objective kind. A full description of how this is done is given in [40], but we provide an informal overview here. The objective t is typically a tuple of the individual objectives, the constraint ϕ is the conjunction of the constraints from the individual objectives, and the order $\prec$ is determined by the multi-objective kind. If the kind is `OBJECTIVE_LEX`, then $\prec$ is the lexicographic tuple-order induced by the orders in the provided objectives. If the kind is `OBJECTIVE_PARETO`, then $\prec$ is defined as: at least one value in the tuple is better while none gets worse. If the kind is `OBJECTIVE_MINMAX`, then $\prec$ is defined as reducing the maximum value (`OBJECTIVE_MAXMIN` is similarly defined as increasing the minimum value). Finally, `OBJECTIVE_BOX` simply optimizes each of the objectives individually and can be seen as a special case of `OBJECTIVE_PARETO`.

Example 3. *Let* `obj3str` *and* `obj3lia` *be two single objectives over the theories of strings and linear integer arithmetic, respectively. The objective* `obj3str` *maximizes the string* `s` *according to the lexicographic order using the linear search strategy, while the objective* `obj3lia` *minimizes the integer* `int_ub_len_s` *using the binary search strategy with an upper bound of* `3`*. We also wish for the integer variable* `int_ub_len_s` *to serve as an upper bound on the length of the string* `s`*. For simplicity, assume that strings are over characters ranging only from* `'a'` *to* `'z'`*.*

We can encode the problem of finding a Pareto-optimal value for the two objectives as follows.

```
(declare-const s String)
(declare-const int_ub_len_s Int)

(assert (< (str.len s) int_ub_len_s))

(define-objective obj3lia OBJECTIVE_MIN int_ub_len_s
                          :strategy STRATEGY_BINARY
                          :upper 3)

(define-objective obj3str OBJECTIVE_MAX s
                          :strategy STRATEGY_LINEAR)

(define-multi-objective obj3par OBJECTIVE_PARETO obj3lia obj3str)
```

Note that we provide an upper bound for the first objective but no lower bound. In such cases, the solver can either infer a lower bound (in this case, a lower bound of 1 can be inferred from the first assertion), use heuristics to attempt to guess and check a lower bound, or use a different search strategy. There are three Pareto-optimal solutions to this problem: {`int_ub_len_s` $\to$ 1, `s` $\to$ ""}, {`int_ub_len_s` $\to$ 2, `s` $\to$ "z"}, and {`int_ub_len_s` $\to$ 3, `s` $\to$ "zz"}.

We next encode the problem of finding a lexicographically minimal solution.

```
(declare-const s String)
(declare-const int_ub_len_s Int)

(assert (< (str.len s) int_ub_len_s))
(assert (str.in_re s (re.* (re.range ""a"" ""z""))))

(define-objective obj3lia OBJECTIVE_MIN int_ub_len_s
                        :strategy STRATEGY_BINARY
                        :upper 3)

(define-objective obj3str OBJECTIVE_MAX s
                        :strategy STRATEGY_LINEAR)

(define-multi-objective obj3lex OBJECTIVE_LEX obj3lia obj3str)
```

The order in which we provide the objectives in the `define-multi-objective` command determines the lexicographic order used when optimizing. The unique solution to this problem is {`s` $\to$ "", `int_ub_len_s` $\to$ 1}.

Box optimization can be used to solve `obj3lia` and `obj3str` independently as follows.

```
(declare-const s String)
(declare-const int_ub_len_s Int)

(assert (< (str.len s) int_ub_len_s))

(define-objective obj3lia OBJECTIVE_MIN int_ub_len_s
                          :strategy STRATEGY_BINARY
                          :upper 3)

(define-objective obj3str OBJECTIVE_MAX s
                          :strategy STRATEGY_LINEAR)

(define-multi-objective obj3box OBJECTIVE_BOX obj3lia obj3str)
```

Here, `obj3str` is *unbounded*, meaning that for every string s, there is a larger string satisfying all constraints. This is possible because the upper bound constraint `:upper 3` applies only to `obj3lia`, meaning it does not have to be satisfied when optimizing `obj3str`, since with *box* optimization, each objective is optimized completely independently. On the other hand, `obj3lia` has a unique optimal solution: {`int_ub_len_s` $\to$ 1, `s` $\to$ ""}.

Example 4. *Given two lines $x + y + 1 = 0$ and $x - y + 2 = 0$ and a circle $x^2 + y^2 = 1$, find a point (x, y) on the circle that is as far as possible from the closer of the two lines.*

We can encode this as a MaxMin problem as follows. To simplify the objective terms, we use the *squared* distance from the point (x, y) to each line as the optimization objective.

```
(declare-const x Real)
(declare-const y Real)

(assert (= (+ (* x x) (* y y)) 1))

(define-objective obj4a OBJECTIVE_MIN (/ (* (+ x y 1) (+ x y 1)) 2))

(define-objective obj4b OBJECTIVE_MIN (/ (* (+ x (* -1 y) 2) (+ x (* -1 y) 2)) 2))

(define-multi-objective obj4 OBJECTIVE_MAXMIN obj4a obj4b)
```

The exact optimal solution to this problem is at the point $(x, y) = (\sqrt{3}/2, 1/2)$. The value of `obj4` at this point is `1/2+3`$\sqrt{3}$`/4`. Since these are irrational values, they cannot be exactly represented using

rational constants. However, an approximation can be computed and returned—this is discussed further in Section 4, which covers examples of various OMT command responses.

Note that `define-multi-objective` does not permit the inclusion of an *objective_attribute*. Strategies and assumptions can only be provided at the single objective level. However, solver-specific extensions can provide additional attributes at the multi-objective definition stage.

MaxSMT The `define-maxsmt-objective` command initiates the definition of a MaxSMT objective by assigning it a name. The semantics is that a hidden GOMT objective $\langle t, \prec, \phi \rangle$ is created, where $\prec$ is arithmetic greater-than, `>`, and ϕ is $True$. Initially, t is set to 0.0. However, it changes every time an `assert-soft` command is issued.

We adopt the `assert-soft` command from the OptiMathSAT OMT syntax extension [38]. The `assert-soft` command takes a formula, exactly one *assert_attribute* of the form `:objective`, which specifies the name of the MaxSMT objective to associate this formula with, and an optional weight, specified using `:weight`. The weight is 1.0 by default if not provided explicitly. The semantics of `(assert-soft a :objective obj :weight w)` is that the objective term t for the MaxSMT objective `obj` is updated to $t + ite(a, w, 0.0)$. Note that incremental MaxSMT can be supported by using `push` and `pop` and/or by adding soft constraints between queries.

It is important to note that the objective term in a MaxSMT objective is not explicitly shown but is implicitly associated with `obj` and internally constructed by the solver. A MaxSMT definition may also include strategy attributes, which are applied as usual, but to the hidden GOMT objective.

Example 5. *Consider a MaxSMT example with* $x \geq 0$ *and* $y \geq 0$ *as hard constraints and* $4x + y - 4 \geq 0$ *and* $2x + 3y - 6 \geq 0$ *as soft constraints, with the weights* 2.0 *and* 1.0*, respectively.*

Recall that the goal is to satisfy all hard constraints and as many soft constraints as possible. The problem can be encoded as follows.

```
(declare-const x Int)
(declare-const y Int)

(assert (>= x 0)
(assert (>= y 0)

(define-maxsmt-objective obj5)

(assert-soft (>= (- (+ (* 4 x) y) 4) 0) :objective obj5 :weight 2.0)
(assert-soft (>= (- (+ (* 2 x) (* 3 y)) 6) 0) :objective obj5)
```

A solution to this problem is the point $(x, y) = (1, 2)$, which satisfies both soft constraints. The hidden objective thus has the maximum possible value of 3. There are infinitely many solutions which achieve this optimal value.

The dual of MaxSMT is MinSMT, in which we want to satisfy as *few* soft assertions as possible. We do not explicitly support MinSMT objectives. However, a MinSMT problem can easily be converted into a MaxSMT problem simply by negating all of the soft assertions.

3.2. Optimization Commands and Solver Responses

The `optimize-sat` command takes as an argument the name of an objective and instructs the solver to find an optimal solution for that objective. Zero or more instances of *assumption_attribute* provide additional assumptions that must be satisfied by any solution. Additional solver-specific attributes may also be provided. The solver should aim to compute an optimal solution for the GOMT problem associated with the objective name that also satisfies all assertions currently in the context as well as any bounds and assumptions associated with the objective or assumptions provided as arguments to the `optimize-sat` command. If one or more search strategies is associated with the objective, the solver should follow those strategies during its search for an optimal solution if possible.

There are several possible responses to an `optimize-sat` command:

- `optimal` indicates that the solver has found an optimal solution to the GOMT problem for the specified objective.
- `limit-optimal` indicates that the solver has found a solution to the GOMT problem. The solution found is not optimal, but (i) no optimal solution exists; and (ii) the amount that the found solution can be improved is bounded.
- `unbounded` indicates that the solver has found a solution and also determined that for every solution to the given problem, a solution exists that is better by more than any bounded amount.
- `non-optimal` indicates that the solver found a solution but not an optimal solution, and the solver was not able to determine the problem to be in either the `limit-optimal` or `unbounded` categories.
- `unsat` indicates that the constraints of the problem alone (without considering optimization) are unsatisfiable
- `unknown` indicates that the solver is unable to find a solution, is unable to determine that the problem is `unsat`, and is unable to further categorize the problem.

If the most recent optimized objective is of kind `OBJECTIVE_BOX`, then instead of returning a single response, the solver returns a list of responses, one for each boxed objective.

The `optimize-sat-next` command is available immediately after a successful (response of `optimal`) call to `optimize-sat` or `optimize-sat-next`. It instructs the solver to attempt to find an optimal solution to the same problem it just solved (with the same constraints) that is different from those computed so far, if any. If there are no additional optimal solutions, it returns `unsat`. Otherwise, the set of possible responses are the same as above.

Anytime Optimization The `optimize-sat` command can, optionally, respect SMT-LIB's resource limiting options, such as `:reproducible-resource-limit`. If the solver's optimization routine supports this option, setting it to 0 disables it. Setting it to a non-zero numeral n, causes the `optimize-sat` command to terminate within a bounded amount of time dependent on n. As in SMT solving, the internal implementation of the `:reproducible-resource-limit` option and its relation to run time or other concrete resources can be solver-specific. However, if the solver cannot find an optimal solution within the resource limit and cannot determine whether the problem is unsatisfiable, unbounded, or limit-optimal, then `optimize-sat` must return either `unknown` or `non-optimal`. It returns `unknown` if no solution (even a non-optimal one) has been found, and returns `non-optimal` if some solution has been found. In the latter case, the best solution found so far can be retrieved upon request. This behavior is in line with both standard SMT solving with resource limits and MaxSAT anytime solving.

3.3. Interpreting Results

We propose adding or modifying the three `get` commands described below.

The `get-value` command is overloaded to take as an argument a list of objective names. This version of `get-value` is available after a call to `optimize-sat` or `optimize-sat-next` that returns anything other than `unsat` or `unknown`. It returns a list of pairs, each of which contains an objective name as the first element of the pair and the corresponding value of that objective's objective term in the computed solution as the second element of the pair.

The `get-model` command is similarly available whenever the most recent invocation of `optimize-sat` or `optimize-sat-next` returns something other than `unsat` or `unknown`. It returns a representation of the computed model that is a solution to the optimization problem, using the same format as is used in the SMT-LIB 2.7 standard. There is one exception to this. If the most recent optimized objective is of kind `OBJECTIVE_BOX`, then `get-model` instead returns a list of pairs. The first element of each pair is the name of one of the boxed objectives, and the second element is the model for that objective.

The `get-info` command can be used to query the solver for additional information about the most recent optimization attempt if that attempt returned either `:limit-optimal` or `:unbounded`. In addition, if that attempt was for an objective of kind `OBJECTIVE_BOX`, then the first argument of `get-info` is

the name of the sub-objective (within the set of boxed objectives) for which information is requested. Otherwise, that argument is omitted. In either case, it always takes an argument specifying the value returned, indicating that more information is desired about that value. There are two possible values: `:limit-optimal` and `:unbounded`.

In the first case, issuing the command `get-info :limit-optimal` returns an explanation of what the solver knows about the limit-optimality, such as some value that is approached in the limit by the objective. In the second case, issuing the command `get-info :unbounded` provides details on the cause of unboundedness, such as whether the solver can determine if the objective term tends toward positive or negative infinity. In both cases, the solver should return a term, and the specific format of the term is solver-specific (i.e., it could be a string literal or it could have a richer term structure).

4. Examples

In this section, we show several full examples illustrating different commands and responses. We first revisit Example 1.

```
(set-logic QF_BV)
(set-option :produce-models true)
(set-option :enable_omt true)
(declare-const bv_var (_ BitVec 8))

(assert (= (bvand bv_var #b00000001) #b00000000))
(assert (= ((_ extract 7 7) bv_var) #b1))

(define-objective obj1c OBJECTIVE_MAX bv_var
                        :strategy STRATEGY_LINEAR)
(optimize-sat obj1c)
(get-model)
```

The output of a successful run is shown below.

```
optimal
((obj1c #b11111110))
(
(define-fun bv_var () (_ BitVec 8) #b11111110)
)
```

The initial response is `optimal`. The result of the call to `get-value` shows that the objective term of `obj1c` has the value `#b11111110` in the optimal solution. And the call to `get-model` simply returns the definition of the objective term, since it is the only user-declared symbol.

We next consider an example that tries to minimize the value of $1/x$.

```
(set-logic QF_LRA)
(set-option :produce-models true)
(set-option :enable_omt true)
(declare-const x Real)

(assert (> x 0))

(define-objective obj6 OBJECTIVE_MIN (/ 1 x))

(optimize-sat obj6)
(get-value (obj6))
(get-model)
(get-info :limit-optimal)
```

A possible output from a solver is shown below.

```
limit-optimal
((obj6 0.0001))
(
(define-fun x () Real) 10000.0)
)
(:limit-optimal "As x approaches infinity, the objective term approaches 0.")
```

Since the objective approaches but never reaches 0, the call to `optimize-sat` returns `limit-optimal`. In this case, the solver finds a model that is within 10^{-4} of the limit value. The request for more information returns a string explaining the limit.

We next revisit Example 4 with additional commands below.

```
(set-logic QF_NRA)
(set-option :produce-models true)
(set-option :enable_omt true)
(declare-const x Real)
(declare-const y Real)

(assert (= (+ (* x x) (* y y)) 1))

(define-objective obj4a OBJECTIVE_MIN (/ (* (+ x y 1) (+ x y 1)) 2))

(define-objective obj4b OBJECTIVE_MIN (/ (* (+ x (* -1 y) 2) (+ x (* -1 y) 2)) 2))

(define-multi-objective obj4 OBJECTIVE_MAXMIN obj4a obj4b)

(optimize-sat obj4)
(get-value (obj4))
(get-model)
```

Recall that the exact optimal solution is when $(x, y) = (\sqrt{3}/2, 1/2)$. However, since the value of x is an irrational, we cannot represent it precisely. The value of `obj4` at this point is also irrational ($1/2+3\sqrt{3}/4$). This is another example where a response of `limit-optimal` would be appropriate. However, a solver that is unable to figure this out might instead return `non-optimal` and then return a decimal approximation for `x`, `y`, and the objective term. One possible solver output is shown below.

```
non-optimal
((obj4 (800/289, 1681/578)))
(
(define-fun x () Real (/ 15 17))
(define-fun y () Real (/ 8 17))
)
```

Here, x, y, and the objective term are approximated with rational constants.

We next show an example with an `unbounded` objective.

```
(set-logic QF_LRA)
(set-option :produce-models true)
(set-option :enable_omt true)
(declare-const x Real)

(assert (> x 0))

(define-objective obj7 OBJECTIVE_MAX x)

(optimize-sat obj7)
(get value (obj7))
(get-model)
(get-info :unbounded)
```

A possible output is given below.

```
unbounded
((obj7 100000.0))
(
(define-fun x () Real) 100000.0)
)
(:unbounded "The objective term approaches positive infinity as x approaches positive infinity.")
```

The final example illustrates the use of the `optimize-sat-next` command with Example 3. It outputs all three Pareto-optimal solutions, each preceded by the `optimal` response.

```
(set-logic QF_SLIA)
(set-option :produce-models true)
(set-option :enable_omt true)
(declare-const s String)
(declare-const int_ub_len_s Int)
```

```
(assert (< (str.len s) int_ub_len_s))
(assert (str.in_re s (re.* (re.range ""a"" ""z""))))

(define-objective obj3lia OBJECTIVE_MIN int_ub_len_s
                          :upper 3)

(define-objective obj3str OBJECTIVE_MAX s)

(define-multi-objective obj3par OBJECTIVE_PARETO obj3lia obj3str))

(optimize-sat obj3par)
(get-value (obj3par))
(optimize-sat-next)
(get-value (obj3par))
(optimize-sat-next)
(get-value (obj3par))
(optimize-sat-next)
```

A possible output may be like the one below.

```
optimal
((obj3par (1, "")))
(
(define-fun int_ub_len_s () Int 1)
(define-fun s () String "")
)
optimal
((obj3par (2, "z")))
(
(define-fun int_ub_len_s () Int 2)
(define-fun s () String "z")
)
optimal
((obj3par (3, "zz")))
(
(define-fun int_ub_len_s () Int 3)
(define-fun s () String "zz")
)
unsat
```

5. Conclusion

The lack of a standardized SMT-LIB syntax for OMT remains a significant obstacle to the broad adoption and interoperability of OMT tools. To address this, we have proposed a concrete extension of the SMT-LIB v2.7 standard that can express all of the optimization modulo theories problems from the OMT literature. These extensions enhance both expressiveness and flexibility, making it possible to encode more complex optimization constructs, such as optimization with respect to any definable partial order, as well as arbitrary combinations of single- and multi-objective problems found in the literature, including MaxSMT and MinSMT. Future work will focus on refining the syntax through community feedback and assembling a representative suite of OMT benchmarks. Ultimately, this effort aims to enable more robust, reusable, and comparable OMT solutions, supporting the growing demand for optimization in formal methods, automated reasoning, and among end users.

Acknowledgments This work was supported in part by the Stanford Agile Hardware Center, the Stanford Center for Automated Reasoning, and by the National Science Foundation (grant 2006407).
Declaration on Generative AI The authors used ChatGPT and Overleaf for spell check and targeted edits. After using these tools, the authors reviewed and edited content as needed and take full responsibility for the publication's content.

References

[1] C. W. Barrett, R. Sebastiani, S. A. Seshia, C. Tinelli, Satisfiability modulo theories, in: Handbook of Satisfiability, volume 185 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2009, pp. 825–885.

[2] T. Liu, S. S. Tyszberowicz, B. Beckert, M. Taghdiri, Computing exact loop bounds for bounded program verification, in: K. G. Larsen, O. Sokolsky, J. Wang (Eds.), Dependable Software Engineering. Theories, Tools, and Applications - Third International Symposium, SETTA 2017, Changsha, China, October 23-25, 2017, Proceedings, volume 10606 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 147–163.

[3] S. Ratschan, Simulation based computation of certificates for safety of dynamical systems, arXiv preprint arXiv:1707.00879 (2017). arXiv:1707.00879.

[4] L. Candeago, D. Larraz, A. Oliveras, E. Rodríguez-Carbonell, A. Rubio, Speeding up the constraint-based method in difference logic, in: N. Creignou, D. Le Berre (Eds.), Theory and Applications of Satisfiability Testing – SAT 2016, Springer International Publishing, Cham, 2016, pp. 284–301.

[5] J. Jiang, L. Chen, X. Wu, J. Wang, Block-wise abstract interpretation by combining abstract domains with SMT, in: A. Bouajjani, D. Monniaux (Eds.), Verification, Model Checking, and Abstract Interpretation - 18th International Conference, VMCAI 2017, Paris, France, January 15-17, 2017, Proceedings, volume 10145 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 310–329.

[6] G. E. Karpenkov, Finding inductive invariants using satisfiability modulo theories and convex optimization, Theses, Université Grenoble Alpes, 2017.

[7] P. Yao, Q. Shi, H. Huang, C. Zhang, Program analysis via efficient symbolic abstraction, Proceedings of the ACM on Programming Languages 5 (2021).

[8] J. Henry, M. Asavoae, D. Monniaux, C. Maïza, How to compute worst-case execution time by optimization modulo theory and a clever encoding of program semantics, SIGPLAN Not. 49 (2014) 43–52.

[9] C. Bertolissi, D. R. dos Santos, S. Ranise, Solving multi-objective workflow satisfiability problems with optimization modulo theories techniques, in: E. Bertino, D. Lin, J. Lobo (Eds.), Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies, SACMAT 2018, Indianapolis, IN, USA, June 13-15, 2018, ACM, 2018, pp. 117–128.

[10] N. Paoletti, Z. Jiang, M. A. Islam, H. Abbas, R. Mangharam, S. Lin, Z. Gruber, S. A. Smolka, Synthesizing stealthy reprogramming attacks on cardiac devices, in: Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 13–22.

[11] T. Tarrach, M. Ebrahimi, S. König, C. Schmittner, R. Bloem, D. Nickovic, Threat repair with optimization modulo theories, 2022. arXiv:2210.03207.

[12] F. Erata, R. Piskac, V. Mateu, J. Szefer, Towards automated detection of single-trace side-channel vulnerabilities in constant-time cryptographic code, arXiv preprint arXiv:2304.02102 (2023).

[13] S. S. Craciunas, R. S. Oliver, M. Chmelík, W. Steiner, Scheduling real-time communication in ieee 802.1qbv time sensitive networks, in: Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS '16, Association for Computing Machinery, New York, NY, USA, 2016, p. 183–192.

[14] G. Kovásznai, C. Biró, B. Erdélyi, Puli - a problem-specific omt solver, EasyChair Preprints (2018).

[15] F. Leofante, E. Giunchiglia, E. Ábrahám, A. Tacchella, Optimal planning modulo theories, in: Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence, 2021, pp. 4128–4134.

[16] A. Bit-Monnot, F. Leofante, L. Pulina, E. Ábrahám, A. Tacchella, Smartplan: a task planner for smart factories, arXiv preprint arXiv:1806.07135 (2018). arXiv:1806.07135.

[17] M. Eraşcu, F. Micota, D. Zaharie, Applying optimization modulo theory, mathematical programming and symmetry breaking for automatic deployment in the cloud of component-based applications extended abstract, in: 4th Women in Logic Workshop, 2020, p. 6.

[18] J. Feng, T. Zhang, C. Yi, Reliability-aware comprehensive routing and scheduling in time-sensitive

networking, in: Wireless Algorithms, Systems, and Applications: 17th International Conference, WASA 2022, Dalian, China, November 24–26, 2022, Proceedings, Part II, Springer, 2022, pp. 243–254.
[19] X. Jin, C. Xia, N. Guan, P. Zeng, Joint algorithm of message fragmentation and no-wait scheduling for time-sensitive networks, IEEE/CAA Journal of Automatica Sinica 8 (2021) 478–490.
[20] G. Marchetto, R. Sisto, F. Valenza, J. Yusupov, A. Ksentini, A formal approach to verify connectivity and optimize vnf placement in industrial networks, IEEE Transactions on Industrial Informatics 17 (2021) 1515–1525.
[21] G. Patti, L. L. Bello, L. Leonardi, Deadline-aware online scheduling of tsn flows for automotive applications, IEEE Transactions on Industrial Informatics (2022).
[22] D. Shen, T. Zhang, J. Wang, Q. Deng, S. Han, X. S. Hu, Qos guaranteed resource allocation for coexisting embb and urllc traffic in 5g industrial networks, in: 2022 IEEE 28th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), IEEE, 2022, pp. 81–90.
[23] C. M. Nguyen, R. Sebastiani, P. Giorgini, J. Mylopoulos, Requirements evolution and evolution requirements with constrained goal models, in: International Conference on Conceptual Modeling, 2016.
[24] C. M. Nguyen, R. Sebastiani, P. Giorgini, J. Mylopoulos, Modeling and reasoning on requirements evolution with constrained goal models, in: A. Cimatti, M. Sirjani (Eds.), Software Engineering and Formal Methods - 15th International Conference, SEFM 2017, Trento, Italy, September 4-8, 2017, Proceedings, volume 10469 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 70–86.
[25] C. M. Nguyen, R. Sebastiani, P. Giorgini, J. Mylopoulos, Multi object reasoning with constrained goal model, Requirements Engineering 23 (2018).
[26] I. Gavran, E. Darulova, R. Majumdar, Interactive synthesis of temporal specifications from examples and natural language, Proceedings of the ACM on Programming Languages 4 (2020).
[27] S. Demarchi, M. Menapace, A. Tacchella, Automating elevator design with satisfiability modulo theories, in: 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), 2019, pp. 26–33.
[28] S. Demarchi, A. Tacchella, M. Menapace, Automated design of complex systems with constraint programming techniques., in: CPS Summer School, PhD Workshop, 2019, pp. 51–59.
[29] N. Tsiskaridze, M. Strange, M. Mann, K. Sreedhar, Q. Liu, M. Horowitz, C. W. Barrett, Automating system configuration, in: Formal Methods in Computer Aided Design, FMCAD 2021, New Haven, CT, USA, October 19-22, 2021, IEEE, 2021, pp. 102–111.
[30] A. Tierno, G. Turri, A. Cimatti, R. Passerone, Symbolic encoding of reliability for the design of redundant architectures, in: 2022 IEEE 5th International Conference on Industrial Cyber-Physical Systems (ICPS), 2022, pp. 01–06.
[31] A. Rybalchenko, C. Vuppalapati, Supercharging plant configurations using z3, in: Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 18th International Conference, CPAIOR 2021, Vienna, Austria, July 5–8, 2021, Proceedings, volume 12735, Springer Nature, 2021, p. 1.
[32] D. Park, D. Lee, I. Kang, S. Gao, B. Lin, C.-K. Cheng, Sp&r: Simultaneous placement and routing framework for standard cell synthesis in sub-7nm, in: 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC), 2020, pp. 345–350.
[33] D. Lee, D. Park, C.-T. Ho, I. Kang, H. Kim, S. Gao, B. Lin, C.-K. Cheng, Sp&r: Smt-based simultaneous place-and-route for standard cell synthesis of advanced nodes, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 40 (2020) 2142–2155.
[34] M. Knüsel, Optimizing Declarative Power Sequencing, Master thesis, ETH Zurich, Zurich, 2021-09.
[35] S. Teso, R. Sebastiani, A. Passerini, Structured learning modulo theories, Artificial Intelligence 244 (2017) 166–187.
[36] A. Sivaraman, G. Farnadi, T. Millstein, G. Van den Broeck, Counterexample-guided learning of monotonic neural networks, Advances in Neural Information Processing Systems 33 (2020) 11936–11948.
[37] Z. Bian, F. A. Chudak, W. G. Macready, A. Roy, R. Sebastiani, S. Varotti, Solving sat and maxsat

with a quantum annealer: Foundations and a preliminary report, in: International Symposium on Frontiers of Combining Systems, 2017.

[38] R. Sebastiani, P. Trentin, Optimathsat: A tool for optimization modulo theories, Journal of Automated Reasoning (2015) 1–38.

[39] N. S. Bjørner, A.-D. Phan, νz - maximal satisfaction with z3, in: International Symposium on Symbolic Computation in Software Science, 2014.

[40] N. Tsiskaridze, C. W. Barrett, C. Tinelli, Generalized optimization modulo theories, in: C. Benzmüller, M. J. H. Heule, R. A. Schmidt (Eds.), Automated Reasoning - 12th International Joint Conference, IJCAR 2024, Nancy, France, July 3-6, 2024, Proceedings, Part I, volume 14739 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 458–479. URL: https://doi.org/10.1007/978-3-031-63498-7_27. doi:10.1007/978-3-031-63498-7_27.

[41] R. Nieuwenhuis, A. Oliveras, On SAT Modulo Theories and Optimization Problems, in: A. Biere, C. P. Gomes (Eds.), Theory and Applications of Satisfiability Testing - SAT 2006, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 156–169.

[42] N. Bjørner, A. Phan, νZ - Maximal Satisfaction with Z3, in: 6th International Symposium on Symbolic Computation in Software Science, SCSS 2014, Gammarth, La Marsa, Tunisia, December 7-8, 2014, 2014, pp. 1–9.

[43] R. Sebastiani, S. Tomasi, Optimization Modulo Theories with Linear Rational Costs, ACM Trans. Comput. Logic 16 (2015) 12:1–12:43.

[44] O. V. Roc, Optimization Modulo Theories, Master's thesis, Polytechnic University of Catalonia, https://upcommons.upc.edu/handle/2099.1/14204, 2011.

[45] R. Sebastiani, P. Trentin, Pushing the envelope of optimization modulo theories with linear-arithmetic cost functions, in: C. Baier, C. Tinelli (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2015, pp. 335–349.

[46] R. Sebastiani, S. Tomasi, Optimization in smt with $\mathcal{LA}(\mathbb{Q})$ cost functions, in: B. Gramlich, D. Miller, U. Sattler (Eds.), Automated Reasoning, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 484–498.

[47] R. Sebastiani, P. Trentin, On optimization modulo theories, maxsmt and sorting networks, in: A. Legay, T. Margaria (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2017, pp. 231–248.

[48] A. Cimatti, A. Franzén, A. Griggio, R. Sebastiani, C. Stenico, Satisfiability Modulo the Theory of Costs: Foundations and Applications, in: J. Esparza, R. Majumdar (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 99–113.

[49] A. Nadel, V. Ryvchin, Bit-Vector Optimization, in: TACAS, 2016.

[50] P. Trentin, R. Sebastiani, Optimization modulo the theories of signed bit-vectors and floating-point numbers, J. Autom. Reason. 65 (2021) 1071–1096.

[51] F. Bigarella, A. Cimatti, A. Griggio, A. Irfan, M. Jonás, M. Roveri, R. Sebastiani, P. Trentin, Optimization modulo non-linear arithmetic via incremental linearization, in: B. Konev, G. Reger (Eds.), Frontiers of Combining Systems - 13th International Symposium, FroCoS 2021, Birmingham, UK, September 8-10, 2021, Proceedings, volume 12941 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 213–231.

[52] D. Jackson, H.-C. Estler, D. Rayside, The Guided Improvement Algorithm for Exact, General-Purpose, Many-Objective Combinatorial Optimization (2009).

[53] Y. Li, A. Albarghouthi, Z. Kincaid, A. Gurfinkel, M. Chechik, Symbolic optimization with SMT solvers, in: POPL, 2014.

[54] K. Fazekas, F. Bacchus, A. Biere, Implicit hitting set algorithms for maximum satisfiability modulo theories, in: D. Galmiche, S. Schulz, R. Sebastiani (Eds.), Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, volume 10900 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 134–151.

[55] P. Trentin, Optimization Modulo Theories with OptiMathSAT, Ph.D. thesis, University of Trento,

2019.

[56] F. Contaldo, P. Trentin, R. Sebastiani, From minizinc to optimization modulo theories, and back, in: Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2020), volume 12296 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 148–166. URL: https://arxiv.org/abs/1912.01476. doi:10.1007/978-3-030-58942-4_10.

[57] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, G. Tack, Minizinc: Towards a standard cp modelling language, in: C. Bessière (Ed.), Principles and Practice of Constraint Programming – CP 2007, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 529–543.

[58] MiniZinc Team, Interfacing Solvers to FlatZinc, MiniZinc, 2025. Available at https://docs.minizinc.dev/en/stable/fzn-spec.html.

[59] P. Trentin, fzn2omt: Tools/scripts to convert minizinc/flatzinc to optimization modulo theories (omt) for bclt, optimathsat, or z3 and satisfiability modulo theories (smt) for cvc4, 2025. URL: https://github.com/PatrickTrentin88/fzn2omt, accessed: 2025-05-08.

[60] M. Bofill, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, A. Rubio, The barcelogic smt solver, in: Computer Aided Verification (CAV 2008), volume 5123 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 294–298. URL: https://link.springer.com/chapter/10.1007/978-3-540-70545-1_27. doi:10.1007/978-3-540-70545-1_27.

[61] L. de Moura, N. Bjørner, Z3: An efficient smt solver, in: Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008), volume 4963 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 337–340. URL: https://doi.org/10.1007/978-3-540-78800-3_24. doi:10.1007/978-3-540-78800-3_24.

[62] H. Barbosa, C. W. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, Y. Zohar, cvc5: A versatile and industrial-strength SMT solver, in: D. Fisman, G. Rosu (Eds.), Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I, volume 13243 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 415–442.

[63] C. Barrett, P. Fontaine, C. Tinelli, The SMT-LIB Standard: Version 2.7, Technical Report, Department of Computer Science, The University of Iowa, 2025. Available at www.SMT-LIB.org.